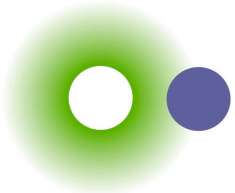
THE UNIVERSITY OF SYDNEY



NICTA

ORACLE®

Why Inheritance Anomaly is Not Worth Solving

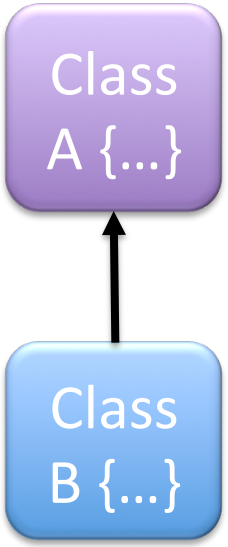
Vincent Gramoli

Andrew Santosa

Context

Object-oriented programming

- Inheritance
  - Classes A, B
  - Inheritance allows code reuse
    - Superclass A
    - Subclass B reuses and extends code of A
- Behavior preservation
  - B implements a subtype of the type of A
  - Subclass B can substitute the superclass A



Concurrent environment

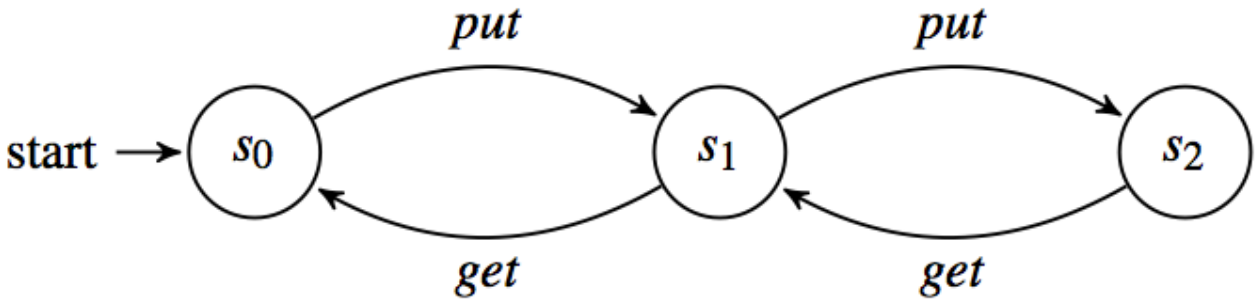
- Class methods can be invoked concurrently
- We assume that methods execute atomically

Context (con't)

We use a finite state automaton to represent each object

- With states as object states
- And transitions as methods

Example: a bounded buffer object BBuf of capacity 2



The *behavior* of an object is the language accepted by its automaton

A language is *behavior preserving* (BP) if its inheritance mechanism guarantees that the behavior of any subclass is a superset of its superclass behavior.

Inheritance Anomaly

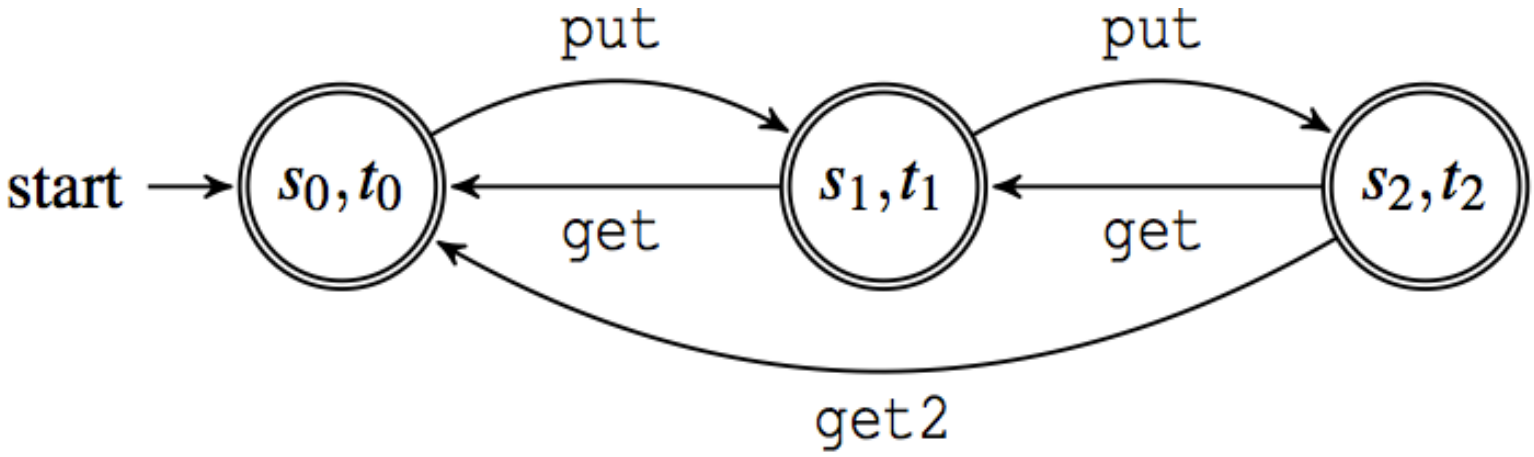
Inheritance Anomaly (IA)

The necessity of redefining some inherited methods to maintain the integrity of concurrent objects [MY93]

Inheritance Anomaly (con't)

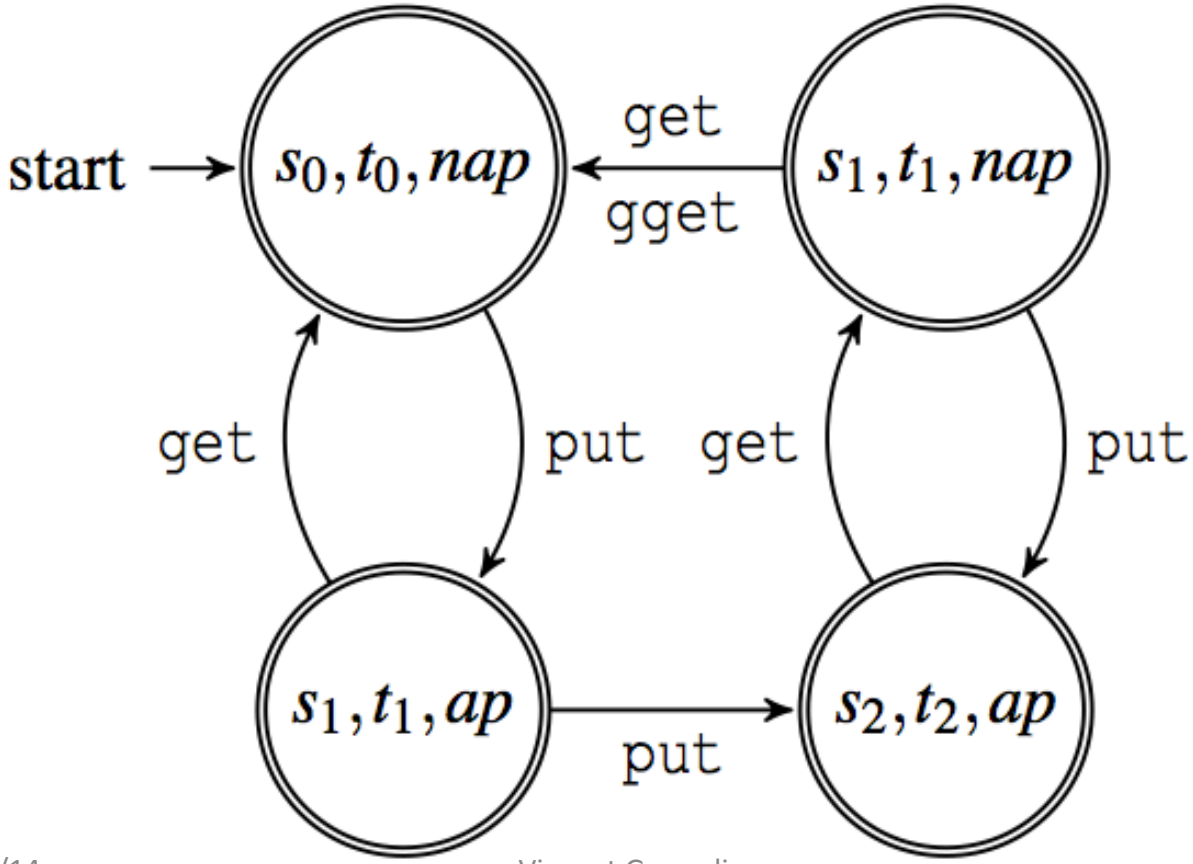
1. Partitioning of states: a new get2 method removes two elements in a row (enabled only if there are >1 elements)

   (put and get should update a counter)



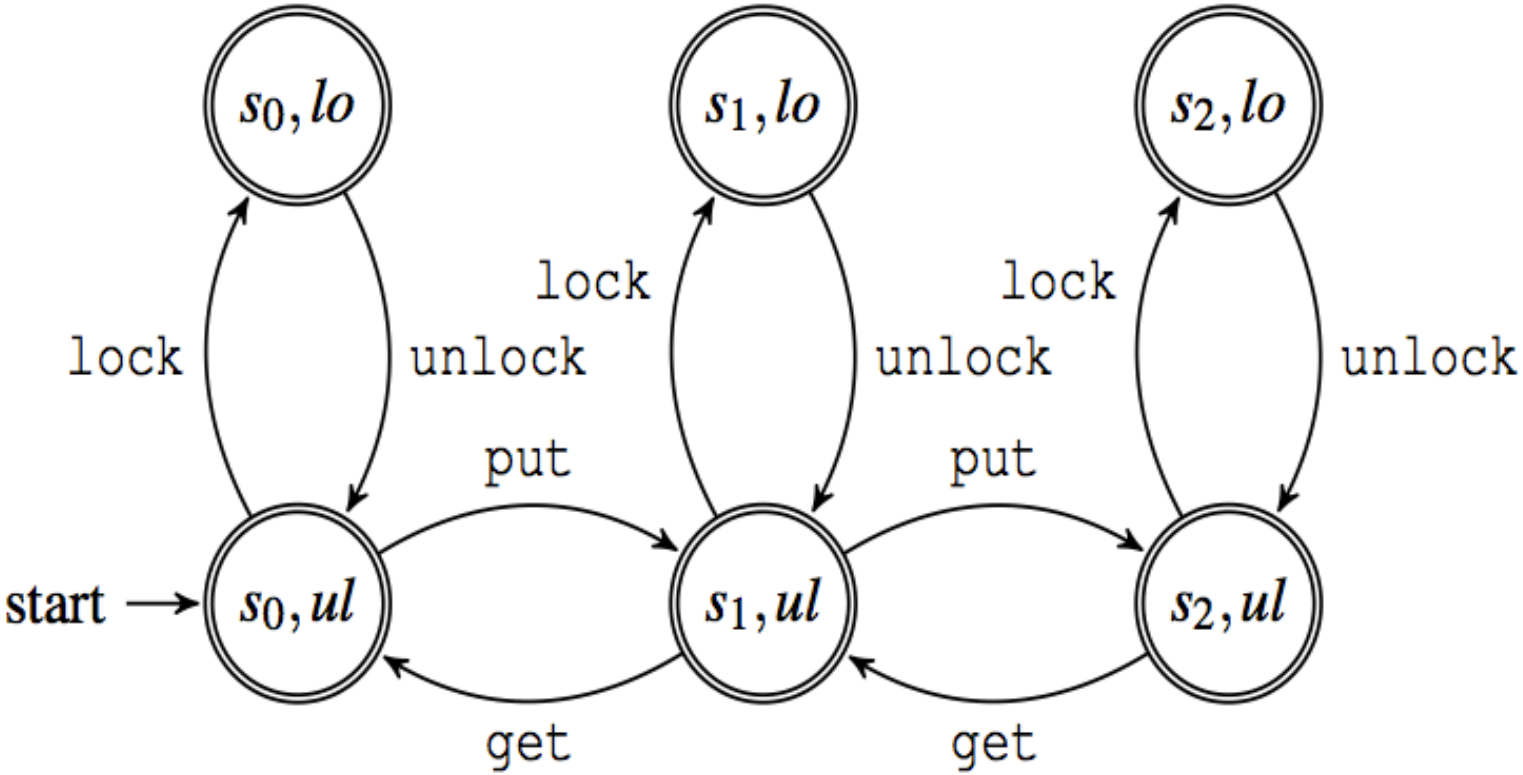
Inheritance Anomaly (con't)

2. History-only sensitivity: a new method gget cannot be executed immediately after a put (put and get should set a flag)



Inheritance Anomaly (con't)

3. Modification of states: combining a lock object with the buffer to enable/disable methods (put and get should check the lock)



Problem

- IA has been known for more than 2 decades
- There exist IA-free (IAF) languages
- IAF languages are not used in practice
- Why?

Languages are either IAF or BP

IAF languages that are not BP

Inheritance anomaly-freedom (IAF) but no behavior preservation (BP) in guard-based languages (e.g., Jeeg [MS05])

```
public class BBuf {
  sync {
    put : (state != FULL);
    get : (state != EMPTY);
  }
  ...
  public void put(Object v)
    throws Exception {
    buf[current] = v;
    current++;
    state = (current>=MAX? FULL : PARTIAL);
  }
  ...
}
```

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

IAF languages that are not BP

Inheritance anomaly-freedom (IAF) but no behavior preservation (BP) in guard-based languages (e.g., Jeeg [MS05])

```
public class BBuf {
  sync {
    put : (state != FULL);
    get : (state != EMPTY);
  }
  ...
  public void put(Object v)
    throws Exception {
    buf[current] = v;
    current++;
    state = (current>=MAX? FULL : PARTIAL);
  }
  ...
}
```

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

Methods put and get are disabled initially

BP languages that are not IAF

Behavior preservation (BP) but no Inheritance anomaly freedom (IAF) in Eiffel [Mey96]

- Design-by-contract for behavior preservation
- Pre and post conditions for methods
- Method redefinition satisfies assertion redeclaration rules:
  - Preconditions can only be weakened
  - Postconditions can only be strengthened

BP languages that are not IAF

Behavior preservation (BP) but no Inheritance anomaly freedom (IAF) in Eiffel [Mey96]

- Design-by-contract for behavior preservation
- Pre and post conditions for methods
- Method redefinition satisfies assertion redeclaration rules:
  - Preconditions can only be weakened
  - Postconditions can only be strengthened

IA-freedom (for modification of states) requires that the method guard be strengthened so that methods are enabled only when the object is unlocked

IAF and BP are incompatible

Proof sketch

1. We first characterize languages, called *sufficiently behavior preserving (SBP)* that cannot be inheritance-anomaly free.

   Lemma: Sufficient behavior preservation (SBP) and inheritance anomaly freedom (IAF) do not coexist in any inheritance mechanism.

   Intuitively, an inheritance mechanism is SBP iff it imposes another restriction in addition to the typing relationships between the subclass and superclass. Such a strong behavior preservation, however, conflicts with IAF.

Proof sketch (con't)

2. Then we can show that if a language is not SBP but IAF then ensuring that it is BP is at least as hard as ensuring regular language containment.

Theorem: Ensuring behavior preservation when an inheritance mechanism is anomaly free is PSPACE hard.

- By contradiction: assume a language is IAF and BP
- By Lemma 1, it cannot be SBP
- If not SBP, then regular language must be contained into another
- The regular language containment problem is PSPACE-hard

In Practice

- Composition rather than inheritance
  - Solve fragile base class problem [GHJV94]
  - Not all synchronization techniques are composable
- Separating synchronization from methods
  - Redefine synchronization (e.g., through guards)
  - Transaction polymorphism [ECOOP'14]

Conclusion

Why are IAF languages not used in practice?

- They cannot ensure behavior preservation
- Behavior preservation is more appealing

This leads to bug-prone concurrent OO programs

In practice, to avoid bugs:

- Either inheritance should be avoided
- Or histories should be constrained by the synchronization

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,* p.107–150. MIT Press, 1993.
- [GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley Longman, 1994.
- [LW'94] B. H. Liskov and J. M. Wing. A behavioral notion of subtyping. *ACM Trans. Prog. Lang. Sys.,* 16(6):1811–1841, Nov. 1994.
- [Mey96] B. Meyer. *Object-Oriented Software Construction.* Prentice Hall PTR, 2nd edition, 1996.
- [ECOOP'98] L. Crnogorac, A. S. Rao, and K. Ramamohanarao. Classifying inheritance mechanisms in concurrent object oriented pro- gramming. In *12th ECOOP,* p.571–600. Springer, 1998.
- [MS05] G. Milicia and V. Sassone. Jeeg: temporal constraints for the synchronization of concurrent objects. *Concurrency - Practice and Experience,* 17(5–6):539–572, 2005.
- [ECOOP'14] Gramoli and Guerraoui. Reusable Concurrent Data Types. ECOOP 2014.