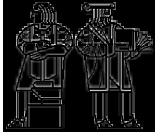


## 6.170 Lecture 14 More on Reasoning



John Guttag  
MIT EECS



Sorry

**I got carried away last week**  
Tried to cover to much too quickly

**Some typos on my slides**  
Fixed in version put online



### Goal of Formal Material

**Enhanced understanding**  
Better programs  
More efficient programming

**Focus on concepts and motivation**  
Not formal definitions



### Plan for Today

**Revisit**  
Invariants  
Reasoning about uses of data abstraction  
Exposing the representation

**Reasoning about subtypes**

**Loop invariants**  
Deferred to later in term



### Invariants

**A predicate that is “always” true**  
More precisely always true in certain contexts

**Rep invariant**  
Property of implementation  
True when not executing an operation of type  
R must hold only on entry and exit of ADT ops  
Need not hold in the middle, or for private methods  
E.g., consider array rep with sorted constraint

**Abstract invariant**  
Property of specification  
Used to understand client programs



### An Abstract Invariant Example

Overview: HorseSets are finite sets of colored horses

```
public HorseSet ()  
    effects: creates a fresh, empty HorseSet  
public void insert (Horse h);  
    modifies: this  
    effects: this' =  
        if  $\exists h1 \in S$  s.t.  $h1.color = h.color$   
        then this  
        else this  $\cup \{h\}$   
public boolean member (horse h);  
    effects: returns (h in this)  
public int size ();  
    effects: returns |this|
```



**Conjecture:**  $|S| > 1 \Rightarrow (\exists h1, h2 \in S [h1.color \neq h2.color])$

**6.170 A Proof**


**Prove:**  $|S| > 1 \Rightarrow (\exists h1, h2 \in S [h1.color \neq h2.color])$   
**Basis:**  $S = \{ \}$ , vacuously true  
**Induction (on insert):**  
**Assume:**  $|S| > 1 \Rightarrow (\exists h1, h2 \in S [h1.color \neq h2.color])$   
**Show:**  $|S1| > 1 \Rightarrow (\exists h3, h4 \in S1 [h3.color \neq h4.color])$ ,  
 where  $S1 = \{ h \mid \exists h5 \in S \text{ s.t. } h5.color = h.color \text{ then } S \text{ else } S \cup \{ h \}$   
**If  $S1 = S$ , then it holds by induction hypothesis**  
**If  $S1 = S \cup \{ h \}$ , there are three cases to consider**  
 $|S| > 1$ : If  $S$  has horses of different colors,  
 by the meaning of union, so does  $S \cup \{ h \}$   
 $|S| = 0$ : Vacuous case, since hypothesis of thm. false  
 $|S| = 1$ : We know that  $S$  did not contain a horse of  $h$ .color,  
 so again we know thm. holds by meaning of union

© John Chapin/John Guttag Spring 2000 Slide 7

**6.170 An Old Joke**

**Prove**  $\forall h1, h2 \in S [h1.color = h2.color]$   
 I.e., there is no such thing as a horse of a different color  
**Basis:**  $S = \{ \}$   
**Induction (on |S|):**  
**Assume:**  $\forall h1, h2 \in S [h1.color = h2.color]$   
**Show:**  $\forall h3, h4 \in S1 [h3.color = h4.color]$ , where  $|S1| = |S| + 1$   
 Remove any element from  $S1$ , since result is of  $|S|$ , all horses  
 are the same color  
 Put that element back in, and remove a different element. Again,  
 all horses are the same color.  
 By transitivity of same color, all horses are the same color

**So, what is the flaw in the proof?**  
**Didn't show that there were at least two horses in set**  
 Therefore transitivity doesn't apply



© John Chapin/John Guttag Spring 2000 Slide 8

**6.170 (Not) Exposing the Rep**

**Tricky to define formally**  
 Won't try and do it here

**Important to understand motivation**  
 Source of many hard to find errors

**Want to ensure**  
 Correctness of implementation a local property  
 Can use induction to reason about implementation  
 Correctness of clients does not depend up rep  
 Can use induction to reason about clients

**If you can't do this**  
 You may have exposed the rep  
 Which is in very poor taste

© John Chapin/John Guttag Spring 2000 Slide 9

**6.170 An Example**

Class AorBSet {  
 Mutable sets of AorB objects (also mutable)  
 public AorBset ( )  
 effects makes a new, empty, AorBSet  
 public insert(AorB e)  
 modifies self  
 effects self' = self U {e}  
 public boolean member(AorB e)  
 effects returns  $e \in \text{self}$   
 public AorB choose( )  
 requires self is not empty  
 effects returns e s.t.  $e \in \text{self}$   
 public toVector( )  
 effects returns a vector containing  
 the elements in self

```
public insert(AorB e) {
  elts.addElement(e);
}

public insert(AorB e) {
  AorB ee;
  ee = new AorB(e.val);
  elts.addElement(ee);
}

public toVector( ) {
  return elts;
}
```

© John Chapin/John Guttag Spring 2000 Slide 10

**6.170 An Example**

Class AorBSet {  
 Mutable sets of AorB objects (also mutable)  
 public AorBset ( )  
 effects makes a new, empty, AorBSet  
 public insert(AorB e)  
 modifies self  
 effects self' = self U {e}  
 public boolean member(AorB e)  
 effects returns  $e \in \text{self}$   
 public AorB choose( )  
 requires self is not empty  
 effects returns e s.t.  $e \in \text{self}$   
 public toVector( )  
 effects returns a vector containing  
 the elements in self

```
public insert(AorB e) {
  elts.addElement(e);
  e.setToA();
}
```

© John Chapin/John Guttag Spring 2000 Slide 11

**6.170 An Example**

Class AorBSet {  
 Mutable sets of AorB objects (also mutable)  
 public AorBset ( )  
 effects makes a new, empty, AorBSet  
 public insert(AorB e)  
 modifies self  
 effects self' = self U {e}  
 public boolean member(AorB e)  
 effects returns  $e \in \text{self}$   
 public AorB choose( )  
 requires self is not empty  
 effects returns e s.t.  $e \in \text{self}$   
 public toVector( )  
 effects returns a vector containing  
 the elements in self

```
public insert(AorB e) {
  elts.addElement(e);
}

public choose( ) {
  return
  elts.elementAt(0);
}
```

© John Chapin/John Guttag Spring 2000 Slide 12

# 6.170 Spring 2000

**6.170 An Example**

Class AorBSet {  
 Mutable sets of AorB objects  
 public AorBset ()  
 effects makes a new, empty, AorBSet  
 public insert(AorB e)  
 modifies self  
 effects self' = self U {e}  
 ...  
 public AorB choose()   
 requires self is not empty  
 effects returns e s.t. e ∈ self  
 public boolean hasA()  
 effects returns true iff this contains e  
 s.t. e.val = A

```
public insert(AorB e) {
  elts.addElement (e);
}

public choose() {
  return
  elts.elementAt(0);
}
```

© John Chapin/John Guttag Spring 2000 Slide 13

**6.170 An Example**

Class AorBSet {  
 Mutable sets of AorB objects  
 public AorBset ()  
 effects makes a new, empty, AorBSet  
 public insert(AorB e)  
 modifies self  
 effects self' =  
 self U {e1 s.t. e1.val = e.val}  
 ...  
 public AorB choose()   
 requires self is not empty  
 effects returns e s.t. e ∈ self  
 public boolean hasA()  
 effects  
 returns true iff this contains e.  
 s.t. e.val = A

```
public insert(AorB e) {
  elts.addElement (e);
}

public insert(AorB e) {
  AorB ee;
  ee = new AorB(e.val);
  elts.addElement (ee);
}
```

© John Chapin/John Guttag Spring 2000 Slide 14

**6.170 An Example**

Class AorBSet {  
 Mutable sets of AorB objects  
 public AorBset ()  
 effects makes a new, empty, AorBSet  
 public insert(AorB e)  
 modifies self  
 effects self' =  
 self U {e1 s.t. e1.val = e.val}  
 ...  
 public AorB choose()   
 requires self is not empty  
 effects returns e s.t. e ∈ self  
 public boolean hasA()  
 effects  
 returns true iff this contains e.  
 s.t. e.val = A

```
ab.setToA();
s.insert(ab);
ab.setToB();
if (s.hasA())
  print Yes
else print No
```

© John Chapin/John Guttag Spring 2000 Slide 15

**6.170 An Example**

Class AorBSet {  
 Mutable sets of AorB objects  
 public AorBset ()  
 effects makes a new, empty, AorBSet  
 public insert(AorB e)  
 modifies self  
 effects self' =  
 self U {e1 s.t. e1.val = e.val  
 & e1 is a new object}  
 ...  
 public AorB choose()   
 requires self is not empty  
 effects returns e s.t. e ∈ self  
 public boolean hasA()  
 effects  
 returns true iff this contains e  
 s.t. e.val = A

```
public insert(AorB e) {
  elts.addElement (e);
}

public insert(AorB e) {
  AorB ee;
  ee = new AorB(e.val);
  elts.addElement (ee);
}
```

© John Chapin/John Guttag Spring 2000 Slide 16

**6.170 Reasoning About Subtypes**

**Recall that if T1 is a subtype of T2**  
 T1's specification is stronger than T2's  
 Programs that work with T2 will also work with T1  
 Implementations of T1 are implementations of T2

**What does this imply about**  
 Rep invariants?  
 Abstraction functions?

**Two distinct cases**  
 Subtype doesn't correspond to subclass  
 Nothing interesting to say  
 Subtype corresponds to subclass  
 Fair amount to say

© John Chapin/John Guttag Spring 2000 Slide 17

**6.170 Abstraction Function and Subtyping**

**Assume T1 a subtype (and subclass) of T2**  
**Sometimes abstraction functions the same**  
 I.e., extra parts of rep ignored


**Consider, for example** IntSet  
 |  
 IntSetWithExtraOps

**When might AF\_T1 be different?**  
 When additions to rep impact the abstract value

**Consider, for example** Window  
 / \  
 ColoredWindow ShapedWindow

© John Chapin/John Guttag Spring 2000 Slide 18

6.170  
Spring 2000

 **Rep Invariants & Subtyping**

---

**Assume T1 a subtype (and subclass) of T2**

**Is T1's rep invariant**


- Weaker than T2's?
- The same?
- Stronger?
- Incomparable?

**Risky to make it weaker or incomparable**  
Inherited code might break

**Same or stronger OK**

---

© John Chapin/John Guttag      Spring 2000      Slide 19

 **Checking the Rep Invariant of a Subtype**

---

**Suppose it is the same**

- If no access to rep of supertype
  - No way for subtype to break invariant
  - But ops should still call method that checks invariant
- If access to part of rep of supertype
  - Must proceed in usual way

**Suppose it is stronger**

- Don't place constraints on parts of rep used by supertype
- Can't count on inherited code to maintain them
- OK to constrain parts of rep introduced in subtype

---

© John Chapin/John Guttag      Spring 2000      Slide 20