
ORACLE®

A Conclusion on Inheritance Anomaly

Why Inheritance Anomaly Is Not Worth Solving
Gramoli and Santosa, ICOOOLPS '14

Andrew E. Santosa
PMTS
Oracle Labs Australia
11 November 2014

The following is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

Inheritance Anomaly

- Inheritance of concurrent code breaks encapsulation [MY93]
- Solutions never adopted in mainstream languages → *Why?*

Our Observation

- Reasonable solution makes concurrency control more powerful
  - Complexity in checking Liskov-Wing substitutability
- Practitioners avoid the general problem by avoiding implementation inheritance [Gang of Four]

Example: Java Bounded Buffer

- **`put`**: Puts object into the buffer, suspends when full
- **`get`** (not shown): Retrieves object from buffer, suspends when empty

```
public class BBuf {
  protected int state;
  protected static final int EMPTY = 0;
  protected static final int PARTIAL = 1;
  protected static final int FULL = 3;
  public BBuf(int max) { ...
        state = EMPTY; }
  public synchronized void put(Object v)
        throws Exception {
        while (state==FULL) { wait(); }
        ...
        state = (current>=MAX? FULL : PARTIAL);
        notifyAll();
  } ...
}
```

Subclassing the Bounded Buffer

- **`get2`**: Retrieves 2objects from the buffer, suspends when full / only one

```
public class XBuf2 extends BBuf {
  protected static final int ONE = 4;
  public XBuf2(int max) { super(max) }
  ...
  public synchronized Object[] get2()
        throws Exception {
        while (state==EMPTY||state==ONE) { wait(); }
        ...
        state = (current<=0? EMPTY : PARTIAL);
        notifyAll();
        return ret;
  }
}
```

Subclassing the Bounded Buffer

- Requires redefinition of **`put, get.`**
- **No encapsulation**
- **No reusability**

```
public class XBuf2 extends BBuf {
  protected static final int ONE = 4;
  ...
  public synchronized void put(Object v)
        throws Exception {
        while (state==FULL) { wait(); }
        ...
        state = (current==1? ONE :
                (current>=MAX? FULL : PARTIAL));
        notifyAll();
  }
  ...
}
```

Inheritance Anomaly Solutions

- Increases the power of concurrency control
  - Ad-hoc constructs
  - Regular expressions
  - Temporal logic

- Solutions never adopted in mainstream languages → *Why?*

Too Much Power?

- JEEG as an example
- Elegant solution with temporal logic
- Deadlock introduced: superclass behavior *not preserved*

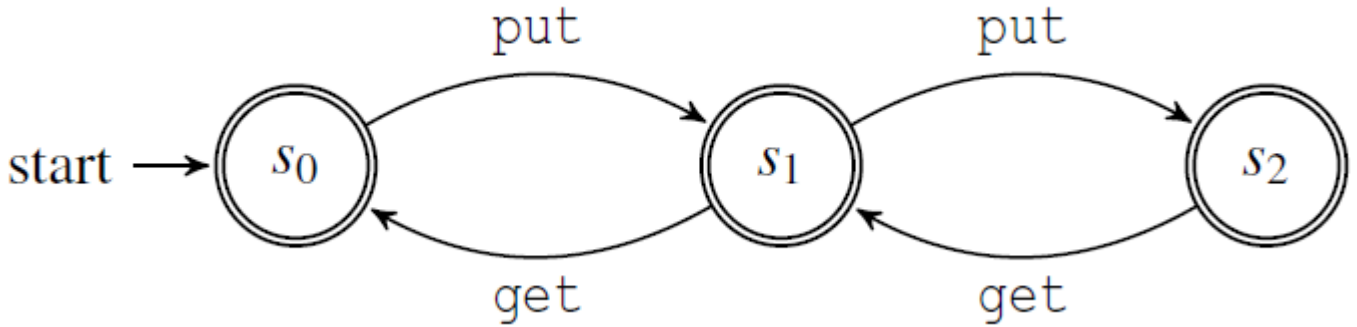
```
public class NewBBuf extends BBuf {
  sync {
       put: (super.putConstr) &&
              (Previous event==get);
       get: (super.getConstr) &&
              (Previous event==put);
  }
}
```

Formal Model of Concurrent Code Inheritance

- Based on NFA
- Not anomalous wrt. all three inheritance anomaly examples in [MY93]

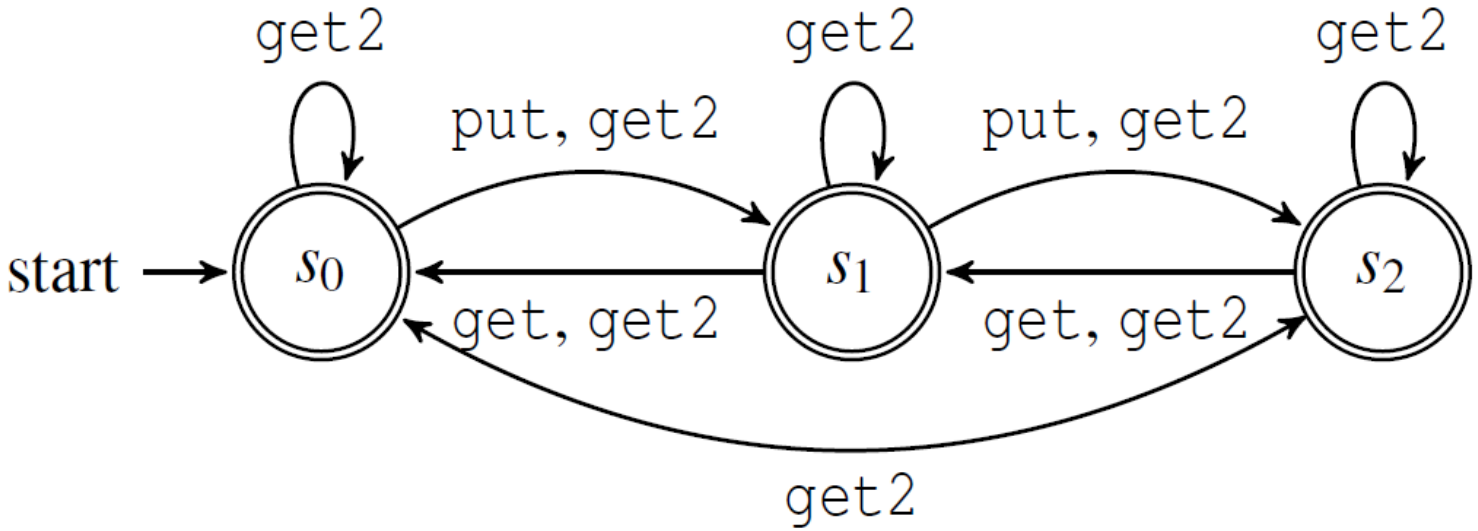
Formal Model of Concurrent Code Inheritance

- **Superclass behavior**
- Interface extension
- Behavior restriction



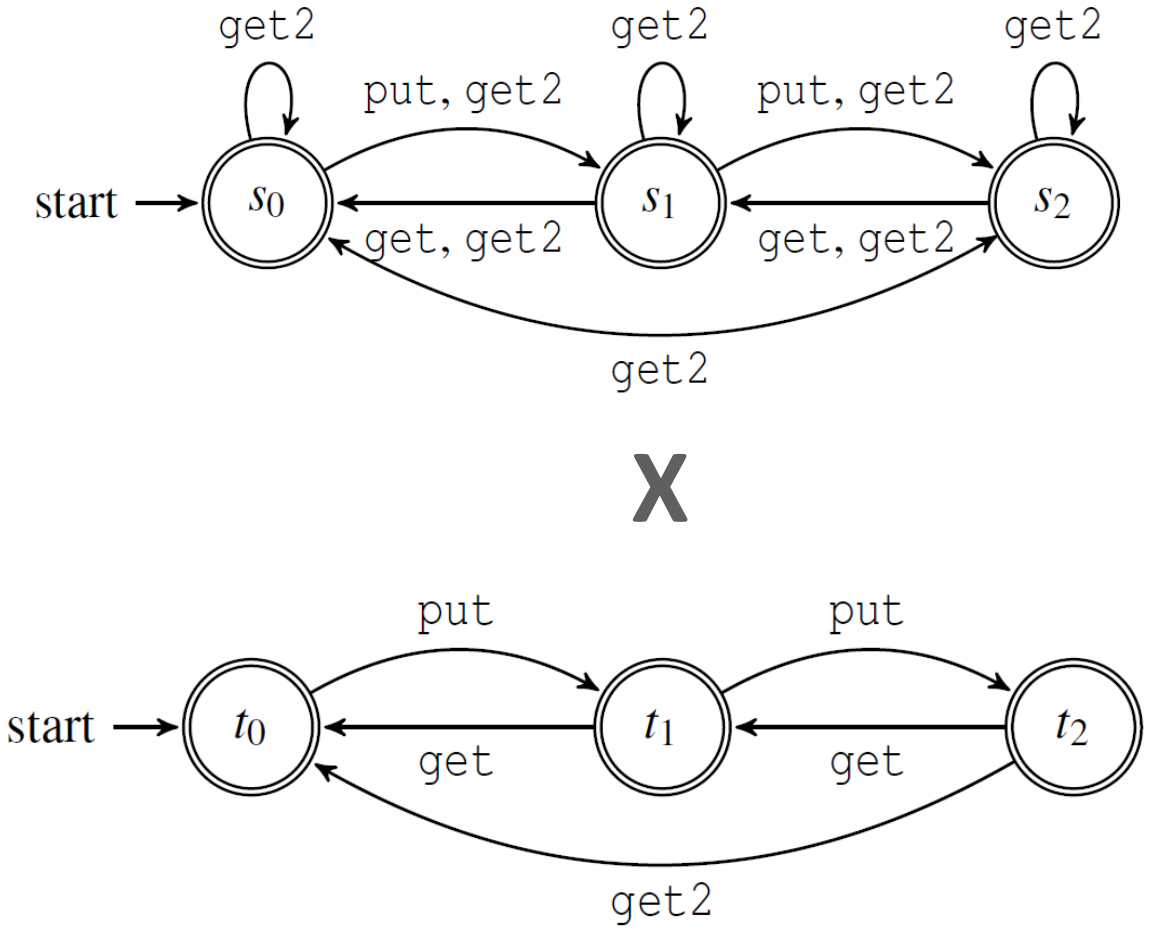
Formal Model of Concurrent Code Inheritance

- Superclass behavior
- **Interface extension**
- Behavior restriction



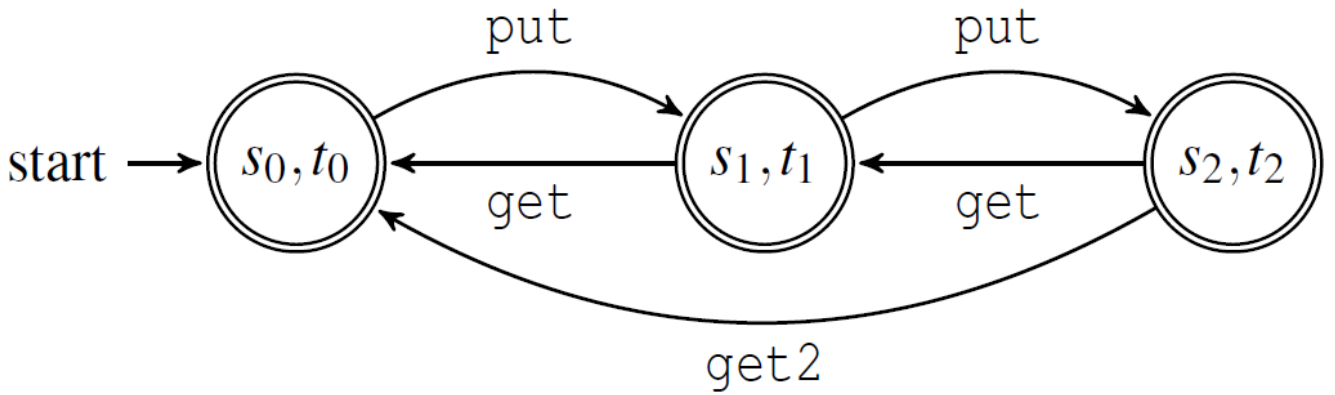
Formal Model of Concurrent Code Inheritance

- Superclass behavior
- Interface extension
- **Behavior restriction**



Formal Model of Concurrent Code Inheritance

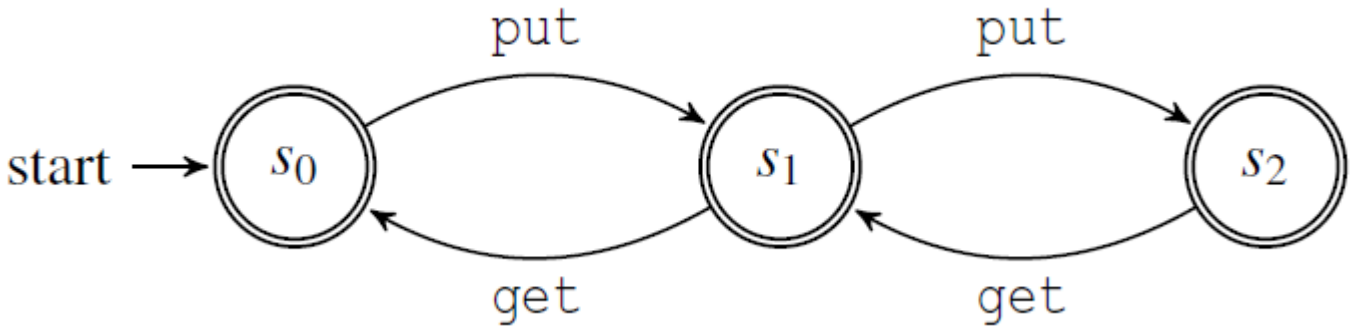
- Superclass behavior
- Interface extension
- **Behavior restriction**



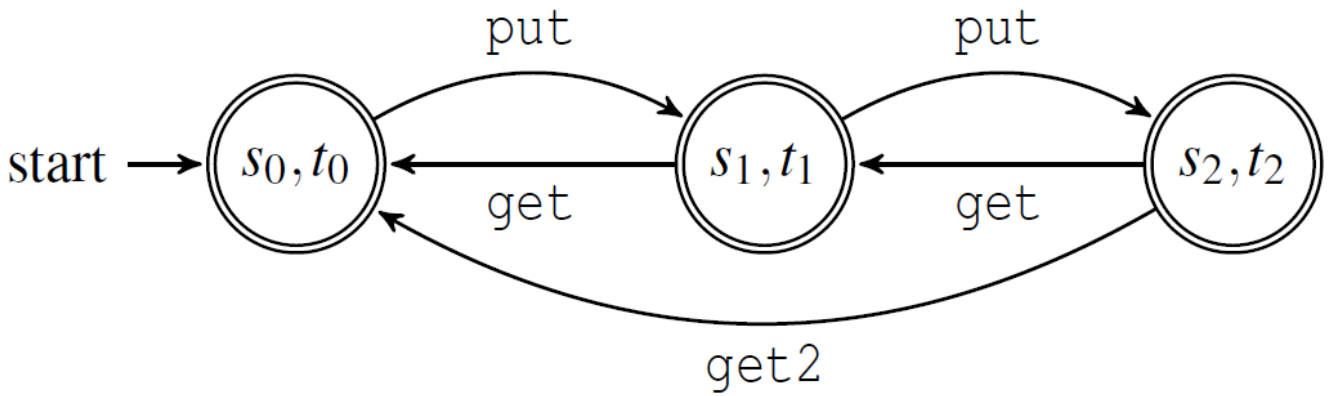
Anomaly Freedom

- A language is ***anomaly free*** iff any subtyping is implementable via ***incremental inheritance*** [CRR98]

- **Incremental inheritance:** Not redefining methods



SUPERTYPE

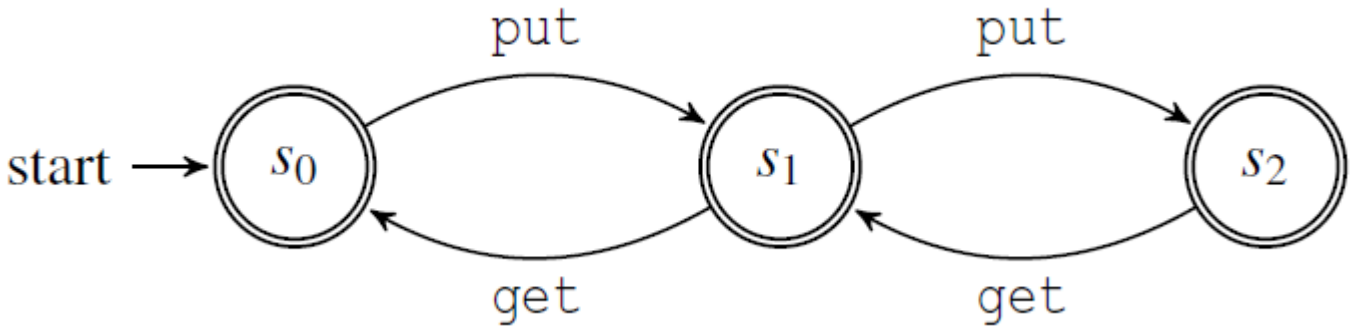


SUBTYPE

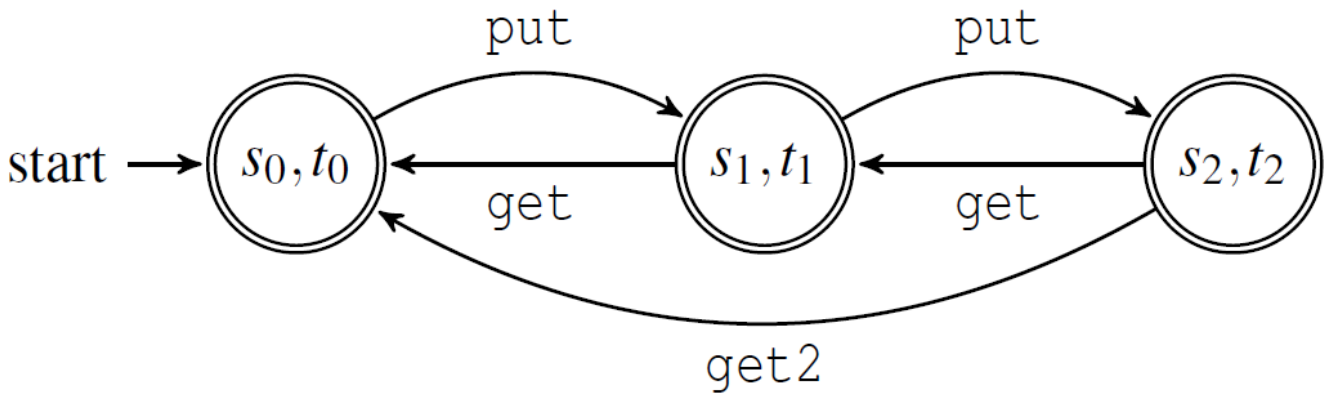
Problem of Anomaly Freedom (Theorem)

- In an anomaly free language, we need to model check to ensure ***behavior preservation*** (PSPACE hard)

- Behavior preservation [CRR98] ensures Liskov-Wing substitutability



SUPERTYPE



SUBTYPE

Inheritance Anomaly as Fragile Base Class Problem

- Tight coupling between subclass and superclass
- Fragile base class solved by programming practice: program to the interface [Gang of Four]
- As much as possible, we implement concurrency at the bottom of the inheritance hierarchy

Conclusion

- Should we still design the next anomaly-free language?
- Hard to ensure subclass objects substitute parent class objects
- For now, this is already tackled by avoiding implementation inheritance

References

- [MY93] S. Matsuoka and A. Yonezawa. Analysis of inheritance anomaly in object-oriented concurrent programming languages. In Research directions in concurrent object-oriented programming, pages 107–150. MIT Press, 1993.

- [CRR98] L. Crnogorac, A. S. Rao, and K. Ramamohanarao. Classifying inheritance mechanisms in concurrent object oriented programming. In 12th ECOOP, pages 571–600. Springer, 1998.

Hardware and Software
Engineered to Work Together


ORACLE®