Reasoning About Class Behavior

Vasileios Koutavas

Northeastern University

Roadmap

- A Motivating Example
- Program Equivalence and Contextual Equivalence
- Technology for Proving Equivalences
- (Bisimulations)
- Deriving Bisimulations for Equivalence
- An Actual Proof of Equivalence

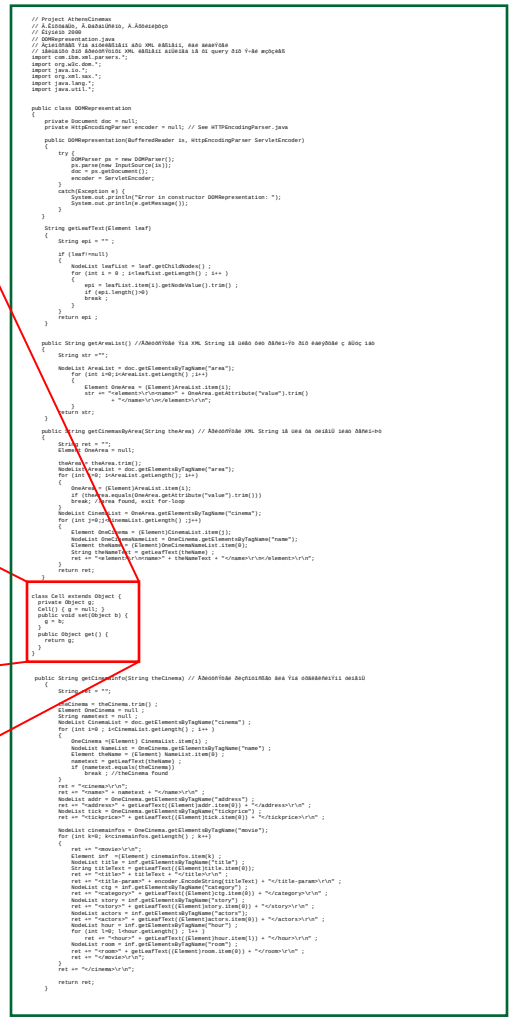
A Cell Class

```
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;
  }
}
```

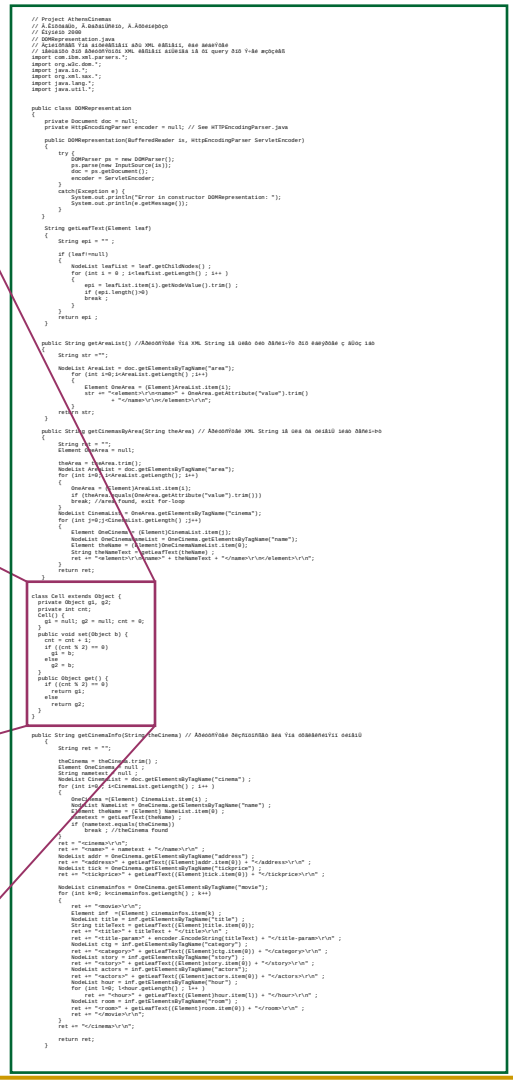
A Cell Class

```
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;
  }
}
```

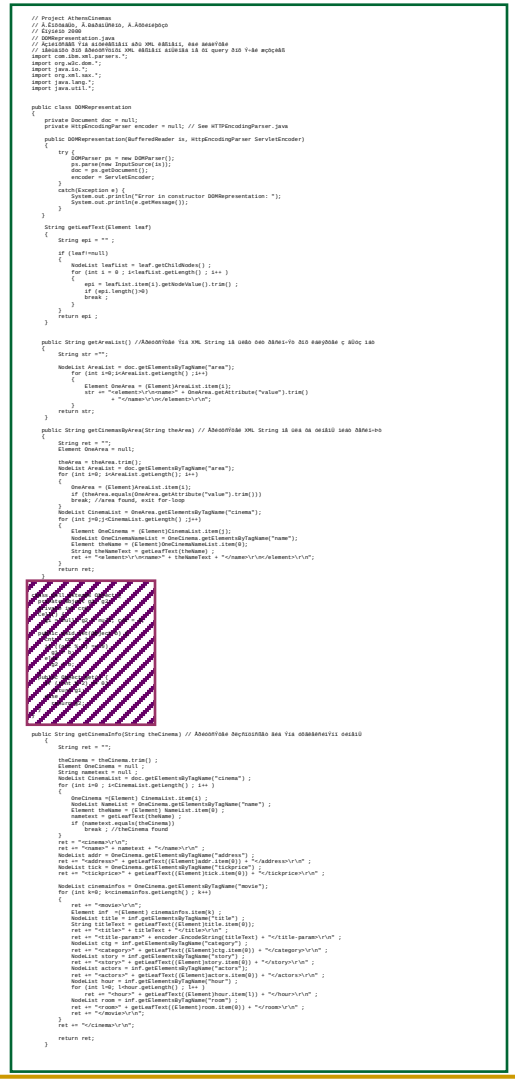
The Same Program



A Cell Class

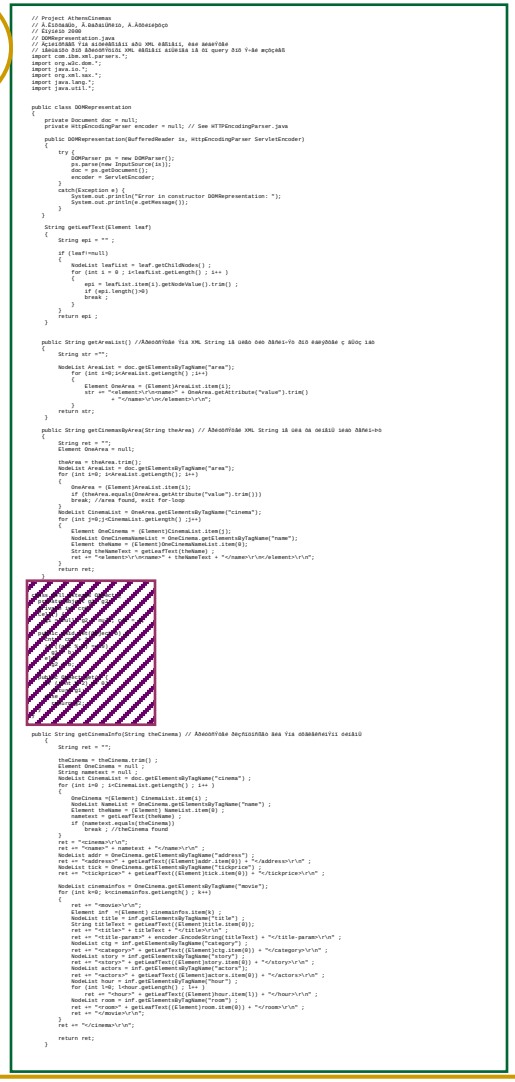
The Same Program

A Cell Class



A Cell Class

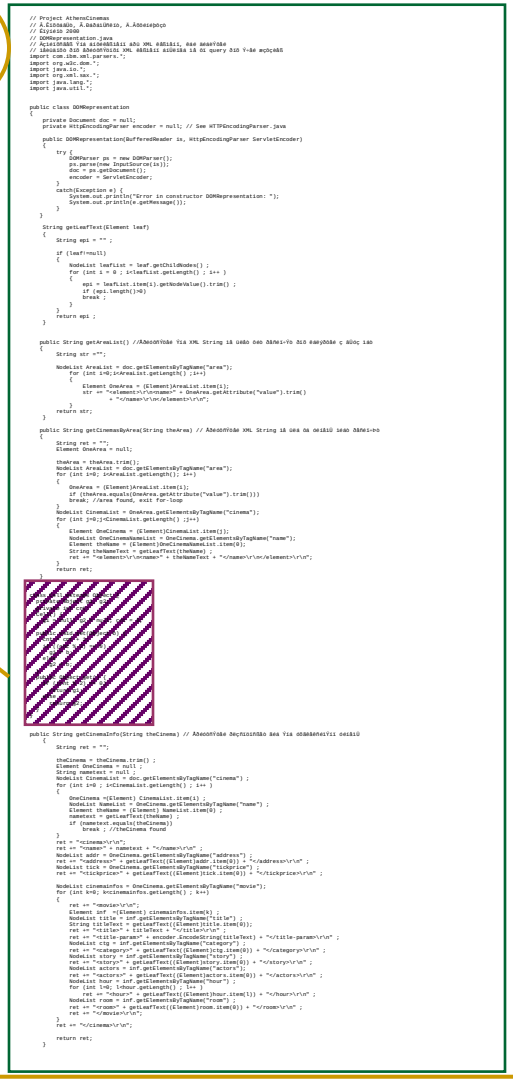
Do **these two** programs have the same behavior?



A Cell Class

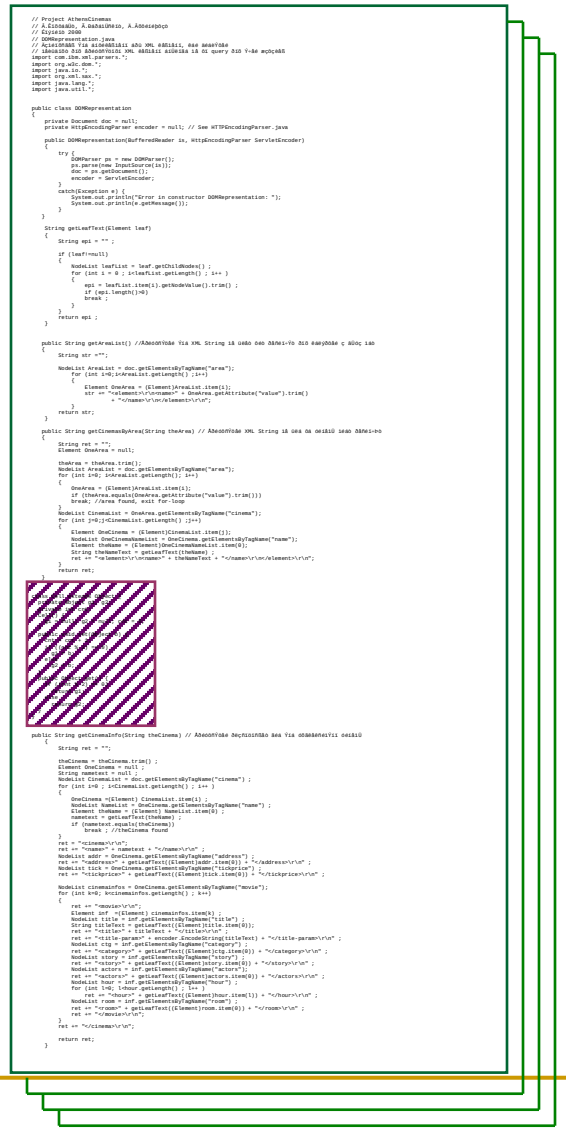
Do **these two** programs have the same behavior?

Program Equivalence



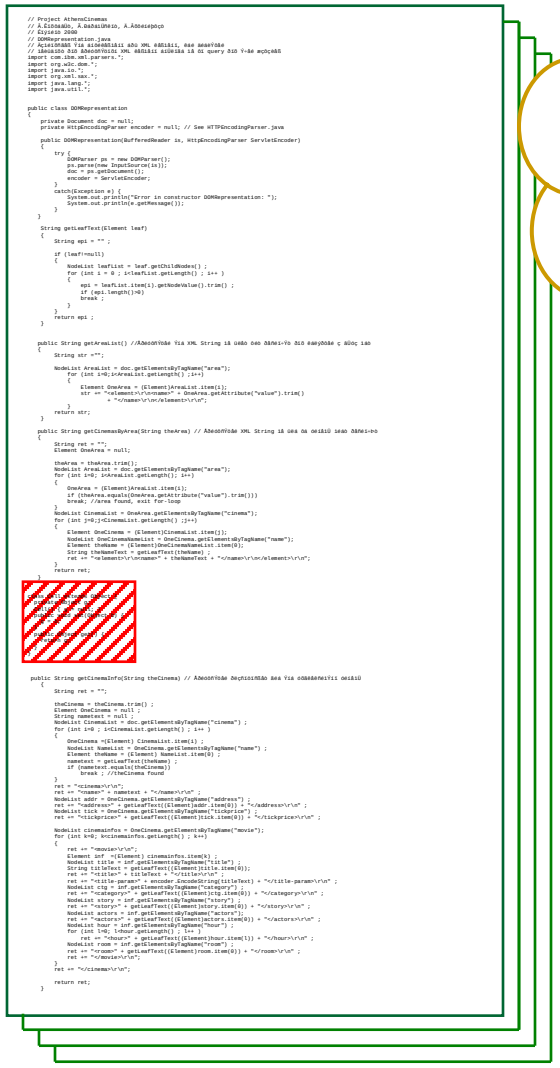
A Cell Class

...



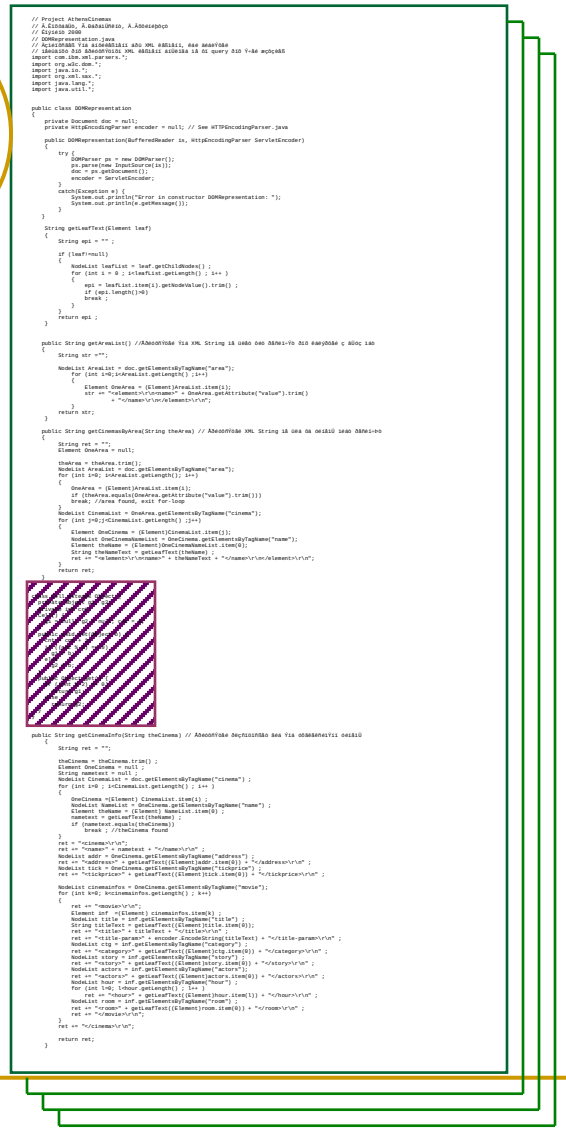
...

A Cell Class



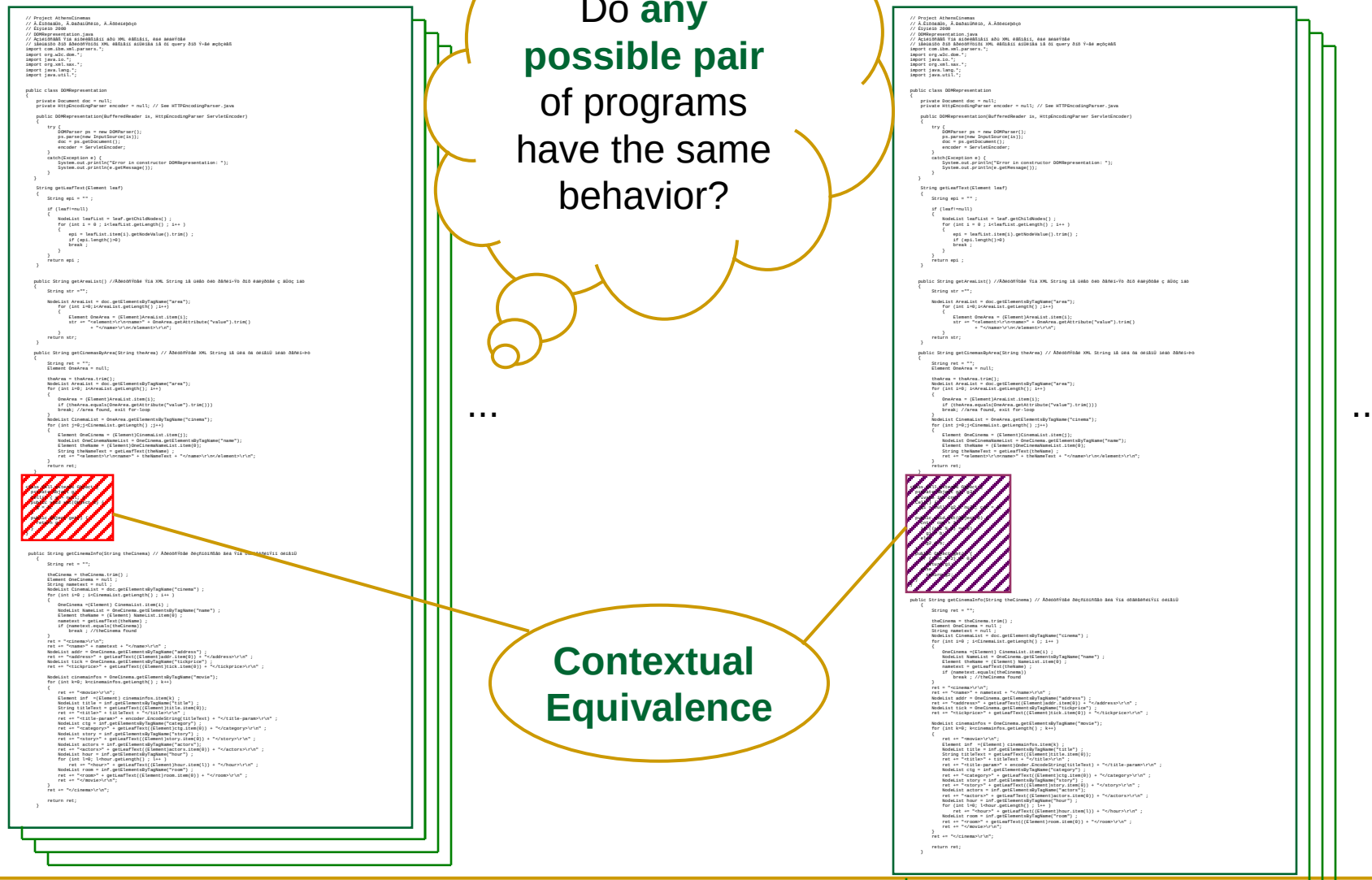
Do **any possible pair** of programs have the same behavior?

…



…

A Cell Class


Do **any possible pair** of programs have the same behavior?
...
...
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[ ]**:

CT[C] $\Downarrow$ iff **CT[C']** $\Downarrow$

Contextual Equivalence

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

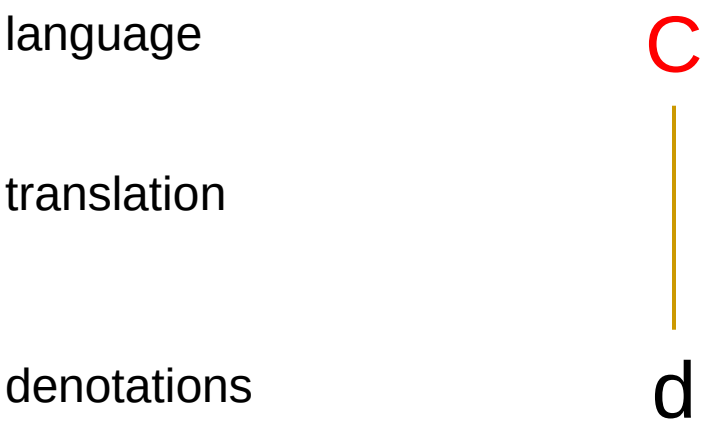
For any class-table context **CT[ ]**:

CT[C] $\Downarrow$ iff **CT[C']** $\Downarrow$

  - The "golden standard of equivalences"
  - Can't be used directly in the proof of a non-trivial equivalence
  - Can't facilitate an inductive proof

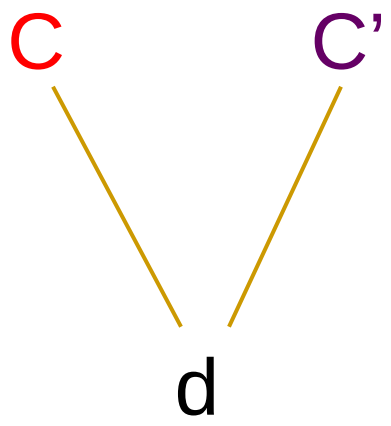
Technology for Proving Cxt Equivalence

- Denotational Semantics



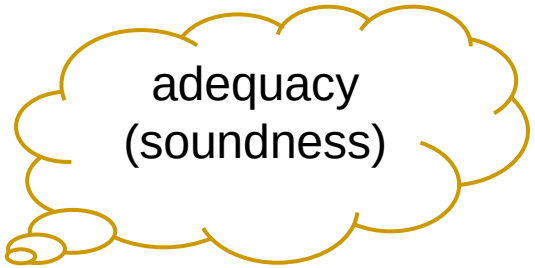
Technology for Proving Cxt Equivalence

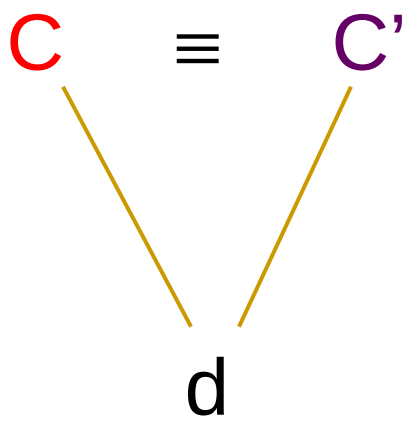
- Denotational Semantics



Technology for Proving Cxt Equivalence

- Denotational Semantics





Technology for Proving Cxt Equivalence

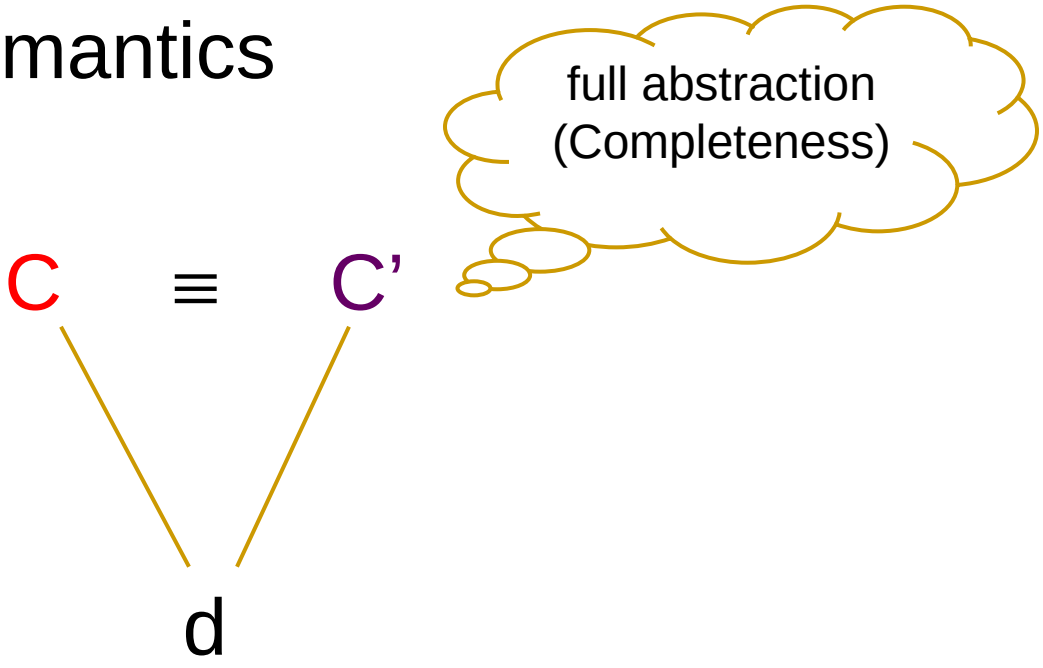
- Denotational Semantics

$$C \equiv C'$$

d

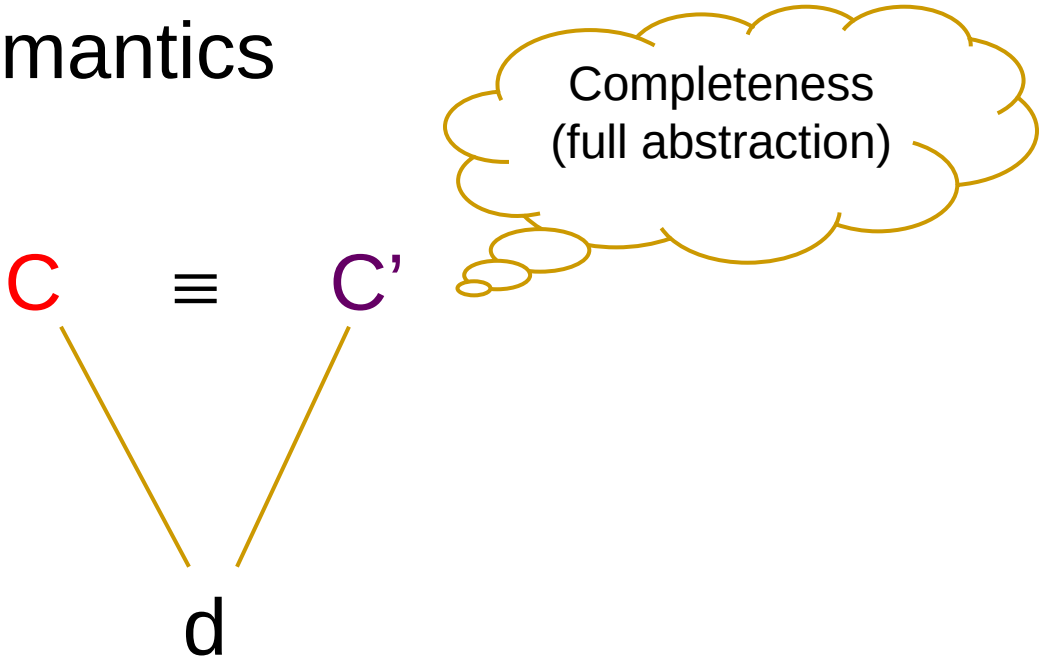
Technology for Proving Cxt Equivalence

- Denotational Semantics



Technology for Proving Cxt Equivalence

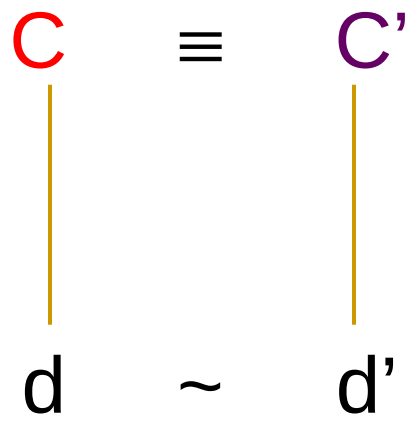
- Denotational Semantics



  - Usually not fully abstract when modelling languages with store

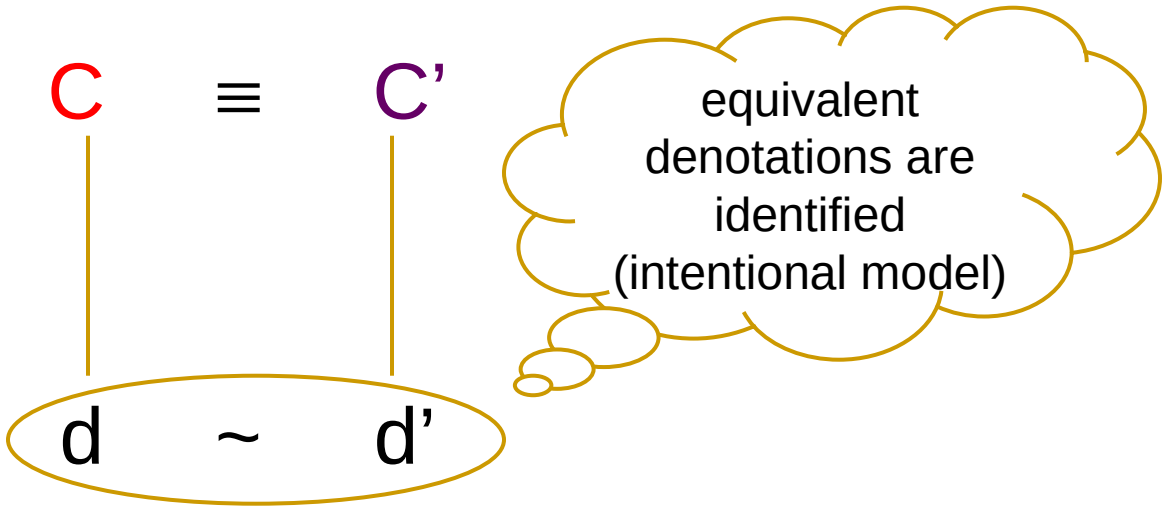
Technology for Proving Cxt Equivalence

- Denotational Semantics + Some Relation



Technology for Proving Cxt Equivalence

- Denotational Semantics + Some Relation



Technology for Proving Cxt Equivalence

- Denotational Semantics + Some Relation
  - Usual Denotational Semantics + Logical Relations
  - Usual Denotational Semantics + Bisimulations [Banerjee&Naumann]
  - Game Semantics + “quotient” relations [Reynolds, Hyland&Ong]

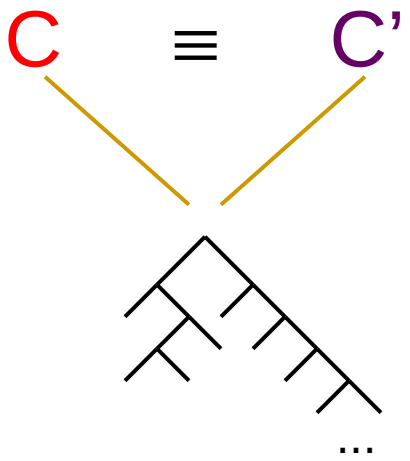
Several achieve full abstraction, but hard to use to prove some equivalences

Technology for Proving Cxt Equivalence

- Trace Equivalence [Jeffrey&Rathke]
  - Define an appropriate trace semantics (not the operational semantics of the language)

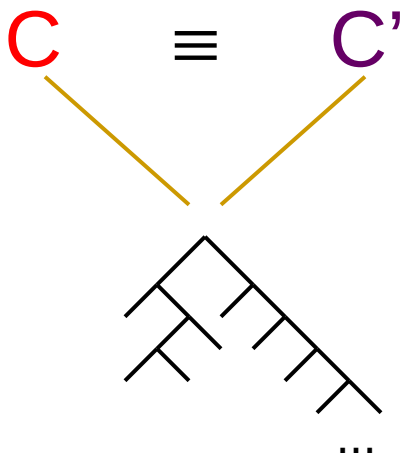
Technology for Proving Cxt Equivalence

- Trace Equivalence [Jeffrey&Rathke]
  - Define an appropriate trace semantics



Technology for Proving Cxt Equivalence

- Trace Equivalence [Jeffrey&Rathke]
  - Define an appropriate trace semantics



Fully abstract for Java Jr. & not so hard to use. Hard to find the appropriate trace semantics. Not obvious how to handle inheritance.

Technology for Proving Cxt Equivalence

- Operational Semantics + Logical Relations [Pitts, Pitts&Stark]
  - Fully abstract for several languages, but hard to use to prove some equivalences
- Operational Semantics + Bisimulations [Abramsky, Sumii&Pierce, Koutavas&Wand]
  - Fully abstract for diverse languages (functional, imperative)
  - Have proven all known hard equivalences
  - Not ready yet to attack complex real-world equivalences

Proving Contextual Equivalence

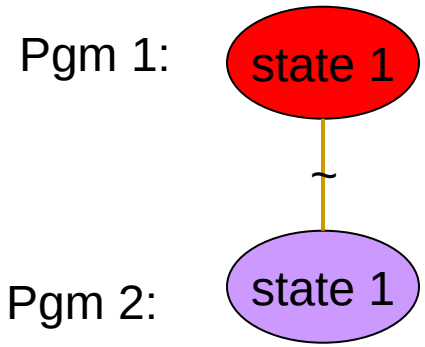
- Cxt Equivalence is the **largest set** of equivalent classes

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

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

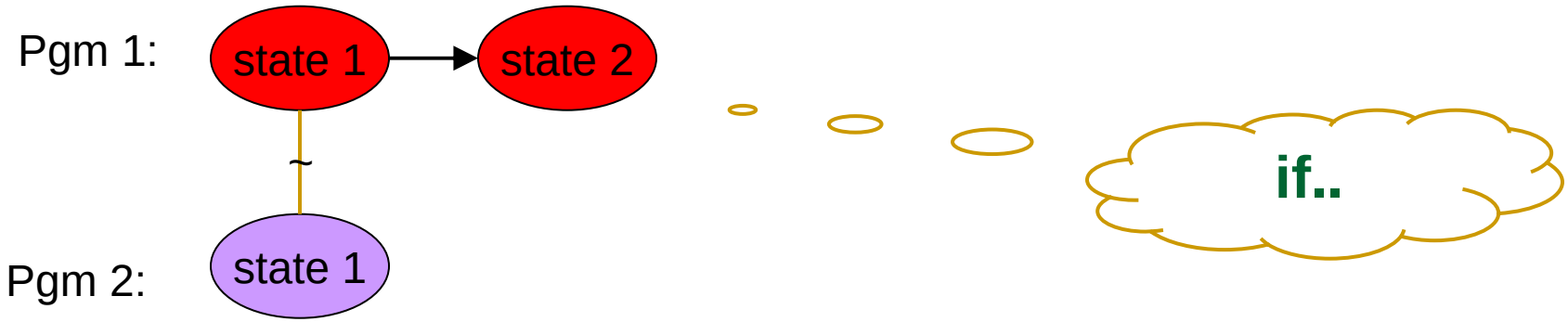
Bisimulations

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



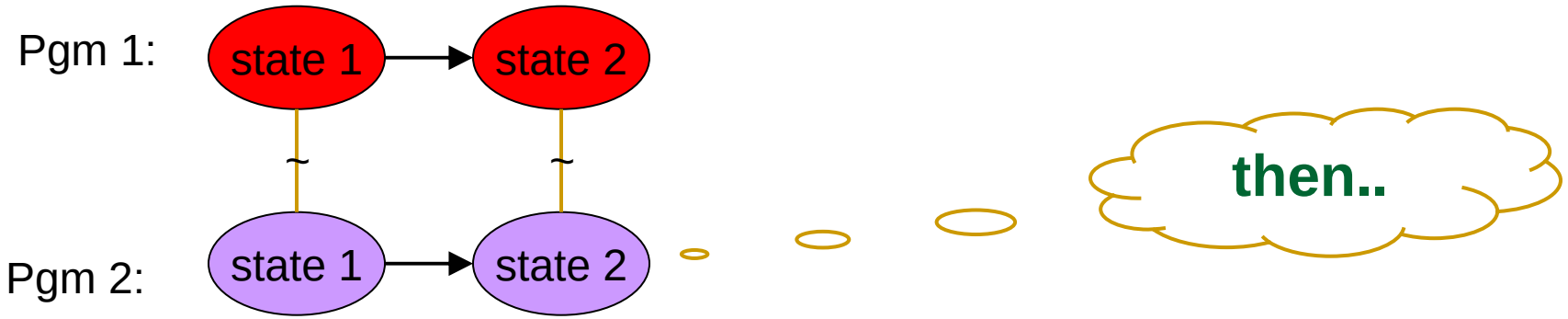
Bisimulations

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



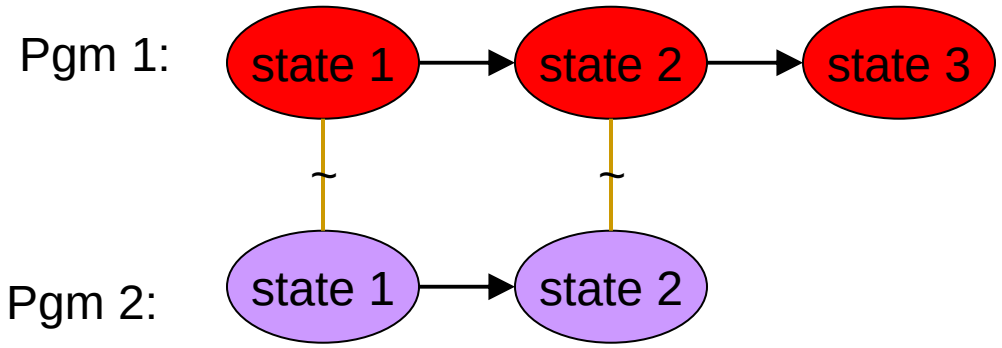
Bisimulations

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



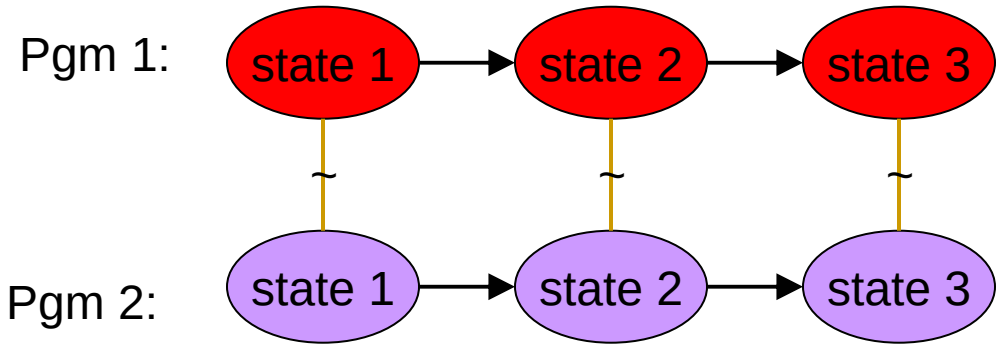
Bisimulations

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



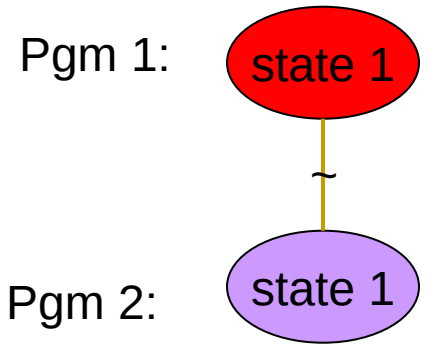
Bisimulations

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



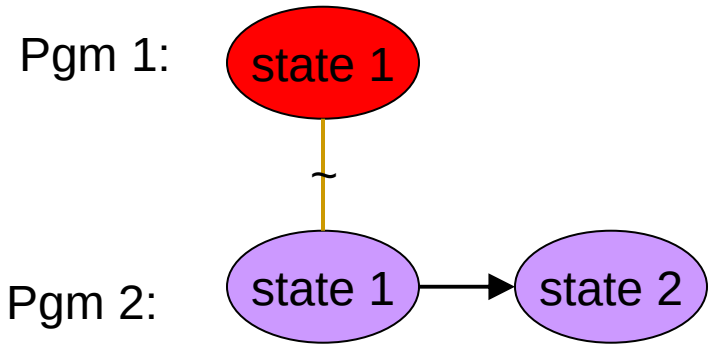
Bisimulations

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



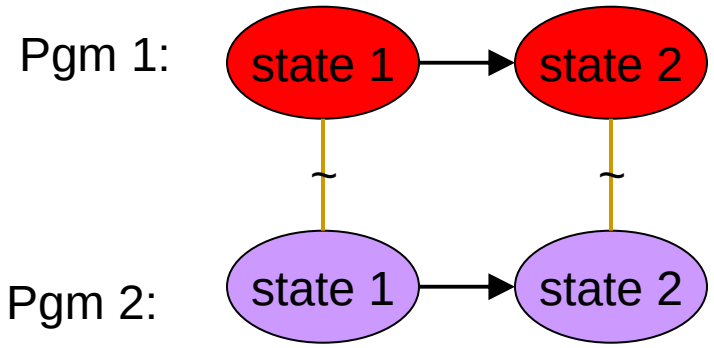
Bisimulations

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



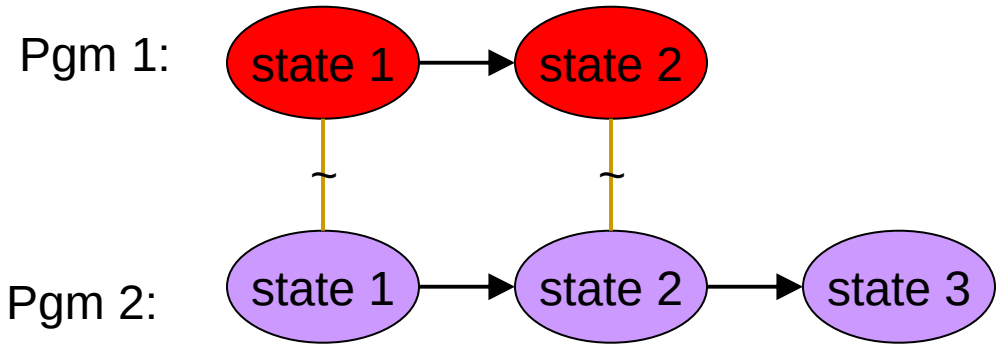
Bisimulations

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



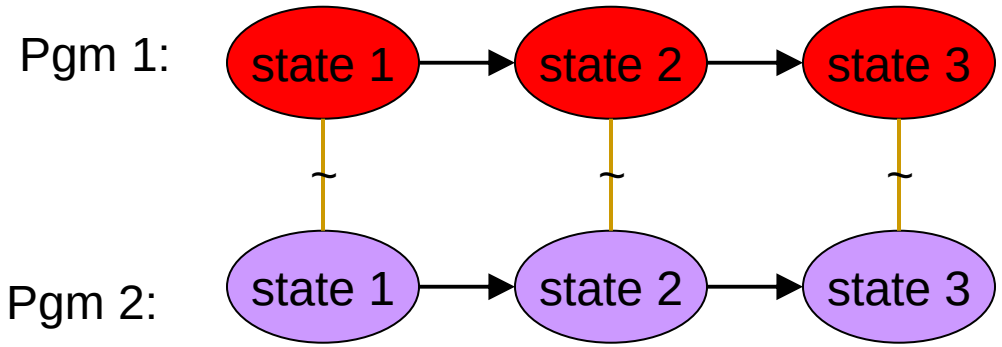
Bisimulations

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



Bisimulations

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



Deriving Bisimulations for Cxt Equivalence

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

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

CT[C] $\Downarrow$ iff **CT[C']** $\Downarrow$

Deriving Bisimulations for Cxt Equivalence

- (**C**, **C'**) ∈ (≡) iff:

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

0, CT[C] ⇓ iff **0, CT[C']** ⇓

Deriving Bisimulations for Cxt Equivalence

- (**C**, **C'**) ∈ (≡) iff:

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

0, CT[C] ⇓ iff **0, CT[C']** ⇓

Define Adequacy:

- (**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 Cxt Equivalence

- (**C**, **C'**) ∈ (≡) iff:

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

0, CT[C] ⇓ iff **0, CT[C']** ⇓

Define Adequacy:

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 Cxt Equivalence

- (**C**, **C'**) ∈ (≡) iff:

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

0, CT[C] ⇓ iff **0, CT[C']** ⇓

Define Adequacy:

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

This is a **big-step bisimulation**

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

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

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

Deriving Bisimulations for Cxt Equivalence

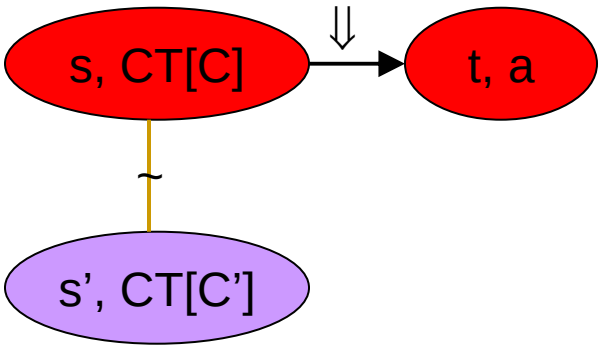
Define Adequacy:

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

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

$$\mathbf{s, CT[C]} \Downarrow \mathbf{t, a} \;\Rightarrow\; \mathbf{s', CT[C']} \Downarrow \mathbf{t', a}$$

$$\mathbf{\&\ (t, t', C, C')} \in (\approx)$$



Deriving Bisimulations for Cxt Equivalence

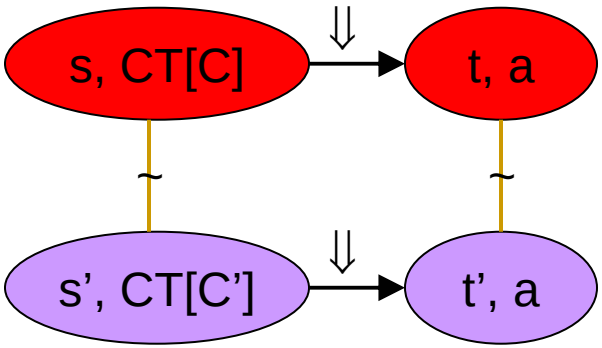
Define Adequacy:

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

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

$$\mathbf{s, CT[C]} \Downarrow \mathbf{t, a} \;\Rightarrow\; \mathbf{s', CT[C']} \Downarrow \mathbf{t', a}$$

$$\mathbf{\&\ (t, t', C, C')} \in (\approx)$$



Deriving Bisimulations for Cxt Equivalence

- Thm (Soundness & Completeness):

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

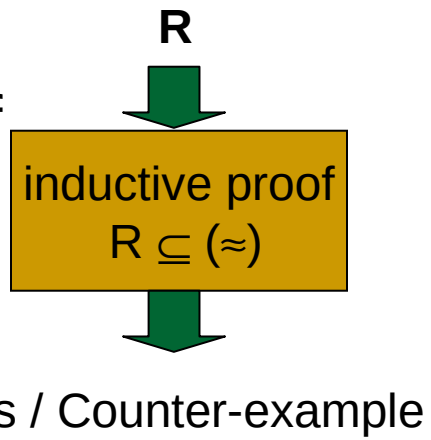
- To prove C and C' equivalent we must:
  - provide a set R
  - $(\mathbf{0}, \mathbf{0}, \mathbf{C}, \mathbf{C'}) \in R$
  - $R \subseteq (\approx)$, by a **standard** inductive proof

Deriving Bisimulations for Cxt Equivalence

- Thm (Soundness & Completeness):

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

- To prove C and C' equivalent we must:
  - provide a set R
  - $(\mathbf{0}, \mathbf{0}, \mathbf{C}, \mathbf{C'}) \in R$
  - $R \subseteq (\approx)$, by a **standard** inductive proof

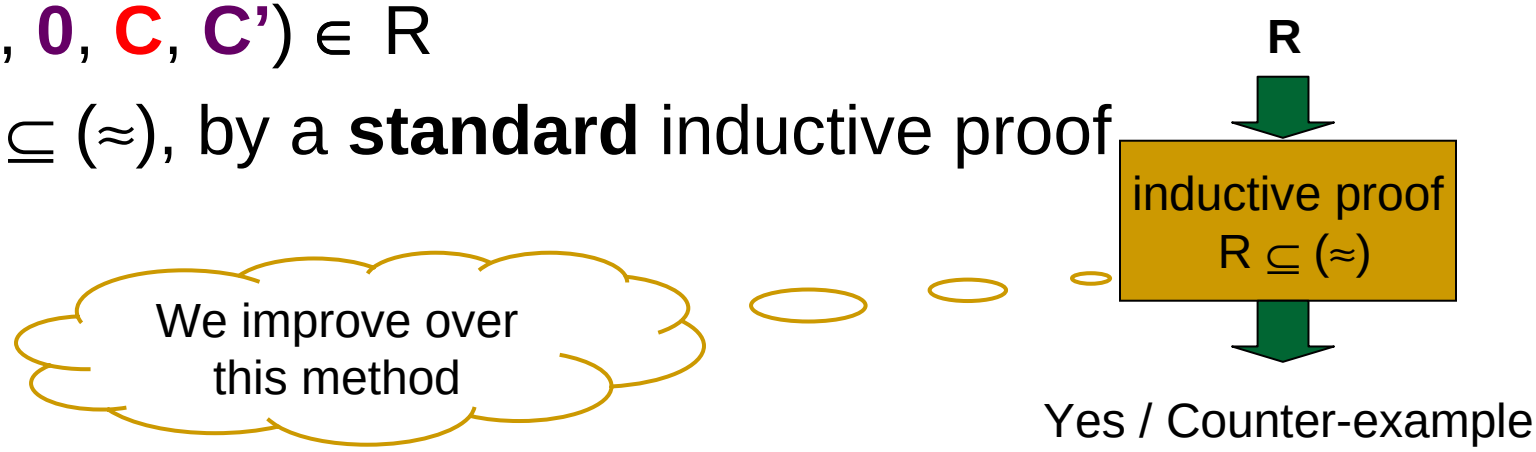


Deriving Bisimulations for Cxt Equivalence

- Thm (Soundness & Completeness):

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

- To prove C and C' equivalent we must:
  - provide a set R
  - $(\mathbf{0}, \mathbf{0}, \mathbf{C}, \mathbf{C'}) \in R$
  - $R \subseteq (\approx)$, by a **standard** inductive proof

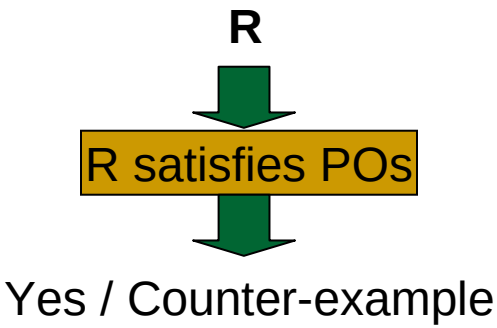


Deriving Bisimulations for Cxt Equivalence

- Inductive proof of $R \subseteq (\approx)$
  - In other words inductive proof of R being **adequate**, or R being a **bisimulation**
- For **any given R** the proof is **mostly** carried out by the induction hypothesis
  - We find **sufficient conditions** on R that can be used to **complete** the proof (**proof obligations** of R)

Deriving Bisimulations for Cxt Equivalence

- The proof of C, C' being equivalent becomes:



Proving the Equivalence of Cells

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

```
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;
  }
}
```

Proving the Equivalence of Cells

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

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

PO 6: Must show that if

$$s, CT, l.get() \Downarrow t, a$$

then

$$s', CT', l.get() \Downarrow t', a$$

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

Proving the Equivalence of Cells

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

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

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

PO 6: Must show that if

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

then

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

and (**s, s',** Cell, Cell') $\in$ R

which is **true**.

Proving the Equivalence of Cells

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

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

Must show that for all

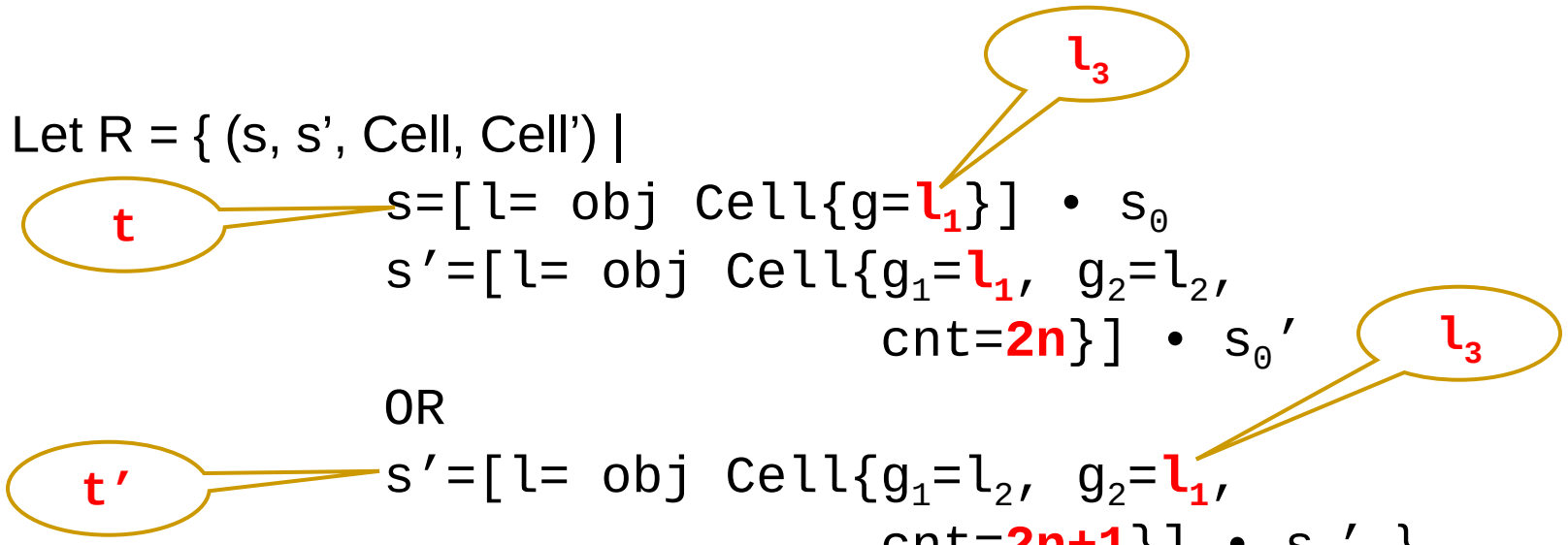
$$s,\ CT,\ l.\text{set}(l_3) \Downarrow t,\ a$$

then

$$s',\ CT',\ l.\text{get}(l_3) \Downarrow t',\ a$$

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

Proving the Equivalence of Cells



Must show that for all

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

then

$$s',\ CT',\ l.get(l_3) \Downarrow t',\ a$$

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

Τέλος