Principles of Software Construction: Objects, Design, and Concurrency

Formal Analysis of Software Artifacts

Jonathan Aldrich    Charlie Garrod
• Piazza Question: How can I automatically test making good moves when the player gets random tiles?

• Answer: Design your game with testing in mind
  • Your ideas?
  • E.g. Provide a method to set the random seed
    • In tests, the seed determines each player’s tiles
    • Look at those tiles, and design your tests accordingly
    • In real executions, seed based on time, or other data
  • E.g. Provide a method to replace tiles with known ones
    • Only used in testing
Testing and Proofs

- Testing (Validation)
  - Observable properties
  - Verify program for one execution
  - Manual development with automated regression
  - Most practical approach now

- Proofs (Verification)
  - Any program property
  - Verify program for all executions
  - Manual development with automated proof checkers
  - Practical for small programs, may scale up in the future

- So why study proofs if they aren’t (yet) practical?
  - Proofs tell us how to think about program correctness
  - Important for development, inspection, dynamic assertions
  - Foundation for static analysis tools
  - These are just simple, automated theorem provers
  - Many are practical today!
Dafny Class Invariants - from the Previous Lecture

```daffny
class SimpleSet {
    var contents: array<int>;
    var size: int;

    function valid() : bool
        reads this, contents;
    {size >= 0
        && contents != null
        && contents.Length >= size }

    valid() represents the class invariant.
    It must be explicitly added to:
    • The constructor postcondition
    • The precondition of functions and methods
    • The postcondition of methods that modify this

    In this case the class invariant asserts that:
    • The size is non-negative
    • The contents array is non-null
    • The contents array is big enough to hold size elements

    A Dafny function is used in specifications. It cannot be called (though function methods can)

    The reads clause of a function defines the objects on whose fields the function’s value depends. This is so Dafny can determine when a function’s value might be affected by a field write.
```
method init(capacity:int)
  requires capacity >= 0;
  modifies this;
  ensures valid();
  ensures size == 0;
  ensures contents.Length == capacity;
  ensures fresh(contents),
{
  contents := new int[capacity];
  size := 0;
}
This is the main effect of the method—changing the mathematical set this object represents.

Specifying a Method in Dafny - from the Previous Lecture

```
method add(i:int)
  requires valid();
  requires size < contents.Length || i in mset();
  modifies this, contents;
  ensures valid();
  ensures mset() == old(mset()) + {i};
  ensures contents == old(contents);
  ensures i in old(mset()) ==> size == old(size);
  ensures !(i in old(mset())) ==> size == old(size)+1;
```
Reference: mset() - from the Previous Lecture

function mset() : set<int>
    reads this, contents;
    requires valid();
{
    set j:int | 0 <= j < size :: contents[j]
}
method contains(i:int) returns (b: bool)

    requires valid();
    ensures b <=> i in mset();
{var j := 0;

while (j < size)
    invariant 0 <= j <= size;
    invariant !(i in set k:int | 0 <= k < j :: contents[k]);
    decreases size - j; {
        if (contents[j] == i) { return true; }
        j := j + 1;
    }

return false;
Behavioral Subtyping (Liskov Substitution Principle)

Let \( q(x) \) be a property provable about objects \( x \) of type \( T \). Then \( q(y) \) should be provable for objects \( y \) of type \( S \) where \( S \) is a subtype of \( T \).

Barbara Liskov

- An object of a subclass should be substitutable for an object of its superclass
- Known already from types:
  - May use subclass instead of superclass
  - Subclass can add, but not remove methods
  - Overridden method must return same or supertype
  - Overridden method may not throw additional exceptions
- Applies more generally to behavior:
  - A subclass must fulfill all contracts that the superclass does
  - Same or stronger invariants
  - Same or stronger postconditions for all methods
  - Same or weaker preconditions for all methods
Behavioral Subtyping (Liskov Substitution Principle)

abstract class Vehicle {
    int speed, limit;
    //@ invariant speed < limit;

    //@ requires speed != 0;
    //@ ensures |speed| < |\old{speed}|
    void break();
}

class Car extends Vehicle {
    int fuel;
    boolean engineOn;
    //@ invariant fuel >= 0;

    //@ requires fuel > 0 && ! engineOn;
    //@ ensures engineOn;
    void start() { … }
    void accelerate() { … }

    //@ requires speed != 0;
    //@ ensures |speed| < |\old{speed}|
    void break() { … }
}

Subclass fulfills the same invariants (and additional ones)
Overridden method has the same pre and postconditions
Behavioral Subtyping (Liskov Substitution Principle)

class Car extends Vehicle {
    int fuel;
    boolean engineOn;
    //@ invariant fuel >= 0;
    //@ requires fuel > 0 && ! engineOn;
    //@ ensures engineOn;
    void start() { … }
    void accelerate() { … }
    //@ requires speed != 0;
    //@ ensures |speed| < |\old{speed}|
    void break() { … }
}

class Hybrid extends Car {
    int charge;
    //@ invariant charge >= 0;
    //@ requires (charge > 0 || fuel > 0) && ! engineOn;
    //@ ensures engineOn;
    void start() { … }
    void accelerate() { … }
    //@ requires speed != 0;
    //@ ensures |speed| < |\old{speed}|
    //@ ensures charge > \old{charge}
    void break() { … }
}

Subclass fulfills the same invariants (and additional ones)
Overridden method start has weaker precondition
Overridden method break has stronger postcondition
Behavioral Subtyping (Liskov Substitution Principle)

class Rectangle {
    int h, w;
    Rectangle(int h, int w) {
        this.h=h; this.w=w;
    }

    //methods
}

class Square extends Rectangle {
    Square(int w) {
        super(w, w);
    }
}

Is Square a behavior subtype of Rectangle?
Behavioral Subtyping (Liskov Substitution Principle)

```java
class Rectangle {
    //@ invariant h>0 && w>0;
    int h, w;

    Rectangle(int h, int w) {
        this.h=h; this.w=w;
    }

    //methods
}

class Square extends Rectangle {
    //@ invariant h==w;
    Square(int w) {
        super(w, w);
    }
}

Is Square a behavior subtype of Rectangle?
Behavioral Subtyping (Liskov Substitution Principle)

class Rectangle {
    //@ invariant h>0 && w>0;
    int h, w;
    
    Rectangle(int h, int w) {
        this.h=h; this.w=w;
    }
    
    void scale(int factor) {
        w=w*factor;
        h=h*factor;
    }
}

class Square extends Rectangle {
    //@ invariant h==w;
    Square(int w) {
        super(w, w);
    }
}

Is Square a behavior subtype of Rectangle?
class Rectangle {
    //@ invariant h>0 && w>0;
    int h, w;

    Rectangle(int h, int w) {
        this.h=h; this.w=w;
    }

    void scale(int factor) {
        w=w*factor;
        h=h*factor;
    }

    void setWidth(int neww) {
        w=neww;
    }
}

class Square extends Rectangle {
    //@ invariant h==w;
    Square(int w) {
        super(w, w);
    }
}

Is Square a behavior subtype of Rectangle?
Behavioral Subtyping (Liskov Substitution Principle)

class Rectangle {
    //@ invariant h>0 && w>0;
    int h, w;

    Rectangle(int h, int w) {
        this.h=h; this.w=w;
    }

    void scale(int factor) {
        w=w*factor;
        h=h*factor;
    }

    void setWidth(int neww) {
        w=neww;
    }
}

class Square extends Rectangle {
    //@ invariant h==w;
    Square(int w) {
        super(w, w);
    }
}

With these methods, Square is not a behavior subtype of Rectangle

← Invalidates stronger invariant (w==h) in subclass
Formal Analysis Summary

- Specification between textual and formal specifications
- Proving (e.g. with Dafny) vs. Testing
- Class Invariants and Behavioral Subtyping
- Tools such as Dafny can make proofs more practical
  - Reduces effort relative to proof by hand
  - Still considerable work in writing specifications and invariants
  - Can be useful in documenting code and finding errors
  - The current tool may miss some defects