Reasoning About Class Behavior
Using the Bisimulation Technique

No.63 NII Shonan Meeting

Vasileios Koutavas
Trinity College Dublin
joint work with **Mitchell Wand**
Northeastern University

supported, in part, by Science Foundation Ireland grant 13/RC/2094

Trinity College Dublin, Coláiste na Tríonóide, Baile Átha Cliath, The University of Dublin
Northeastern University
sfi Science Foundation Ireland
lero THE IRISH SOFTWARE RESEARCH CENTRE

(Appeared in FOOL/WOOD 2007)

Roadmap

- A Motivating Example
- Contextual Equivalence
- Technology for Proving Equivalences
- Deriving Bisimulations for Equivalence
- An Actual Proof of Equivalence
- Language Extensions

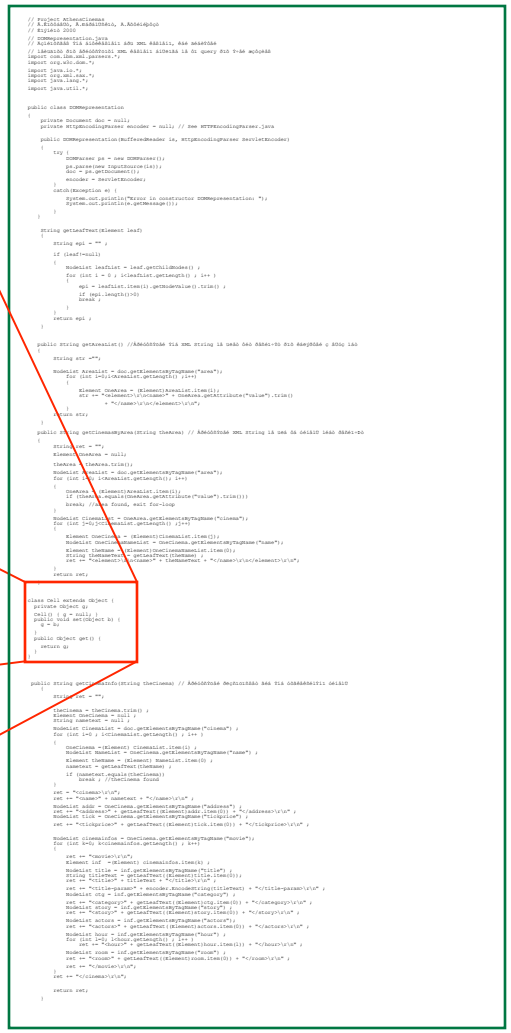
A Cell Class

```
class Cell extends Object {
  private Object g;
  Cell() { g = null; }
  public void set(Object b) {
    g = b;
  }
  public Object get() {
    return g;
  }
}
```

A Cell Class

```
class Cell extends Object {
   private Object g;
   Cell() { g = null; }
   public void set(Object b) {
      g = b;
   }
   public Object get() {
      return g;
   }
}
```

Some Program



A Cell Class

Some Program

A Cell Class

```
class Cell extends Object {
  private Object g1, g2;
  private int cnt;
  Cell() {
    g1 = null; g2 = null; cnt = 0;
  }
  public void set(Object b) {
    cnt = cnt + 1;
    if ((cnt % 2) == 0)
      g1 = b;
    else
      g2 = b;
  }
  public Object get() {
    if ((cnt % 2) == 0)
      return g1;
    else
      return g2;
  }
}
```

A Cell Class

```
class Cell extends Object {
  private Object g1, g2;
  private int cnt;
  Cell() {
    g1 = null; g2 = null; cnt = 0;
  }
  public void set(Object b) {
    cnt = cnt + 1;
    if ((cnt % 2) == 0)
      g1 = b;
    else
      g2 = b;
  }
  public Object get() {
    if ((cnt % 2) == 0)
      return g1;
    else
      return g2;
  }
}
```

A Cell Class

```
class Cell extends Object {
  private Object g1, g2;
  private int cnt;
  Cell() {
    g1 = null; g2 = null; cnt = 0;
  }
  public void set(Object b) {
    cnt = cnt + 1;
    if ((cnt % 2) == 0)
      g1 = b;
    else
      g2 = b;
  }
  public Object get() {
    if ((cnt % 2) == 0)
      return g1;
    else
      return g2;
  }
}
```

A Cell Class

```
class Cell extends Object {
  private Object g1, g2;
  private int cnt;
  Cell() {
    g1 = null; g2 = null; cnt = 0;
  }
  public void set(Object b) {
    cnt = cnt + 1;
    if ((cnt % 2) == 0)
      g1 = b;
    else
      g2 = b;
  }
  public Object get() {
    if ((cnt % 2) == 0)
      return g1;
    else
      return g2;
  }
}
```

A Cell Class

```
class Cell extends Object {
  private Object g1, g2;
  private int cnt;
  Cell() {
    g1 = null; g2 = null; cnt = 0;
  }
  public void set(Object b) {
    cnt = cnt + 1;
    if ((cnt % 2) == 0)
      g1 = b;
    else
      g2 = b;
  }
  public Object get() {
    if ((cnt % 2) == 0)
      return g1;
    else
      return g2;
  }
}
```

The Same Program

A Cell Class

The Same Program

A Cell Class

A Cell Class

...

...

A Cell Class

Is the behavior the same for **any possible pair** of programs?

…

…

A Cell Class

Is the behavior the same for **any possible pair** of programs?

…

…

Contextual Equivalence

Contextual Equivalence (≡)

- **C** and **C'** are **contextually equivalent** iff:

For any class-table context **CT[ ]**:

CT[C] and **CT[C']** **have the same behavior**

Contextual Equivalence (≡)

- **C** and **C'** are **contextually equivalent** iff:

For any class-table context **CT[ ]**:

$$\mathbf{CT[C]} \Downarrow \text{ iff } \mathbf{CT[C']} \Downarrow$$

Contextual Equivalence (≡)

- **C** and **C'** are **contextually equivalent** iff:

For any class-table context **CT[ ]**:

$$CT[C] \Downarrow \text{ iff } CT[C'] \Downarrow$$

  - The "golden standard of equivalences"
  - Hard to use directly to prove non-trivial equivalences

Technology for Proving Equivalence

- Denotational Semantics
  - Usual Denotational Semantics + Logical Relations [Pitts]
  - Usual Denotational Semantics + Bisimulations [Banerjee&Naumann]
  - Game Semantics + “quotient” [Reynolds, Hyland&Ong]
  - Fully abstract Game Semantics [Abramsky+, Murawski&Tzevelekos,Laird]
- Trace Equivalence [Jeffrey&Rathke,Laird]
- Operational Semantics + Logical Relations [Pitts, Pitts&Stark,Appel&McAllester,Ahmed,Dreyer+]
- Operational Semantics + Bisimulations [Abramsky, Sumii&Pierce, Koutavas&Wand, Sangiorgi+]

Technology for Proving Equivalence

- Denotational Semantics
  - Usual Denotational Semantics + Logical Relations [Pitts]
  - Usual Denotational Semantics + Bisimulations [Banerjee&Naumann]
  - Game Semantics + "quotient" [Reynolds, Hyland&Ong]
  - Fully abstract Game Semantics [Abramsky+, Murawski&Tzevelekos,Laird]
- Trace Equivalence [Jeffrey&Rathke,Laird]
- Operational Semantics + Logical Relations [Pitts, Pitts&Stark,Appel&McAllester,Ahmed,Dreyer+]
- Operational Semantics + Bisimulations [Abramsky, Sumii&Pierce, Koutavas&Wand, Sangiorgi+]

Technology for Proving Equivalence

This talk: Environmental Bisimulations for Java

- Denotational Semantics
  - Usual Denotational Semantics + Logical Relations [Pitts]
  - Usual Denotational Semantics + Bisimulations [Banerjee&Naumann]
  - Game Semantics + "quotient" [Reynolds, Hyland&Ong]
  - Fully abstract Game Semantics [Abramsky+, Murawski&Tzevelekos,Laird]
- Trace Equivalence [Jeffrey&Rathke,Laird]
- Operational Semantics + Logical Relations [Pitts, Pitts&Stark,Appel&McAllester,Ahmed,Dreyer+]
- Operational Semantics + Bisimulations [Abramsky, Sumii&Pierce, Koutavas&Wand, Sangiorgi+]

Technology for Proving Equivalence

- The standard definition of (≡) describes the largest set of equivalent classes

$$(\equiv) \;=\; \{\, (C, C') \mid \ldots \,\}$$

Technology for Proving Equivalence

- The standard definition of (≡) describes the largest set of equivalent classes

$$(\equiv) \;=\; \{\, (C, C') \mid \ldots \,\}$$

- Find a more "convenient" definition for a set R and show:
  - $R \subseteq (\equiv)$ (Soundness of R)
  - $(\equiv) \subseteq R$ (Completeness of R)

Technology for Proving Equivalence

- The standard definition of (≡) describes the largest set of equivalent classes

$$(\equiv) \;=\; \{\, (C, C') \mid \ldots \,\}$$

- Find a more "convenient" definition for a set R and show:
  - $R \subseteq (\equiv)$ (Soundness of R)
  - $(\equiv) \subseteq R$ (Completeness of R)
- Use the definition of R in proofs instead of (≡)

Challenges of the Defined Set R

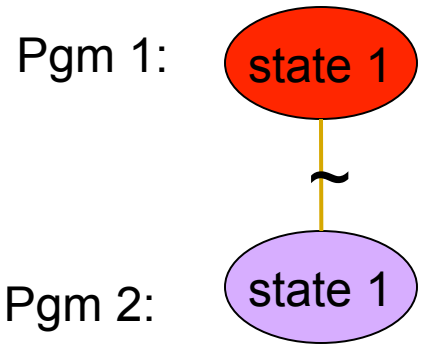
- For a Java-like language R should help reason about:
  - Different **state fields** in related classes
  - **Inheritance** and **(down-)casting** as a way of distinguishing classes by the context
  - Higher-order features—**Callbacks**

Roadmap

- *A Motivating Example*
- *Contextual Equivalence*
- *Technology for Proving Equivalences*
- **Deriving Bisimulations for Equivalence**
- An Actual Proof of Equivalence
- Language Extensions

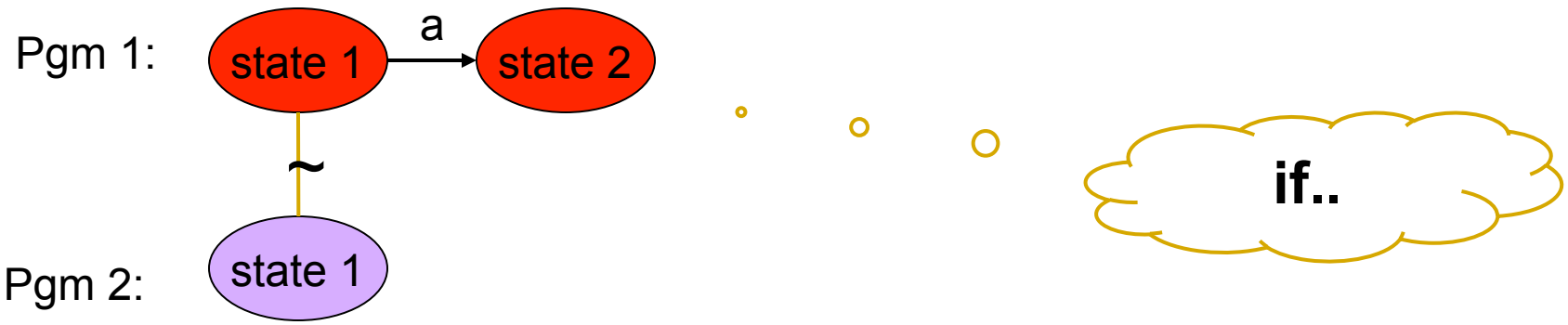
Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



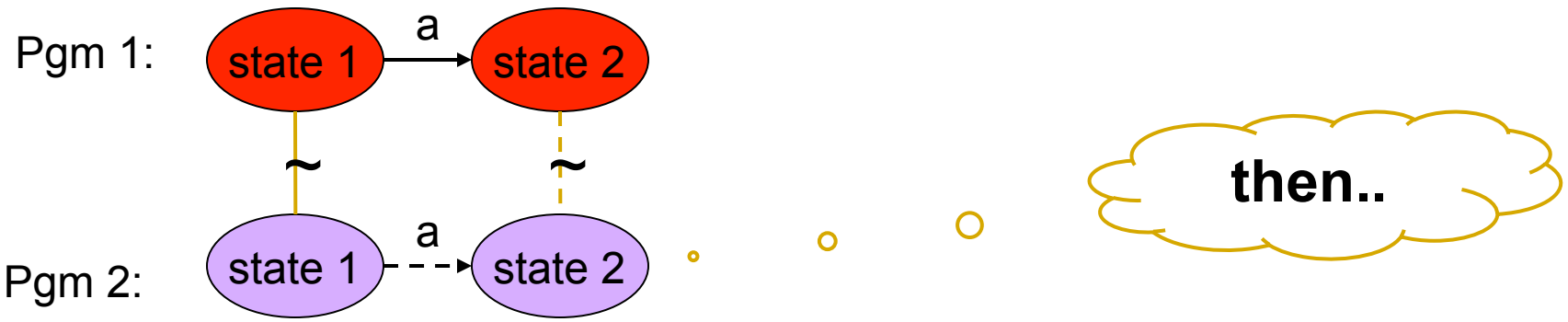
Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



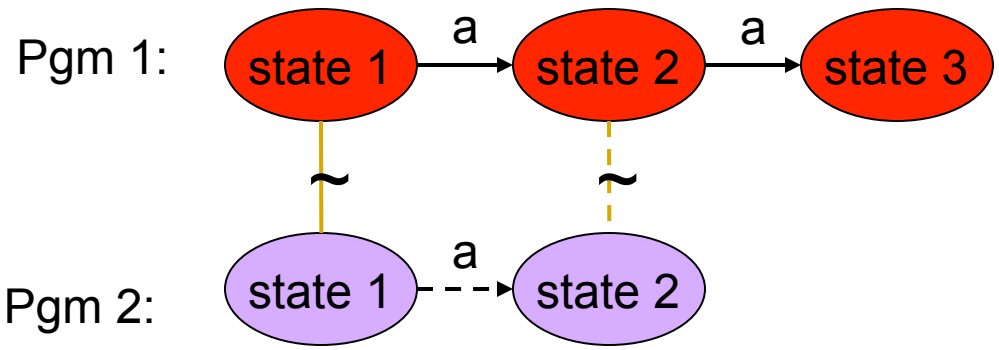
Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



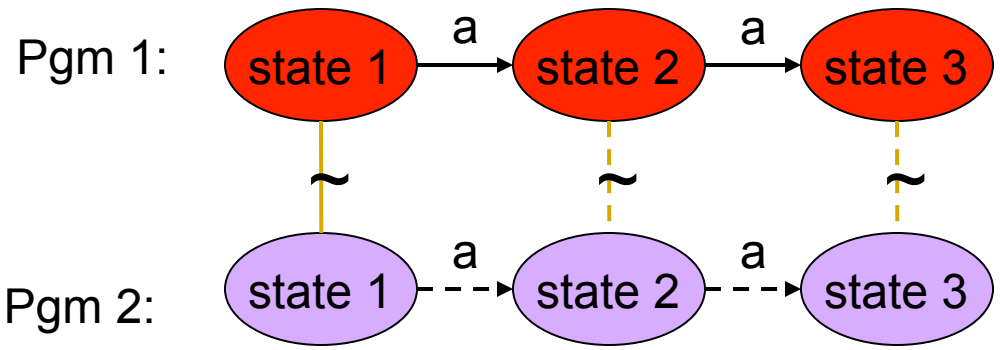
Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



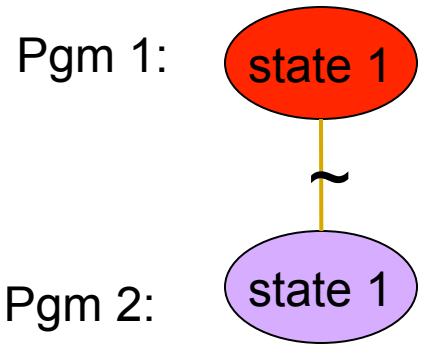
Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



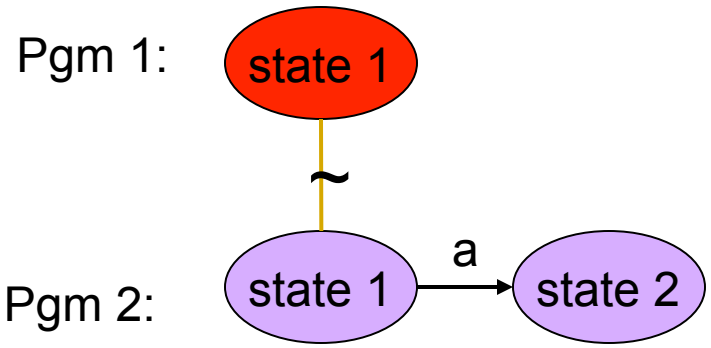
Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



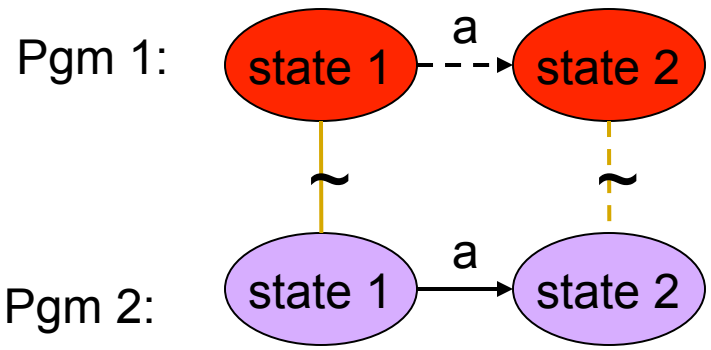
Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



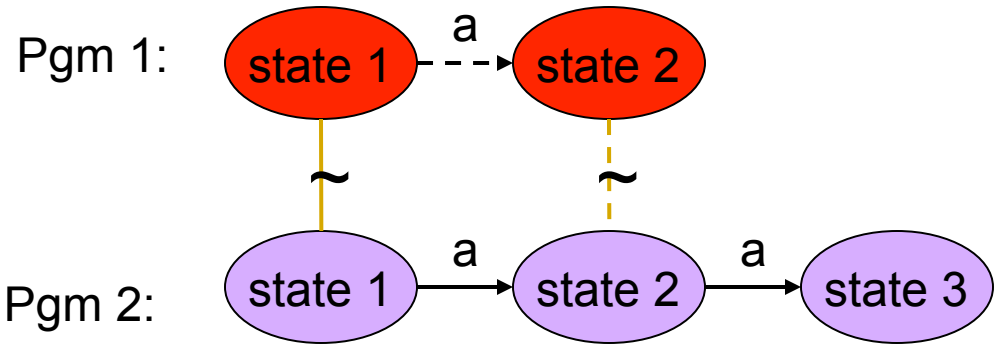
Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



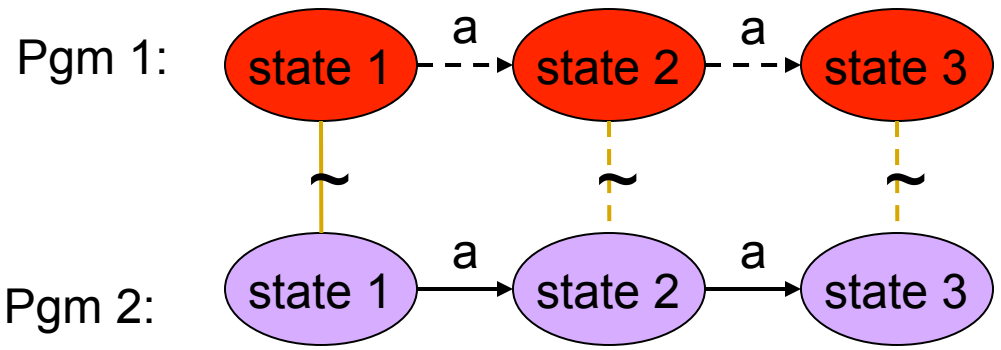
Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



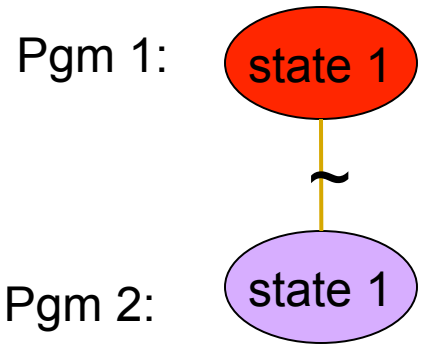
Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



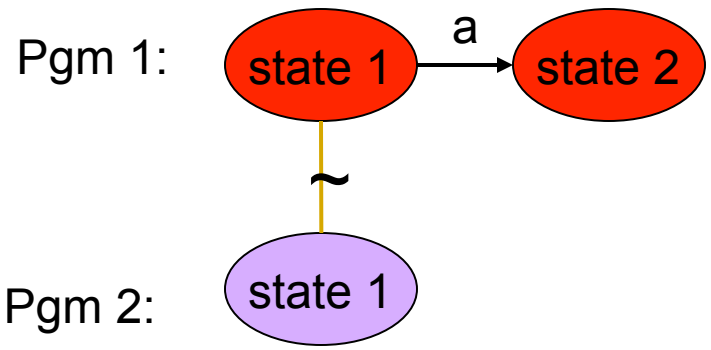
Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



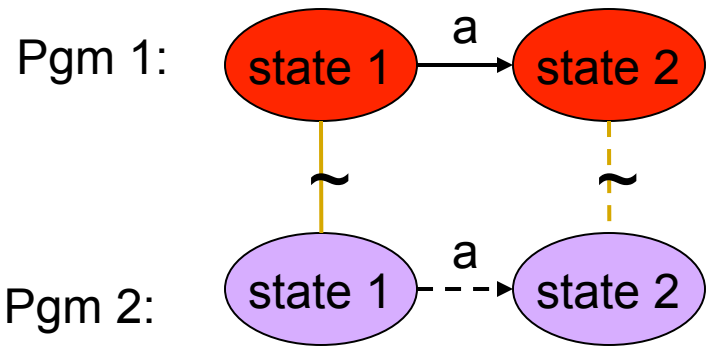
Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



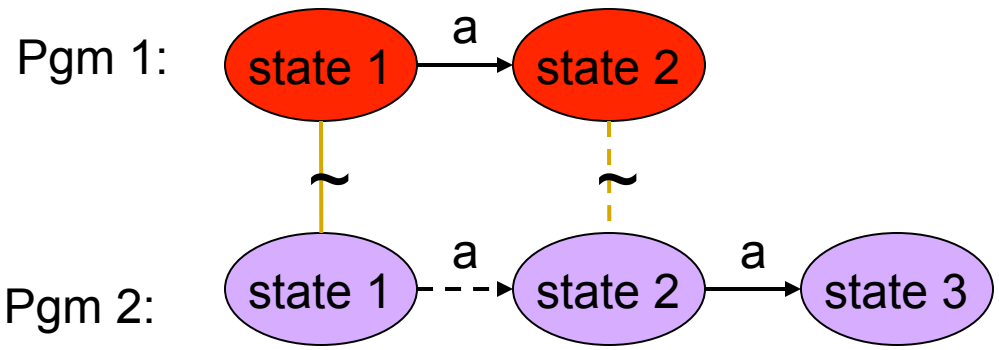
Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



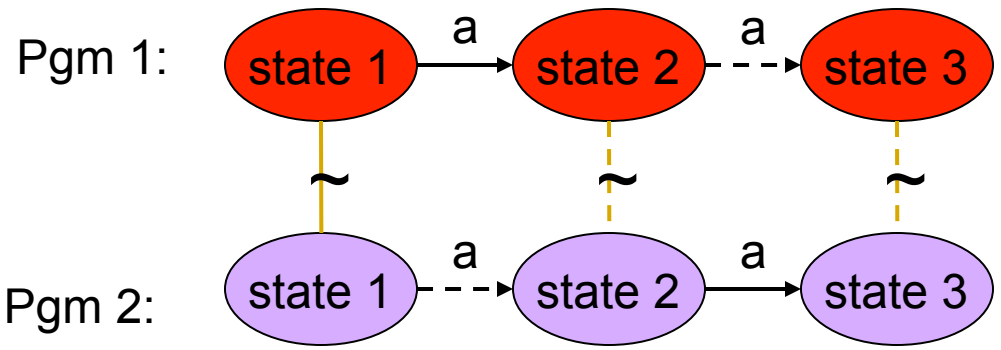
Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



Bisimulations

- Introduced in Concurrent Calculi [Milner], and adapted to functional languages [Abramsky].



Deriving Bisimulations for (≡)

- $(C, C') \in (\equiv)$ iff:

For any class-table context **CT[ ]**:

$$(0, CT[C]) \Downarrow \text{ iff } (0, CT[C']) \Downarrow$$

Deriving Bisimulations for (≡)

- $(C, C') \in (\equiv)$ iff:

For any class-table context **CT[ ]**:

$$(0, CT[C]) \Downarrow \text{ iff } (0, CT[C']) \Downarrow$$

Empty store

Deriving Bisimulations for (≡)

- $(C, C') \in (\equiv)$ iff:

For any class-table context **CT[ ]**:

$$(0, CT[C]) \Downarrow \text{ iff } (0, CT[C']) \Downarrow$$

Define Adequacy (≈) as the largest set s.t.:

- $(s, s', C, C') \in (\approx)$ iff:

For any class-table context **CT[ ]**:

$$(s, CT[C]) \Downarrow t, a \Rightarrow (s', CT[C']) \Downarrow t', a \;\&\; (t, t', C, C') \in (\approx)$$

Deriving Bisimulations for (≡)

- (**C**, **C'**) ∈ (≡) iff:

For any class-table context **CT[ ]**:

(0, CT[C]) ⇓ iff **(0, CT[C'])** ⇓

Define Adequacy (≈) as the large

and the reverse

- (**s**, **s'** , **C**, **C'**) ∈ (≈) iff:

For any class-table context **CT[ ]**:

(s, CT[C]) ⇓ t, a ⇐ (s' , CT[C']) ⇓ t' , a

& (**t**, **t'** , **C**, **C'**) ∈ (≈)

Deriving Bisimulations for (≡)

- (**C**, **C'**) ∈ (≡) iff:

For any class-table context **CT[ ]**:

(0, CT[C]) ⇓ iff **(0, CT[C'])** ⇓

Define Adequacy (≈) as the la

- (**s**, **s'** , **C**, **C'**) ∈ (≈) iff:

For any class-table context **CT[ ]**:

(s, CT[C]) ⇓ **t, a** ⇒ **(s'** , **CT[C'])** ⇓ **t'** , **a**

& (**t**, **t'** , **C**, **C'**) ∈ (≈)

This is a **big-step bisimulation**

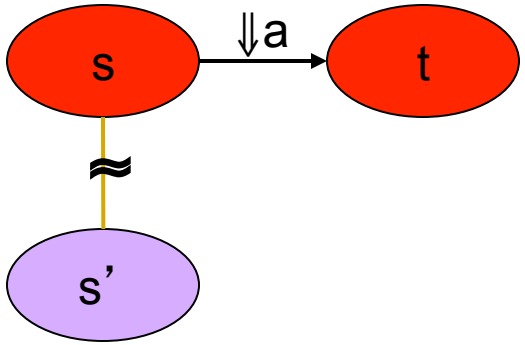
Deriving Bisimulations for (≡)

Define Adequacy as the largest set s.t.:

- $(s, s', C, C') \in (\approx)$ iff:

For any class-table context **CT[ ]**:

$$(s, CT[C]) \Downarrow t, a \Rightarrow (s', CT[C']) \Downarrow t', a$$
$$\& (t, t', C, C') \in (\approx)$$



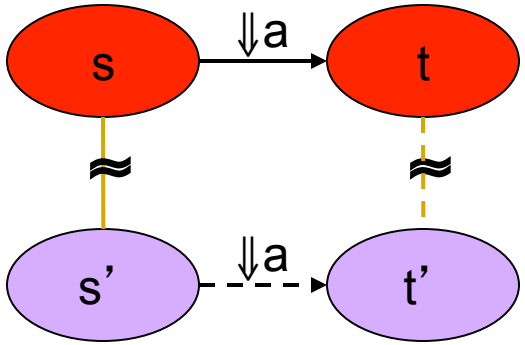
Deriving Bisimulations for (≡)

Define Adequacy as the largest set s.t.:

- $(s, s', C, C') \in (\approx)$ iff:

For any class-table context **CT[ ]**:

$$(s, CT[C]) \Downarrow t, a \Rightarrow (s', CT[C']) \Downarrow t', a$$
$$\& \ (t, t', C, C') \in (\approx)$$



Deriving Bisimulations for (≡)

Define Adequacy as the largest set s.t.:

- $(\mathbf{s}, \mathbf{s'}, \mathbf{C}, \mathbf{C'}) \in (\approx)$ iff:

For any class-table context **CT[ ]**:

$$(\mathbf{s}, \mathbf{CT[C]}) \Downarrow \mathbf{t}, \mathbf{a} \;\Rightarrow\; (\mathbf{s'}, \mathbf{CT[C']}) \Downarrow \mathbf{t'}, \mathbf{a}$$
$$\& \; (\mathbf{t}, \mathbf{t'}, \mathbf{C}, \mathbf{C'}) \in (\approx)$$

- Thm (Soundness & Completeness):

$$(\mathbf{0}, \mathbf{0}, \mathbf{C}, \mathbf{C'}) \in (\approx) \quad \text{iff} \quad (\mathbf{C}, \mathbf{C'}) \in (\equiv)$$

Deriving Bisimulations for (≡)

To prove **C** and **C'** equivalent we must:

- provide a set R
- (**0**, **0**, **C**, **C'**) ∈ R
- show R ⊆ (≈)

Deriving Bisimulations for (≡)

To prove **C** and **C'** equivalent we must:

- provide a set R
- (**0**, **0**, **C**, **C'**) ∈ R
- show R ⊆ (≈)
  - find an **inductive principle**

Examples of Inductive Principles

If $S \subseteq Nat$, to show **S=Nat**, we must show:

1. $0 \in S$
2. $\forall k.\ k \in S \Rightarrow k+1 \in S$

Examples of Inductive Principles

If $S \subseteq$ Nat, to show **S=Nat**, we must show:

1. $0 \in S$
2. $\forall k.\ k \in S \Rightarrow k+1 \in S$

OR:

1. $0 \in S$
2. $\forall k.\ (\forall j.\ j<k \Rightarrow j \in S) \Rightarrow k \in S$

Examples of Inductive Principles

If $S \subseteq Nat$, to show **S=Nat**, we

1. $0 \in \mathbf{S}$
2. $\forall k.\ k \in \mathbf{S} \Rightarrow k+1 \in \mathbf{S}$

The inductive principle is **independent of S**

OR:

1. $0 \in \mathbf{S}$
2. $\forall k.\ (\forall j.\ j<k \Rightarrow j \in \mathbf{S}) \Rightarrow k \in \mathbf{S}$

Examples of Inductive Principles

If $S \subseteq Nat$, to show **S=Nat**, we

1. $0 \in S$
2. $\forall k.\ k \in S \Rightarrow k+1 \in S$

Proof Obligations of S

OR:

1. $0 \in S$
2. $\forall k.\ (\forall j.\ j<k \Rightarrow j \in S) \Rightarrow k \in S$

Deriving Bisimulations for (≡)

To prove **C** and **C'** equivalent we must:

- provide a set R
- (**0**, **0**, **C**, **C'**) ∈ R
- show R ⊆ (≈)
  - find an **inductive principle**

Deriving Bisimulations for (≡)

To prove **C** and **C'** equivalent we must:

- provide a set R
- (**0**, **0**, **C**, **C'**) ∈ R
- show R ⊆ (≈)
  - find an **inductive principle**
    - depends on the language, operational semantics, and (≈)
    - it is **independent of the set R**
    - it requires some **proof obligations on R**

Deriving Bisimulations for (≡)

To prove **C** and **C'** equivalent we must:

- provide a set R
- (**0**, **0**, **C**, **C'**) ∈ R
- R satisfies the **Proof Obligations**

Inductive Principle from the def. of (≈)

$R \subseteq (\approx)$ iff

- $\forall (s, s', C, C') \in R$:

For any class-table context **CT[ ]**:

$$(s, CT[C]) \Downarrow t, a \Rightarrow (s', CT[C']) \Downarrow t', a \;\&\; (t, t', C, C') \in R$$

Inductive Principle from the def. of (≈)

$R \subseteq (\approx)$ iff $\forall k.$

- $\forall (s, s', C, C') \in R.$

For any class-table context **CT[ ]**:

$$(s, CT[C]) \Downarrow^{<k} t, a \Rightarrow (s', CT[C']) \Downarrow t', a$$
$$\& (t, t', C, C') \in R$$

Inductive Principle from the def. of (≈)

$R \subseteq (\approx)$ iff $\forall k.$ **IH(k)**

IH(k)

- $\forall (s, s', C, C') \in R$.

For any class-table context **CT[ ]**:

$$(s, CT[C]) \Downarrow^{<k} t, a \Rightarrow (s', CT[C']) \Downarrow t', a \;\&\; (t, t', C, C') \in R$$

Inductive Principle from the def. of (≈)

$R \subseteq (\approx)$ iff $\forall k.IH(k)$

iff

$$\forall k.\ (\forall j.\ j<k \Rightarrow IH(j)) \Rightarrow IH(k)$$

Inductive Principle from the def. of (≈)

$R \subseteq (\approx)$ iff $\forall k.IH(k)$

iff

$$\forall k.\ (\forall j.\ j<k \Rightarrow IH(j)) \Rightarrow IH(k)$$

- Try the proof for **arbitrary R** (proof construction scheme)
  - find **necessary & sufficient proof obligations on R**
  - turn proof obligations into **Bisimulation Conditions**

Inductive Principle from the def. of (≈)

$R \subseteq (\approx)$ iff $\forall k.IH(k)$

iff

$\forall k.\ (\forall j.\ j<k \Rightarrow IH(j)) \Rightarrow IH(k)$

Great to have in proofs of equivalence

- Try the proof for **arbitrary R** (proof construction scheme)
  - find **necessary & sufficient proof obligations on R**
  - turn proof obligations into **Bisimulation Conditions**

Application of Our Technique

- We have derived Proof Obligations for
  - An imperative, class-based language without inheritance (J1)
    - 6 bisimulation conditions: typing&closure conditions on R
      - Context can **read/update public fields**
      - Context can **call public methods with arguments from R***
  - The extension of J1 with **inheritance** (J2)
    - The context can **override methods in subclasses**
    - The context can **access protected fields/methods**
  - The extension of J2 with **downcasting** (J3)
    - No change in proof obligations

Proving the Equivalence of Cells (J1)

```
class Cell {
  private Object g;
  Cell() { g = null; }
  public void set(Object b) {
    g = b;
  }
  public Object get() {
    return g;
  }
}
```

```
class Cell {
  private Object g1, g2;
  private int cnt;
  Cell() {
    g1 = null; g2 = null; cnt = 0;
  }
  public void set(Object b) {
    cnt = cnt + 1;
    if ((cnt % 2) == 0)
      g1 = b;
    else
      g2 = b;
  }
  public Object get() {
    if ((cnt % 2) == 0)
      return g1;
    else
      return g2;
  }
}
```

Proving the Equivalence of Cells (J1)

Let R = { (s, s', Cell, Cell') |

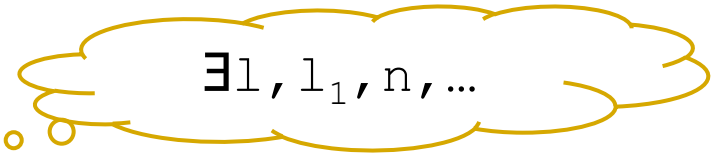
$s=[l= \text{obj Cell}\{g=\mathbf{l_1}\}] \bullet s_0$

$(\ s'=[l= \text{obj Cell}\{g_1=\mathbf{l_1}, g_2=l_2, \text{cnt}=\mathbf{2n}\}] \bullet s_0'$

OR

$s'=[l= \text{obj Cell}\{g_1=l_2, g_2=\mathbf{l_1}, \text{cnt}=\mathbf{2n+1}\}] \bullet s_0'\ )\ \}$

Proving the Equivalence of Cells (J1)



Let R = { (s, s', Cell, Cell') |

$s=[l= obj\ Cell\{g=\mathbf{l_1}\}] \bullet s_0$

($s'=[l= obj\ Cell\{g_1=\mathbf{l_1}, g_2=l_2,\ cnt=\mathbf{2n}\}] \bullet s_0'$

OR

$s'=[l= obj\ Cell\{g_1=l_2, g_2=\mathbf{l_1},\ cnt=\mathbf{2n+1}\}] \bullet s_0'$) }

Proving the Equivalence of Cells (J1)

Let R = { (s, s', Cell, Cell') |

$$
\begin{array}{ll}
 & s=[l= \text{obj Cell}\{g=\mathbf{l_1}\}] \bullet s_0 \\
(& s'=[l= \text{obj Cell}\{g_1=\mathbf{l_1}, g_2=l_2, \text{cnt}=\mathbf{2n}\}] \bullet s_0' \\
\text{OR} & \\
 & s'=[l= \text{obj Cell}\{g_1=l_2, g_2=\mathbf{l_1}, \text{cnt}=\mathbf{2n+1}\}] \bullet s_0' \ ) \ \}
\end{array}
$$

PO 6: Must show that if

$$s, \text{CT[Cell]}, l.\text{get}() \Downarrow t, a$$

then

$$s', \text{CT[Cell']}, l.\text{get}() \Downarrow t', a$$

and (t, t', Cell, Cell') $\in$ R

Proving the Equivalence of Cells (J1)

Let R = { (s, s', Cell, Cell') |

$$s=[l= \text{obj Cell}\{g=\mathbf{l_1}\}] \bullet s_0$$
$$(\ s'=[l= \text{obj Cell}\{g_1=\mathbf{l_1}, g_2=l_2, \text{cnt}=\mathbf{2n}\}] \bullet s_0'$$
$$\text{OR}$$
$$s'=[l= \text{obj Cell}\{g_1=l_2, g_2=\mathbf{l_1}, \text{cnt}=\mathbf{2n+1}\}] \bullet s_0'\ ) \}$$

PO 6: Must show that if

$$s,\ CT[Cell],\ l.get() \Downarrow \mathbf{s},\ \mathbf{l_1}$$

then

$$s',\ CT[Cell'],\ l.get() \Downarrow \mathbf{s'},\ \mathbf{l_1}$$

and (**s**, **s'** , Cell, Cell') ∈ R
which is **true**.

Proving the Equivalence of Cells (J1)

Let R = { (s, s', Cell, Cell') |

$$s=[l= \text{obj Cell}\{g=\mathbf{l_1}\}] \bullet s_0$$
$$(\ s'=[l= \text{obj Cell}\{g_1=\mathbf{l_1}, g_2=l_2, \text{cnt}=\mathbf{2n}\}] \bullet s_0'$$
$$\text{OR}$$
$$s'=[l= \text{obj Cell}\{g_1=l_2, g_2=\mathbf{l_1}, \text{cnt}=\mathbf{2n+1}\}] \bullet s_0'\ ) \}$$

PO 6: Must show that for all $\mathbf{l_3}$

$$s,\ CT[Cell],\ l.\mathbf{set}(l_3) \Downarrow t,\ a$$

then

$$s',\ CT[Cell'],\ l.\mathbf{set}(l_3) \Downarrow t',\ a$$

and (t, t', Cell, Cell') ∈ R

Proving the Equivalence of Cells (J1)

Let R = { (s, s', Cell, Cell') |

$$
\begin{array}{l}
\quad s=[l=\ obj\ Cell\{g=\mathbf{l_1}\}] \bullet s_0 \\
(\ \ s'=[l=\ obj\ Cell\{g_1=\mathbf{l_1}, g_2=l_2,\ cnt=\mathbf{2n}\}] \bullet s_0' \\
\text{OR} \\
\quad s'=[l=\ obj\ Cell\{g_1=l_2, g_2=\mathbf{l_1},\ cnt=\mathbf{2n+1}\}] \bullet s_0'\ )\ \}
\end{array}
$$

PO 6: Must show that for all l_3

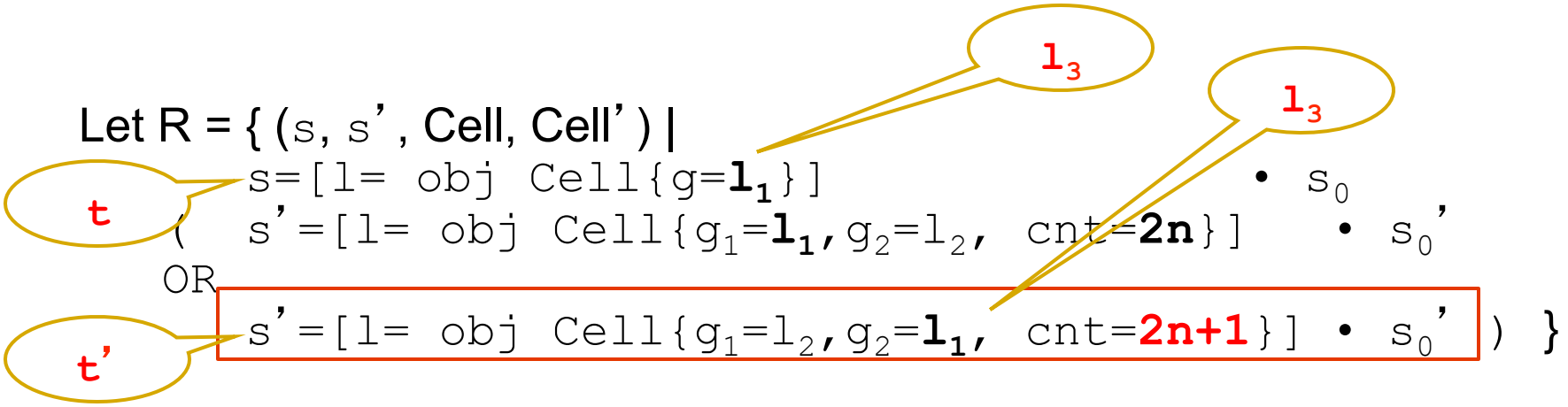
$$s,\ CT[Cell],\ l.set(l_3) \Downarrow t,\ a$$

then

$$s',\ CT[Cell'],\ l.set(l_3) \Downarrow t',\ a$$

and (t, t', Cell, Cell') ∈ R

Proving the Equivalence of Cells (J1)



PO 6: Must show that for all l_3

$$s,\ CT[Cell],\ l.set(l_3) \Downarrow t,\ a$$

then

$$s',\ CT[Cell'],\ l.set(l_3) \Downarrow t',\ a$$

and $(t, t', Cell, Cell') \in R$
which is **true**.

Proving the Equivalence of Cells (J1)

Let R = { (s, s', Cell, Cell') |

$$s=[l=\ obj\ Cell\{g=\mathbf{l_1}\}] \bullet s_0$$

$$(\ \ s'=[l=\ obj\ Cell\{g_1=\mathbf{l_1}, g_2=l_2,\ cnt=\mathbf{2n}\}] \bullet s_0'$$

OR

$$s'=[l=\ obj\ Cell\{g_1=l_2, g_2=\mathbf{l_1},\ cnt=\mathbf{2n+1}\}] \bullet s_0'\ )\ \}$$

PO 6: Must show that for all l_3

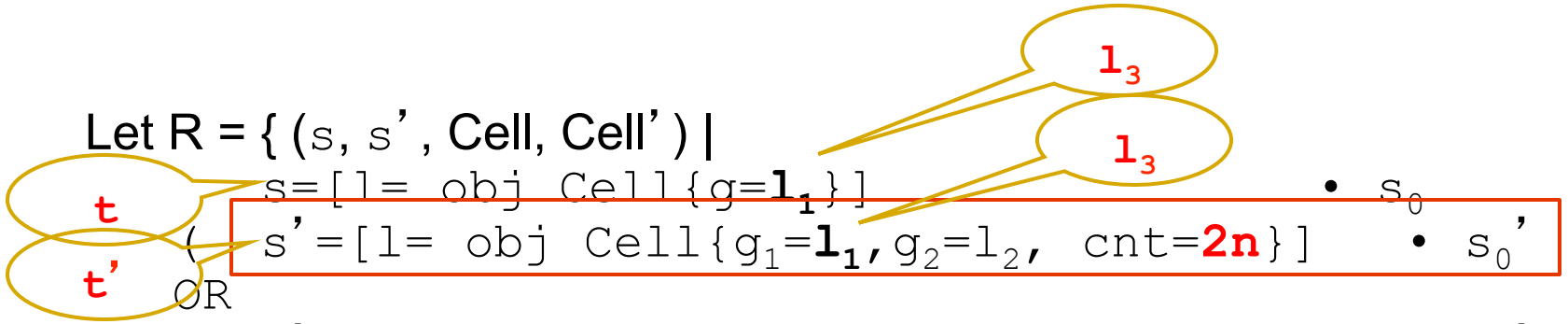
$$s,\ CT[Cell],\ l.set(l_3) \Downarrow t,\ a$$

then

$$s',\ CT[Cell'],\ l.set(l_3) \Downarrow t',\ a$$

and (t, t', Cell, Cell') ∈ R

Proving the Equivalence of Cells (J1)



PO 6: Must show that for all l_3

$$s,\ CT[Cell],\ l.set(l_3) \Downarrow t,\ a$$

then

$$s',\ CT[Cell'],\ l.set(l_3) \Downarrow t',\ a$$

and $(t, t', Cell, Cell') \in R$

which is **true**.

Proving the Equivalence of Cells (J1)

Let R = { (s, s', Cell, Cell') |

$$
\begin{array}{ll}
 & s=[l=\ \text{obj}\ \text{Cell}\{g=\mathbf{l_1}\}] \bullet s_0 \\
(& s'=[l=\ \text{obj}\ \text{Cell}\{g_1=\mathbf{l_1}, g_2=l_2,\ \text{cnt}=\mathbf{2n}\}] \bullet s_0' \\
\text{OR} & \\
 & s'=[l=\ \text{obj}\ \text{Cell}\{g_1=l_2, g_2=\mathbf{l_1},\ \text{cnt}=\mathbf{2n+1}\}] \bullet s_0' \ ) \ \}
\end{array}
$$

The rest of the PO are satisfied by construction of R

Proving the Equivalence of Cells (J2, J3)

Let R = { (s, s', Cell, Cell') |

$$
\begin{aligned}
& s=[l= \text{obj } \mathbf{C}\{g=\mathbf{l_1}, \ldots\}] \bullet s_0 \\
(\; & s'=[l= \text{obj } \mathbf{C}\{g_1=\mathbf{l_1}, g_2=l_2, \text{cnt}=\mathbf{2n}, \ldots\}] \bullet s_0' \\
\text{OR} \\
& s'=[l= \text{obj } \mathbf{C}\{g_1=l_2, g_2=\mathbf{l_1}, \text{cnt}=\mathbf{2n+1}, \ldots\}] \bullet s_0' \;)\}
\end{aligned}
$$

Proving the Equivalence of Cells (J2, J3)

Let R = { (s, s', Cell, Cell') |

$$s = [l = \text{obj } \mathbf{C}\{g = \mathbf{l_1}, \ldots\}] \bullet s_0$$

$$(\; s' = [l = \text{obj } \mathbf{C}\{g_1 = \mathbf{l_1}, g_2 = l_2, \text{cnt} = \mathbf{2n}, \ldots\}] \bullet s_0'$$

$$\text{OR}$$

$$s' = [l = \text{obj } \mathbf{C}\{g_1 = l_2, g_2 = \mathbf{l_1}, \text{cnt} = \mathbf{2n+1}, \ldots\}] \bullet s_0' \;)\}$$

C <: Cell

Proving the Equivalence of Cells (J2, J3)

Let R = { (s, s', Cell, Cell') |

$s = [l = \text{obj } \mathbf{C}\{g = \mathbf{l_1}, \ldots\}]$ • s_0

($s' = [l = \text{obj } \mathbf{C}\{g_1 = \mathbf{l_1}, g_2 = l_2, \text{cnt} = \mathbf{2n}, \ldots\}]$ • s_0'

OR

$s' = [l = \text{obj } \mathbf{C}\{g_1 = l_2, g_2 = \mathbf{l_1}, \text{cnt} = \mathbf{2n+1}, \ldots\}]$ • s_0')}

Proving the Equivalence of Cells (J2, J3)

Let R = { (s, s', Cell, Cell') |

$$s=[l= \text{obj } \mathbf{C}\{g=\mathbf{l_1}, \ldots\}] \bullet s_0$$

$$(\ s'=[l= \text{obj } \mathbf{C}\{g_1=\mathbf{l_1}, g_2=l_2, \text{cnt}=\mathbf{2n}, \ldots\}] \bullet s_0'$$

OR

$$s'=[l= \text{obj } \mathbf{C}\{g_1=l_2, g_2=\mathbf{l_1}, \text{cnt}=\mathbf{2n+1}, \ldots\}] \bullet s_0' \ )\}$$

The rest of the proof is the same

Application of Our Technique

- We have derived Proof Obligations for
  - An imperative, class-based language without inheritance (J1)
    - 6 bisimulation conditions: typing&closure conditions on R
      - Context can **read/update public fields**
      - Context can **call public methods with arguments from R***
  - The extension of J1 with **inheritance** (J2)
    - The context can **override methods in subclasses**
    - The context can **access protected fields/methods**
  - The extension of J2 with **downcasting** (J3)
    - No change in proof obligations

Application of Our Technique

- We have derived Proof Obligations for
  - An imperative, class-based language without inheritance (J1)
    - 6 bisimulation conditions: typing&clos
      - Context can **read/update public fie**
      - Context can **call public methods**
  - The extension of J1 with **inher**
    - The context can **override metho**
    - The context can **access protecte**
  - The extension of J2 with **do ncasting** (J3)
    - No change in proof obligations

Are there no extra interactions b/w term-context? **(conservative extension?)**

Downcasting breaks equivalences [A. Kennedy]

```
class RO {
 protected int val;
 public RO(int val) { this.val = val; }
 public int get() { return this.val; }
}
class RW extends RO {
 public RW(int val) { super(val); }
 public void set(int val) { this.val = val; }
}
class Callback { public void act(RO arg) { } }
```

```
class C {
 public int foo(Callback f) {
   RW rw = new RW(5);
   f.act(rw);
   return rw.get();
 }
}
```

≠

```
class C {
 public int foo(Callback f) {
   RW rw = new RW(5);
   f.act(rw);
   return 5;
 }
}
```

Downcasting breaks equivalences [A. Kennedy]

```
class RO {
 protected int val;
 public RO(int val) { this.val = val; }
 public int get() { return this.val; }
}
class RW extends RO {
 public RW(int val) { super(val); }
 public void set(int val) { this.val = val; }
}
class Callback { public void act(RO arg) { } }
```

Proof technique is correct:
we need to consider all callbacks (including those that downcast);
val can have any value after call to f

```
class C {
 public int foo(Callback f) {
   RW rw = new RW(5);
   f.act(rw);
   return rw.get();
 }
}
```

≠

```
class C {
 public int foo(Callback f) {
   RW rw = new RW(5);
   f.act(rw);
   return 5;
 }
}
```

Downcasting breaks equivalences [A. Kennedy]

```
class RO {
 protected int val;
 public RO(int val) { this.val = val; }
 public int get() { return this.val; }
}
class RW extends RO {
 public RW(int val) { super(val); }
 public void set(int val) { this.val = val; }
}
class Callback { public void act(RO arg) { } }
```

Proof technique is correct:
we need to consider all callbacks (including

We adapted the relations to show intuitive change in POs when adding downcasting.

```
class C {
 public int foo(Callback f) {
   RW rw = new RW(5);
   f.act(rw);
   return rw.get();
 }
}
```

≠

```
class C {
 public int foo(Callback f) {
   RW rw = new RW(5);
   f.act(rw);
   return 5;
 }
}
```

Conclusions

- Bisimulation proof technique of equivalence
  - derived by a “brute-force” but well-structured proof of equivalence
- It works really well in proofs
  - without a detailed model of interactions
- Can help us understand the semantics of the language
  - by looking at example equivalences
  - not by looking at definitions

Thanks!