

---

# 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

## Some Program

```

// Project: KtCellClass
// Author(s): A. Koutavas, A. Koutavas
// Date: 1/2/2020
// Description: This program is a Java application that demonstrates the use of a class named Cell.
// The program is a console application that prompts the user to enter the number of cells and the length of the DNA sequence.
// It then calculates the number of possible DNA sequences and prints the result.

import java.util.Scanner;

public class KtCellClass {
    private Scanner scanner = null;

    public KtCellClass() {
        scanner = new Scanner(System.in);
    }

    public void run() {
        System.out.println("Please enter the number of cells: ");
        int numCells = scanner.nextInt();

        System.out.println("Please enter the length of the DNA sequence: ");
        int dnaLength = scanner.nextInt();

        int numSequences = calculateNumSequences(numCells, dnaLength);

        System.out.println("The number of possible DNA sequences is: " + numSequences);
    }

    private int calculateNumSequences(int numCells, int dnaLength) {
        long result = 1;
        for (int i = 0; i < numCells; i++) {
            result *= (int) Math.pow(4, dnaLength);
        }
        return (int) result;
    }

    public static void main(String[] args) {
        KtCellClass ktCellClass = new KtCellClass();
        ktCellClass.run();
    }
}

```



# 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

```
// Project: ArrayListDemo
// Author: A. NAKALAKIS, A. KALAZOGHOU
// Version: 1.0
// Description: This program demonstrates the use of the ArrayList class.
// It shows how to create an ArrayList, add elements, and retrieve them.
// It also shows how to use the Iterator class to traverse the ArrayList.
// It is a simple program that demonstrates the basic usage of the
// ArrayList class.
// It is a simple program that demonstrates the basic usage of the
// ArrayList class.

import java.util.*;

public class ArrayListDemo {
    private ArrayList<String> arr = null; // An ArrayList of Strings.
    private int size = 0; // The number of elements in the ArrayList.

    public ArrayListDemo() {
        arr = new ArrayList<>();
    }

    public void add(String s) {
        arr.add(s);
        size++;
    }

    public void remove(String s) {
        arr.remove(s);
        size--;
    }

    public String toString() {
        return arr.toString();
    }

    public void print() {
        for (String s : arr)
            System.out.println(s);
    }

    public static void main(String[] args) {
        ArrayListDemo arr = new ArrayListDemo();
        arr.add("A");
        arr.add("B");
        arr.add("C");
        arr.add("D");
        arr.add("E");
        arr.add("F");
        arr.add("G");
        arr.add("H");
        arr.add("I");
        arr.add("J");
        arr.add("K");
        arr.add("L");
        arr.add("M");
        arr.add("N");
        arr.add("O");
        arr.add("P");
        arr.add("Q");
        arr.add("R");
        arr.add("S");
        arr.add("T");
        arr.add("U");
        arr.add("V");
        arr.add("W");
        arr.add("X");
        arr.add("Y");
        arr.add("Z");
        arr.print();
    }
}
```

# A Cell Class

## The Same Program

```
// Project: KerasCell.java
// @author: A. Keras, A. Keras, A. Keras
// @date: 2017.04.01
// @version: 1.0
// @description: This class implements a cell class. It is a class that represents a cell in a neural network.
// @license: MIT License
// @url: https://github.com/keras-team/keras/blob/master/keras/layers/core_layers.py

import java.util.*;

public class KerasCell {
    private double[] data = null;
    private double[] weights = null;
    private double[] bias = null;

    public KerasCell(int[] data, double[] weights, double[] bias) {
        this.data = data;
        this.weights = weights;
        this.bias = bias;
    }

    public double[] getWeights() {
        return weights;
    }

    public double[] getBias() {
        return bias;
    }

    public double[] getData() {
        return data;
    }

    public double[] getOutput() {
        double sum = 0;
        for (int i = 0; i < data.length; i++) {
            sum += data[i] * weights[i];
        }
        return new double[] { sum };
    }

    public double[] getOutput(int[] data) {
        double sum = 0;
        for (int i = 0; i < data.length; i++) {
            sum += data[i] * weights[i];
        }
        return new double[] { sum };
    }

    public double[] getOutput(int[] data, double[] bias) {
        double sum = 0;
        for (int i = 0; i < data.length; i++) {
            sum += data[i] * weights[i];
        }
        return new double[] { sum + bias };
    }

    public double[] getOutput(int[] data, double[] bias, double[] weights) {
        double sum = 0;
        for (int i = 0; i < data.length; i++) {
            sum += data[i] * weights[i];
        }
        return new double[] { sum + bias };
    }

    public double[] getOutput(int[] data, double[] bias, double[] weights, double[] biasWeights) {
        double sum = 0;
        for (int i = 0; i < data.length; i++) {
            sum += data[i] * weights[i];
        }
        return new double[] { sum + bias + biasWeights };
    }

    public double[] getOutput(int[] data, double[] bias, double[] weights, double[] biasWeights, double[] biasWeightsWeights) {
        double sum = 0;
        for (int i = 0; i < data.length; i++) {
            sum += data[i] * weights[i];
        }
        return new double[] { sum + bias + biasWeights + biasWeightsWeights };
    }

    public double[] getOutput(int[] data, double[] bias, double[] weights, double[] biasWeights, double[] biasWeightsWeights, double[] biasWeightsWeightsWeights) {
        double sum = 0;
        for (int i = 0; i < data.length; i++) {
            sum += data[i] * weights[i];
        }
        return new double[] { sum + bias + biasWeights + biasWeightsWeights + biasWeightsWeightsWeights };
    }

    public double[] getOutput(int[] data, double[] bias, double[] weights, double[] biasWeights, double[] biasWeightsWeights, double[] biasWeightsWeightsWeights, double[] biasWeightsWeightsWeightsWeights) {
        double sum = 0;
        for (int i = 0; i < data.length; i++) {
            sum += data[i] * weights[i];
        }
        return new double[] { sum + bias + biasWeights + biasWeightsWeights + biasWeightsWeightsWeights + biasWeightsWeightsWeightsWeights };
    }
}
}
```





# A Cell Class

Do these two programs have the same behavior?

```
// Project: KinesisSimulator
// Author: A. Katsiforos, A. Katsiforos
// Date: 2019
// Description: A KinesisSimulator class that simulates a cell's behavior.
// It has methods for getting the cell's position, velocity, and acceleration.
// It also has methods for getting the cell's position, velocity, and acceleration at a given time.
// The cell's position is represented by a 2D vector.
// The cell's velocity is represented by a 2D vector.
// The cell's acceleration is represented by a 2D vector.
// The cell's position, velocity, and acceleration are represented by double arrays.
// The cell's position, velocity, and acceleration are represented by double arrays.

public class KinesisSimulator {
    private double dx = null;
    private double dy = null;
    private double vx = null;
    private double vy = null;
    private double ax = null;
    private double ay = null;

    public KinesisSimulator() {
        // Initialize the cell's position, velocity, and acceleration.
        dx = 0.0;
        dy = 0.0;
        vx = 0.0;
        vy = 0.0;
        ax = 0.0;
        ay = 0.0;
    }

    // Get the cell's position.
    public double[] getPosition() {
        return new double[] { dx, dy };
    }

    // Get the cell's velocity.
    public double[] getVelocity() {
        return new double[] { vx, vy };
    }

    // Get the cell's acceleration.
    public double[] getAcceleration() {
        return new double[] { ax, ay };
    }

    // Set the cell's position.
    public void setPosition(double dx, double dy) {
        this.dx = dx;
        this.dy = dy;
    }

    // Set the cell's velocity.
    public void setVelocity(double vx, double vy) {
        this.vx = vx;
        this.vy = vy;
    }

    // Set the cell's acceleration.
    public void setAcceleration(double ax, double ay) {
        this.ax = ax;
        this.ay = ay;
    }

    // Simulate the cell's behavior.
    public void simulate() {
        // Simulate the cell's behavior.
        // This method is a placeholder for the actual simulation logic.
    }
}
```



```
// Project: KinesisSimulator
// Author: A. Katsiforos, A. Katsiforos
// Date: 2019
// Description: A KinesisSimulator class that simulates a cell's behavior.
// It has methods for getting the cell's position, velocity, and acceleration.
// It also has methods for getting the cell's position, velocity, and acceleration at a given time.
// The cell's position is represented by a 2D vector.
// The cell's velocity is represented by a 2D vector.
// The cell's acceleration is represented by a 2D vector.
// The cell's position, velocity, and acceleration are represented by double arrays.
// The cell's position, velocity, and acceleration are represented by double arrays.

public class KinesisSimulator {
    private double dx = null;
    private double dy = null;
    private double vx = null;
    private double vy = null;
    private double ax = null;
    private double ay = null;

    public KinesisSimulator() {
        // Initialize the cell's position, velocity, and acceleration.
        dx = 0.0;
        dy = 0.0;
        vx = 0.0;
        vy = 0.0;
        ax = 0.0;
        ay = 0.0;
    }

    // Get the cell's position.
    public double[] getPosition() {
        return new double[] { dx, dy };
    }

    // Get the cell's velocity.
    public double[] getVelocity() {
        return new double[] { vx, vy };
    }

    // Get the cell's acceleration.
    public double[] getAcceleration() {
        return new double[] { ax, ay };
    }

    // Set the cell's position.
    public void setPosition(double dx, double dy) {
        this.dx = dx;
        this.dy = dy;
    }

    // Set the cell's velocity.
    public void setVelocity(double vx, double vy) {
        this.vx = vx;
        this.vy = vy;
    }

    // Set the cell's acceleration.
    public void setAcceleration(double ax, double ay) {
        this.ax = ax;
        this.ay = ay;
    }

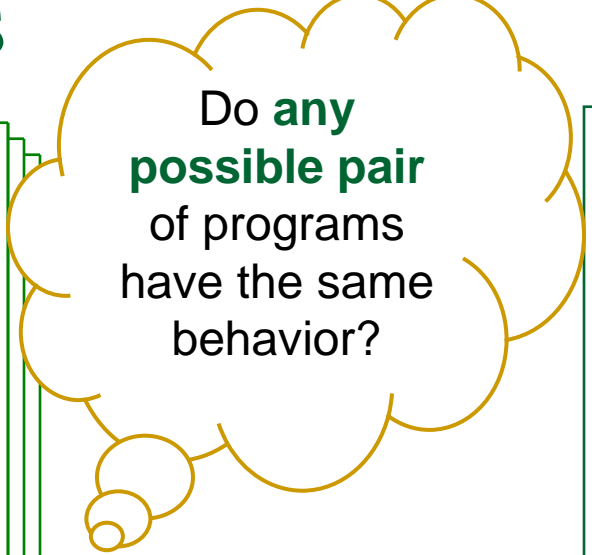
    // Simulate the cell's behavior.
    public void simulate() {
        // Simulate the cell's behavior.
        // This method is a placeholder for the actual simulation logic.
    }
}
```







# A Cell Class



```

// Project: KIMMCLASS
// A. KIMMIS, A. MARANIKIS, A. ADALITSAKIS
// KIMMCLASS.TEX
// KIMMCLASS.TEX (LINES) AND THE KIMMIS, A.D. ADALITSAKIS
// LATEX FILE KIMMCLASS.TEX AND THE KIMMIS, A.D. ADALITSAKIS
// INPUT AND OUTPUT FILES
// INPUT: DATA.LIN
// INPUT: DATA.LIN
// INPUT: DATA.LIN
// INPUT: DATA.LIN
// INPUT: DATA.LIN

public class KimmClass
{
private Document doc = null;
private String[] programNames = null; // See KIMMCLASS.TEX
public KimmClass(int[] rowIndices, int[] colIndices, String[] programNames)
{
try {
this.doc = new Document();
getProgramNames(programNames);
this.programNames = programNames;
}
catch (Exception e) {
System.out.println("Error in constructor KimmClass");
return;
}
}

private void printRowIndices(int row)
{
String rowStr = "";
for (int i = 0; i < rowIndices.length; i++)
{
int colIndex = rowIndices[i];
if (colIndex < 0)
break;
}
}

public String getRowIndices() // ADALITSAKIS.TEX
{
String rowStr = "";
int i = 0;
while (i < rowIndices.length)
{
int colIndex = rowIndices[i];
if (colIndex < 0)
break;
}
return rowStr;
}

public void setRowIndices(int[] rowIndices) // ADALITSAKIS.TEX
{
this.rowIndices = rowIndices;
}

public void printColumnIndices(int col)
{
String colStr = "";
for (int i = 0; i < colIndices.length; i++)
{
int rowIndex = colIndices[i];
if (rowIndex < 0)
break;
}
}

public void setColumnIndices(int[] colIndices) // ADALITSAKIS.TEX
{
this.colIndices = colIndices;
}

public void printCellContents() // ADALITSAKIS.TEX
{
String cellStr = "";
int row = 0;
while (row < rowIndices.length)
{
int col = 0;
while (col < colIndices.length)
{
int rowIndex = rowIndices[row];
int colIndex = colIndices[col];
if (rowIndex < 0 || colIndex < 0)
break;
}
}
}

public void printCellContents(int row, int col) // ADALITSAKIS.TEX
{
int rowIndex = rowIndices[row];
int colIndex = colIndices[col];
if (rowIndex < 0 || colIndex < 0)
return;
}

public void printCellContents(int row, int col, String cellStr) // ADALITSAKIS.TEX
{
int rowIndex = rowIndices[row];
int colIndex = colIndices[col];
if (rowIndex < 0 || colIndex < 0)
return;
}

public void printCellContents(int row, int col, String cellStr, int rowStrLen) // ADALITSAKIS.TEX
{
int rowIndex = rowIndices[row];
int colIndex = colIndices[col];
if (rowIndex < 0 || colIndex < 0)
return;
}

public void printCellContents(int row, int col, String cellStr, int rowStrLen, int colStrLen) // ADALITSAKIS.TEX
{
int rowIndex = rowIndices[row];
int colIndex = colIndices[col];
if (rowIndex < 0 || colIndex < 0)
return;
}

public void printCellContents(int row, int col, String cellStr, int rowStrLen, int colStrLen, int rowStrStart, int colStrStart) // ADALITSAKIS.TEX
{
int rowIndex = rowIndices[row];
int colIndex = colIndices[col];
if (rowIndex < 0 || colIndex < 0)
return;
}

public void printCellContents(int row, int col, String cellStr, int rowStrLen, int colStrLen, int rowStrStart, int colStrStart, int rowStrEnd, int colStrEnd) // ADALITSAKIS.TEX
{
int rowIndex = rowIndices[row];
int colIndex = colIndices[col];
if (rowIndex < 0 || colIndex < 0)
return;
}
}

```



```

// Project: KIMMCLASS
// A. KIMMIS, A. MARANIKIS, A. ADALITSAKIS
// KIMMCLASS.TEX
// KIMMCLASS.TEX (LINES) AND THE KIMMIS, A.D. ADALITSAKIS
// LATEX FILE KIMMCLASS.TEX AND THE KIMMIS, A.D. ADALITSAKIS
// INPUT AND OUTPUT FILES
// INPUT: DATA.LIN
// INPUT: DATA.LIN
// INPUT: DATA.LIN
// INPUT: DATA.LIN
// INPUT: DATA.LIN

public class KimmClass
{
private Document doc = null;
private String[] programNames = null; // See KIMMCLASS.TEX
public KimmClass(int[] rowIndices, int[] colIndices, String[] programNames)
{
try {
this.doc = new Document();
getProgramNames(programNames);
this.programNames = programNames;
}
catch (Exception e) {
System.out.println("Error in constructor KimmClass");
return;
}
}

private void printRowIndices(int row)
{
String rowStr = "";
for (int i = 0; i < rowIndices.length; i++)
{
int colIndex = rowIndices[i];
if (colIndex < 0)
break;
}
}

public String getRowIndices() // ADALITSAKIS.TEX
{
String rowStr = "";
int i = 0;
while (i < rowIndices.length)
{
int colIndex = rowIndices[i];
if (colIndex < 0)
break;
}
return rowStr;
}

public void setRowIndices(int[] rowIndices) // ADALITSAKIS.TEX
{
this.rowIndices = rowIndices;
}

public void printColumnIndices(int col)
{
String colStr = "";
for (int i = 0; i < colIndices.length; i++)
{
int rowIndex = colIndices[i];
if (rowIndex < 0)
break;
}
}

public void setColumnIndices(int[] colIndices) // ADALITSAKIS.TEX
{
this.colIndices = colIndices;
}

public void printCellContents() // ADALITSAKIS.TEX
{
String cellStr = "";
int row = 0;
while (row < rowIndices.length)
{
int col = 0;
while (col < colIndices.length)
{
int rowIndex = rowIndices[row];
int colIndex = colIndices[col];
if (rowIndex < 0 || colIndex < 0)
break;
}
}
}

public void printCellContents(int row, int col) // ADALITSAKIS.TEX
{
int rowIndex = rowIndices[row];
int colIndex = colIndices[col];
if (rowIndex < 0 || colIndex < 0)
return;
}

public void printCellContents(int row, int col, String cellStr) // ADALITSAKIS.TEX
{
int rowIndex = rowIndices[row];
int colIndex = colIndices[col];
if (rowIndex < 0 || colIndex < 0)
return;
}

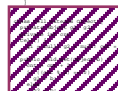
public void printCellContents(int row, int col, String cellStr, int rowStrLen) // ADALITSAKIS.TEX
{
int rowIndex = rowIndices[row];
int colIndex = colIndices[col];
if (rowIndex < 0 || colIndex < 0)
return;
}

public void printCellContents(int row, int col, String cellStr, int rowStrLen, int colStrLen) // ADALITSAKIS.TEX
{
int rowIndex = rowIndices[row];
int colIndex = colIndices[col];
if (rowIndex < 0 || colIndex < 0)
return;
}

public void printCellContents(int row, int col, String cellStr, int rowStrLen, int colStrLen, int rowStrStart, int colStrStart) // ADALITSAKIS.TEX
{
int rowIndex = rowIndices[row];
int colIndex = colIndices[col];
if (rowIndex < 0 || colIndex < 0)
return;
}

public void printCellContents(int row, int col, String cellStr, int rowStrLen, int colStrLen, int rowStrStart, int colStrStart, int rowStrEnd, int colStrEnd) // ADALITSAKIS.TEX
{
int rowIndex = rowIndices[row];
int colIndex = colIndices[col];
if (rowIndex < 0 || colIndex < 0)
return;
}
}

```





# A Cell Class

Do any possible pair of programs have the same behavior?

```
// Project: Kikimora
// Author: A. Kikimora, A. Kikimora
// Description: A Kikimora
// License: MIT License
// Copyright (c) 2018, A. Kikimora, A. Kikimora
// All rights reserved.

// This program is free software: you can redistribute it and/or modify
// it under the terms of the MIT License.
// You should have received a copy of the MIT License along with
// this program. If not, see <http://www.opensource.org/licenses/mit-license.html>.

// Input: A string s.
// Output: A string t.
// Example:
// Input: "abc"
// Output: "cba"

public class Kikimora {
    private Document doc = null;
    private String s;
    private String t;

    public Kikimora(String s) {
        this.s = s;
    }

    public String getOutput() {
        return t;
    }

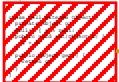
    public void process() {
        // ...
    }
}

// ...

public String getOutput() {
    return t;
}

// ...

public void process() {
    // ...
}
}
```



```
// Project: Kikimora
// Author: A. Kikimora, A. Kikimora
// Description: A Kikimora
// License: MIT License
// Copyright (c) 2018, A. Kikimora, A. Kikimora
// All rights reserved.

// This program is free software: you can redistribute it and/or modify
// it under the terms of the MIT License.
// You should have received a copy of the MIT License along with
// this program. If not, see <http://www.opensource.org/licenses/mit-license.html>.

// Input: A string s.
// Output: A string t.
// Example:
// Input: "abc"
// Output: "cba"

public class Kikimora {
    private Document doc = null;
    private String s;
    private String t;

    public Kikimora(String s) {
        this.s = s;
    }

    public String getOutput() {
        return t;
    }

    public void process() {
        // ...
    }
}

// ...

public String getOutput() {
    return t;
}

// ...

public void process() {
    // ...
}
}
```



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:

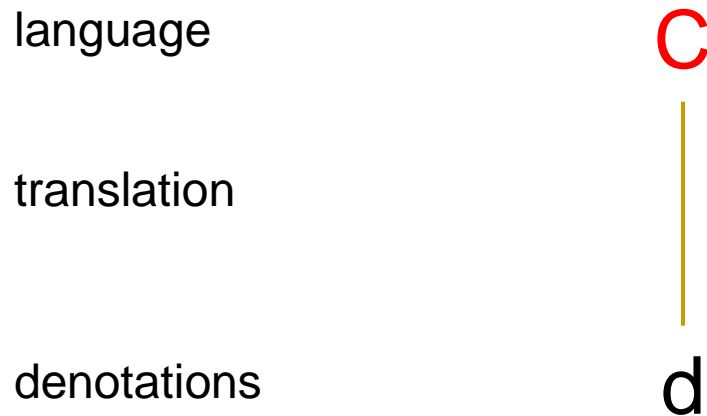
For any class-table context **CT[ ]**:

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

- 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

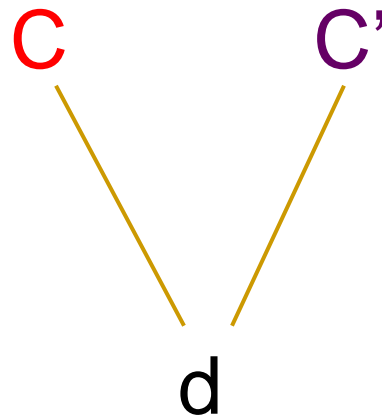
## ■ Denotational Semantics



---

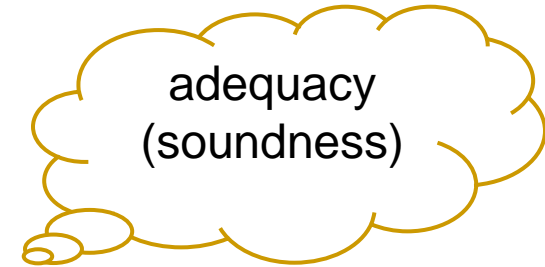
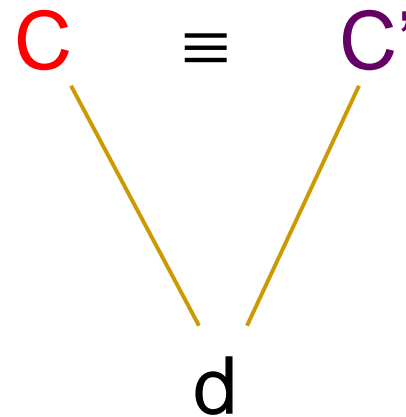
# Technology for Proving Cxt Equivalence

- Denotational Semantics



# Technology for Proving Cxt Equivalence

## ■ Denotational Semantics



---

# Technology for Proving Cxt Equivalence

- Denotational Semantics

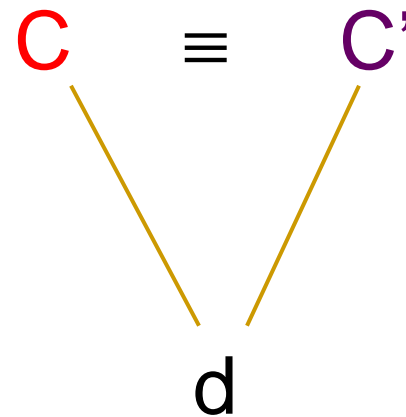
$C \equiv C'$

d



# Technology for Proving Cxt Equivalence

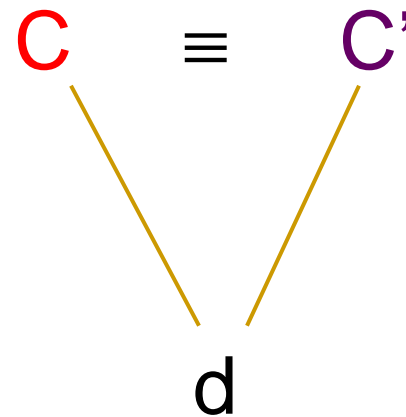
## ■ Denotational Semantics



full abstraction  
(Completeness)

# Technology for Proving Cxt Equivalence

## ■ Denotational Semantics

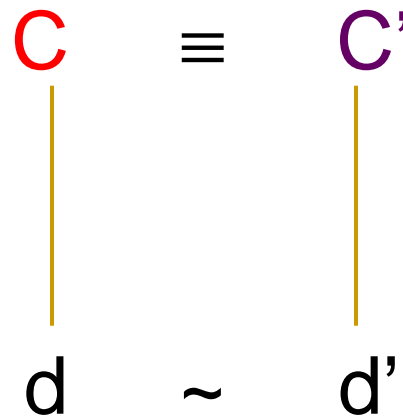


Completeness  
(full abstraction)

- Usually not fully abstract when modelling languages with store

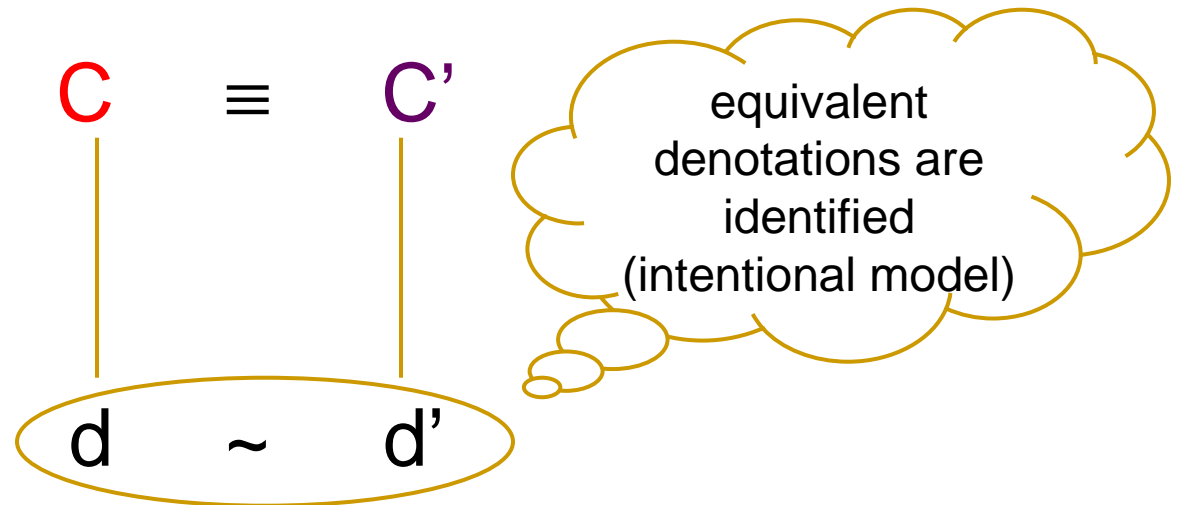
# Technology for Proving Cxt Equivalence

- 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

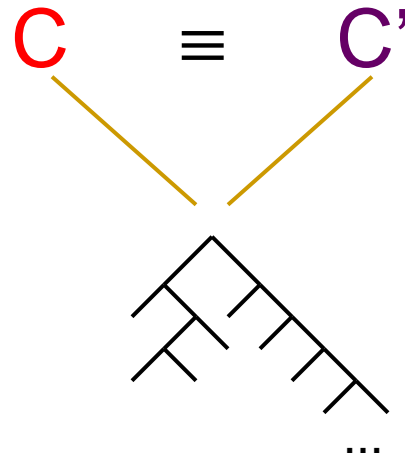
---

# Technology for Proving Cxt Equivalence

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

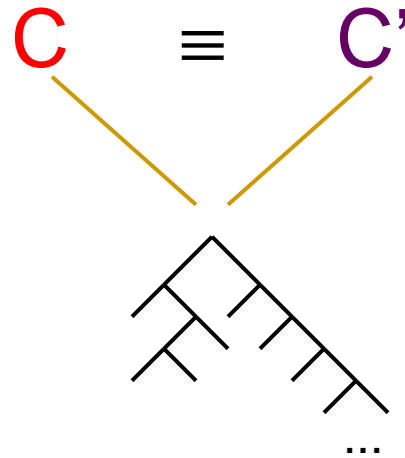
# Technology for Proving Cxt Equivalence

- 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

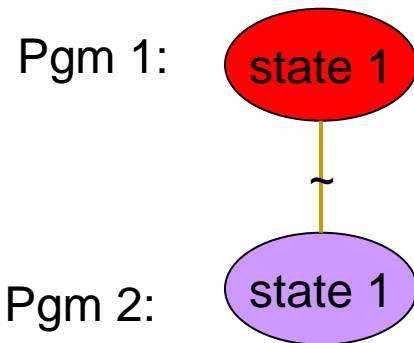
- Cxt Equivalence is the **largest set** of equivalent classes

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

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

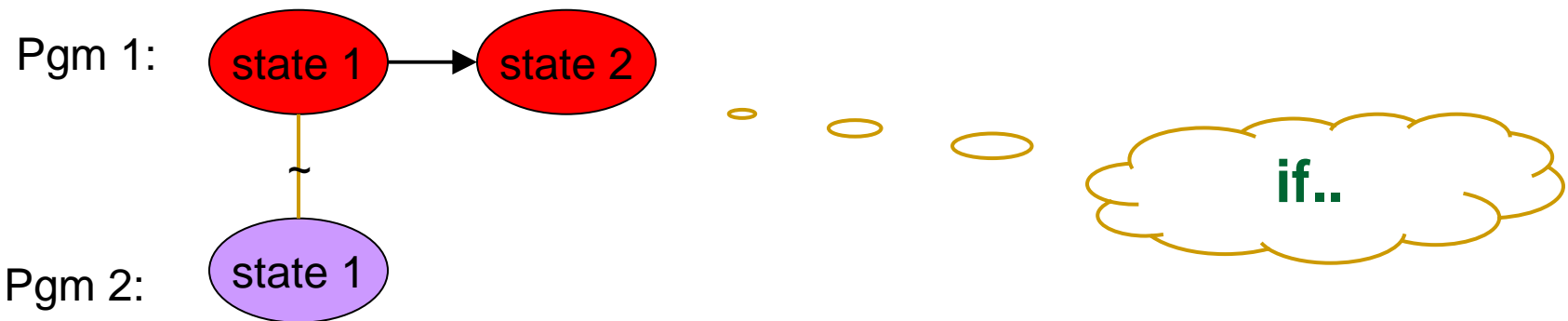
# Bisimulations

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



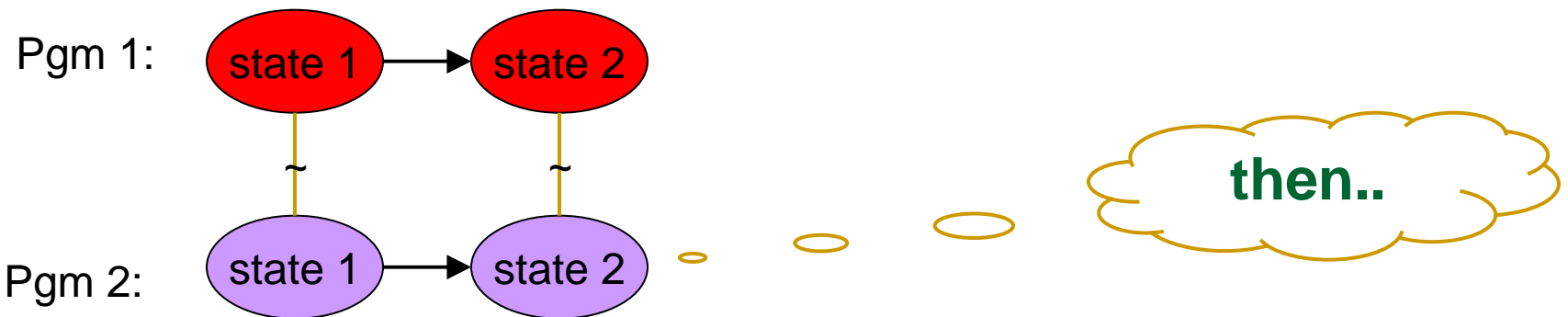
# Bisimulations

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



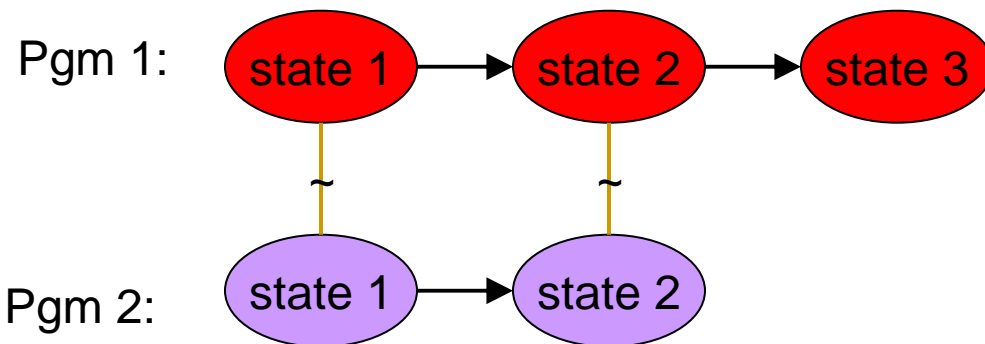
# Bisimulations

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



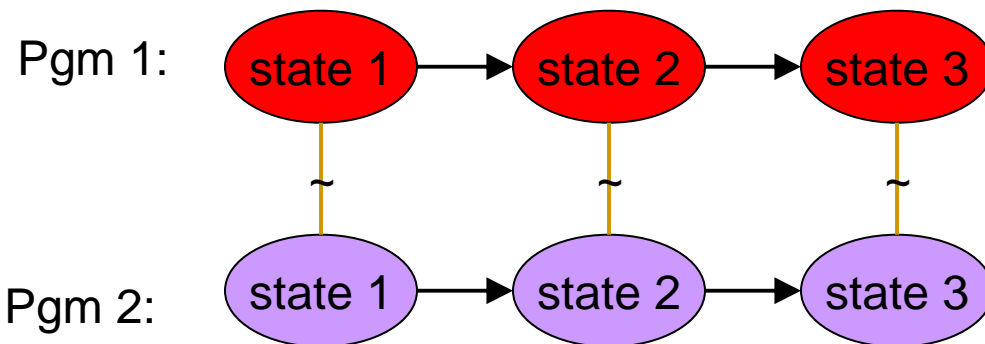
# Bisimulations

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



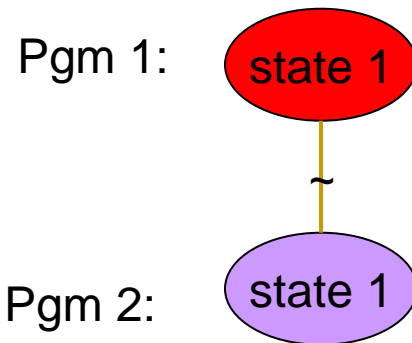
# Bisimulations

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



# Bisimulations

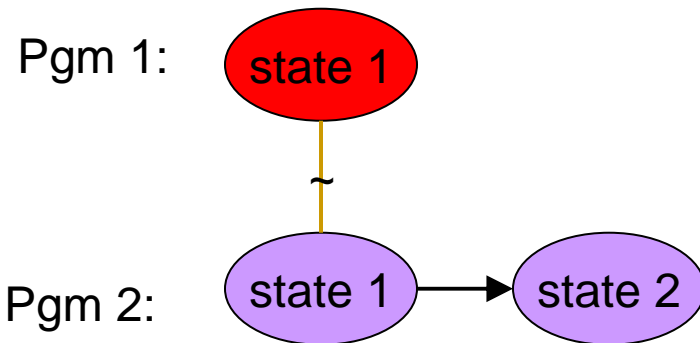
- Introduced in Concurrent Calculi [Hennessy, Milner], and adapted in functional languages [Abramsky].





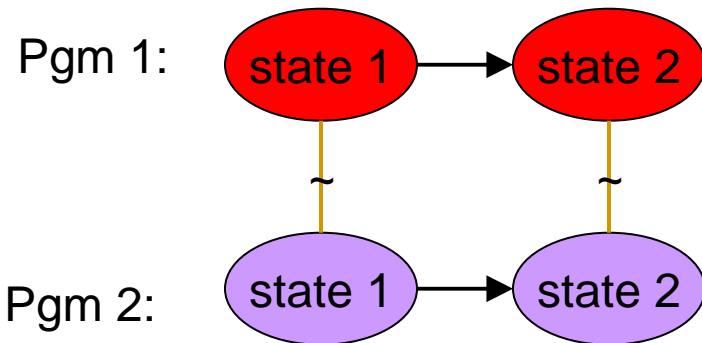
# Bisimulations

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



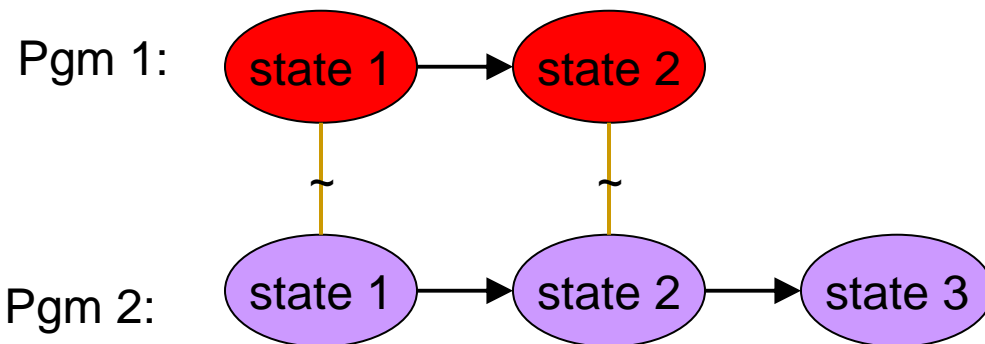
# Bisimulations

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



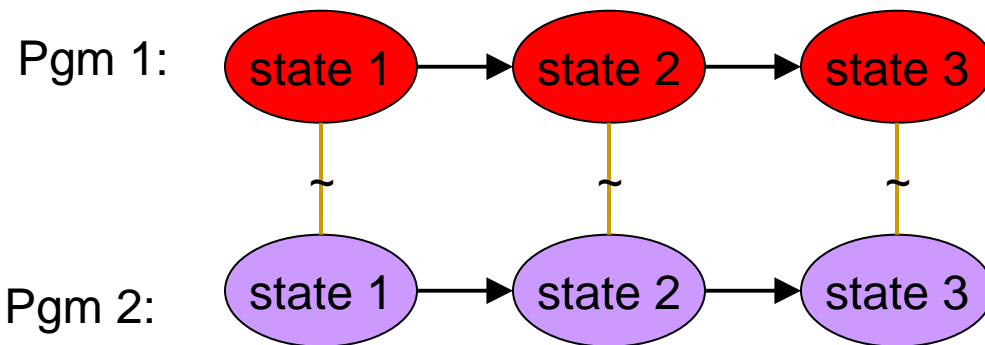
# Bisimulations

- 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[ ]**:

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

---

# Deriving Bisimulations for Cxt Equivalence

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

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

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

# Deriving Bisimulations for Cxt Equivalence

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

For any class-table context  $\mathbf{CT}[\ ]$ :

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

Define Adequacy:

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

For any class-table context  $\mathbf{CT}[\ ]$ :

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

# Deriving Bisimulations for Cxt Equivalence

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

For any class-table context  $\mathbf{CT}[\ ]$ :

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

Define Adequacy:

and the reverse

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

For any class-table context  $\mathbf{CT}[\ ]$ :

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

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



# Deriving Bisimulations for Cxt Equivalence

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

For any class-table context  $\mathbf{CT}[\ ]$ :

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

Define Adequacy:

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

For any class-table context  $\mathbf{CT}[\ ]$ :

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



This is a **big-step bisimulation**

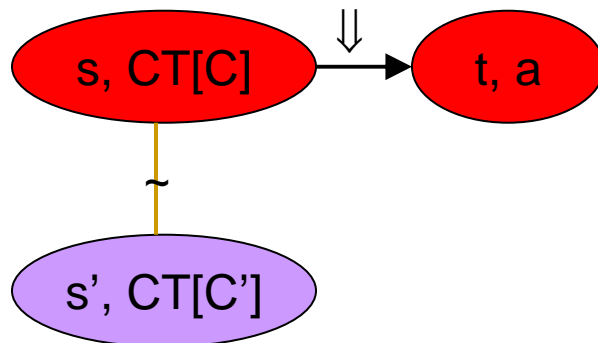
# Deriving Bisimulations for Cxt Equivalence

Define Adequacy:

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

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

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



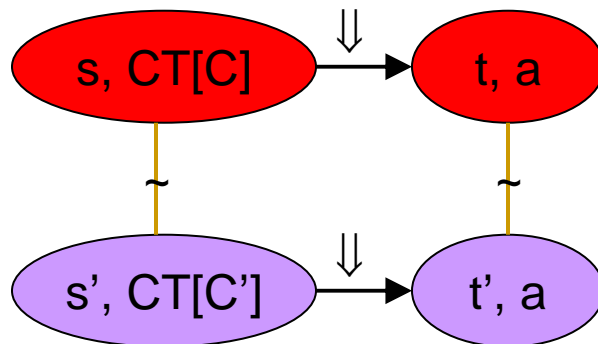
# Deriving Bisimulations for Cxt Equivalence

Define Adequacy:

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

For any class-table context  $\mathbf{CT}[\ ]$ :

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



# Deriving Bisimulations for Cxt Equivalence

- Thm (Soundness & Completeness):

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

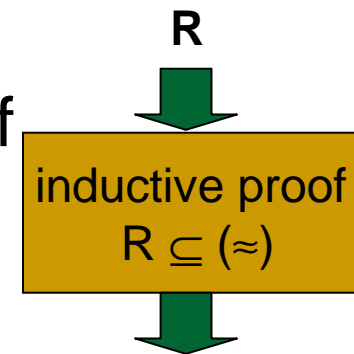
- To prove C and C' equivalent we must:
  - provide a set R
  - $(\mathbf{0}, \mathbf{0}, \mathbf{C}, \mathbf{C}') \in R$
  - $R \subseteq (\approx)$ , by a **standard** inductive proof

# Deriving Bisimulations for Cxt Equivalence

- Thm (Soundness & Completeness):

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

- 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



Yes / Counter-example

# Deriving Bisimulations for Cxt Equivalence

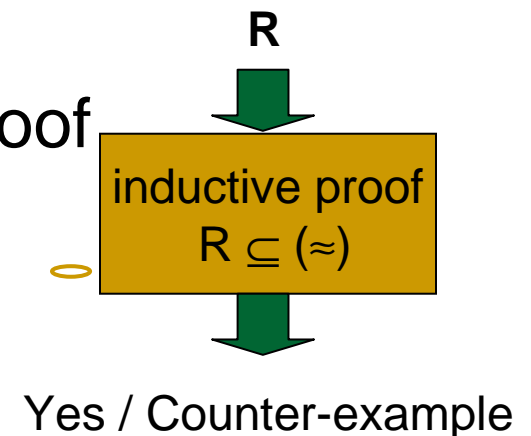
- Thm (Soundness & Completeness):

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

- 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

We improve over this method



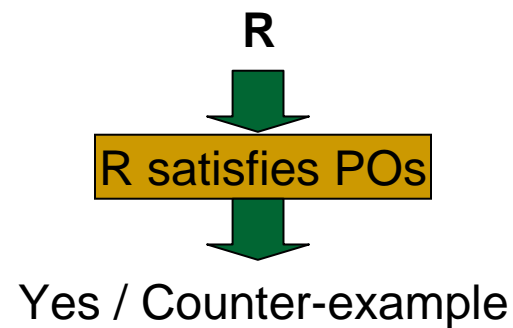
---

# 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  $R = \{ (s, s', \text{Cell}, \text{Cell}') \mid$   
     $s = [l = \text{obj Cell}\{g = \mathbf{l_1}\}] \cdot s_0$   
     $s' = [l = \text{obj Cell}\{g_1 = \mathbf{l_1}, g_2 = l_2',$   
             $\text{cnt} = \mathbf{2n}\}] \cdot s_0' \dots \}$

**PO 6:** Must show that if

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

then

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

and  $(t, t', \text{Cell}, \text{Cell}') \in R$

# Proving the Equivalence of Cells

Let  $R = \{ (s, s', \text{Cell}, \text{Cell}') \mid$   
     $s = [l = \text{obj Cell}\{g=l_1\}] \cdot s_0$   
     $s' = [l = \text{obj Cell}\{g_1=l_1, g_2=l_2',$   
             $\text{cnt}=2n\}] \cdot s_0' \dots \}$

**PO 6:** Must show that if

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

then

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

and  $(\mathbf{s}, \mathbf{s}', \text{Cell}, \text{Cell}') \in R$

which is **true**.

# Proving the Equivalence of Cells

Let  $R = \{ (s, s', \text{Cell}, \text{Cell}') \mid$

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

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

OR

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

Must show that for all

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

then

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

and  $(t, t', \text{Cell}, \text{Cell}') \in R$

# Proving the Equivalence of Cells

Let  $R = \{ (s, s', \text{Cell}, \text{Cell}') \mid$

$t$

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

OR

$t'$

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

Must show that for all

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

then

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

and  $(t, t', \text{Cell}, \text{Cell}') \in R$

---

Τέλος