

6.170 Lecture 4 Abstraction and Specification (part 1)



John Guttag
MIT EECS

Two Kinds of Abstraction

Abstraction by parameterization
Beloved by 6.001

Abstraction by specification
Beloved by 6.170



© Michael Ernst/John Guttag Spring 2001 Slide 2

Lambda Abstraction

Replace variables by formal parameters
Actual bound to formal at time of call

Increase the applicability of procedures
Good reason not to hardwire things
E.g., file names
Good reason not to use global or static variables



© Michael Ernst/John Guttag Spring 2001 Slide 3

Potential Drawbacks to Using Parameters

Loss of efficiency
Only if large things need to be copied from actual to formal
Usually only copying a pointer

Long argument lists
Not if things packaged well

Lose state across calls
Issue if state associated with procedure rather than objects
Why we have static variables

© Michael Ernst/John Guttag Spring 2001 Slide 4

Abstraction by Specification

Roughly speaking, separate what from how
Why also important, but something different

Describes required behavior
Not means of achieving it
Often not even exact behavior
E.g., compute square root of x within ϵ

Specification denotes a (usually infinite) set of programs
An abstraction of each of its members

E.g., set of all procedures that sort lists
Abstracts from what kinds of lists
Abstracts from properties of sort, e.g., stable or not
Abstracts from algorithm, e.g., merge sort

© Michael Ernst/John Guttag Spring 2001 Slide 5

An Example

```
int squareRoot(int in)
  requires in is a perfect square
  effects returns a square root of in
```

Admits a variety of algorithms for computing square root
Probably a good thing

Admits a number of different answers
Always positive square root
Always negative square root
Either negative or positive square root

Is this a good thing ?
Maybe
Maybe not

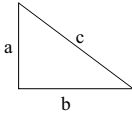
© Michael Ernst/John Guttag Spring 2001 Slide 6

Danger: Relevance Depends Upon Uses

Less specific specifications
 Lead to better implementations
 More flexibility for later specialization
 Inheritance later in term

More specific specifications
 Lead to wider direct usability

$c = \sqrt{a^2 + b^2}$



© Michael Ernst/John Guttag Spring 2001 Slide 7

Specifications and Implementations

Try to write specification first
 Often amend while implementing
 Easier to implement (e.g., constraint added)
 More useful (e.g., constraint not needed)

Usually one implementation/specification
 Important that one not confuse the two
 Specification denotes set of potential implementations

Users should depend only on properties
 Guaranteed by specification

Provides
 Locality of using programs
 Modifiability for implementer of abstraction

© Michael Ernst/John Guttag Spring 2001 Slide 8

Using Specification Ensures Modifiability

Replace implementation of the abstraction
 Need not change using programs

Might have to recompile
 But never reprogram

Why change implementation of abstraction?
 Fix bug (i.e., failure to correspond to specification)
 Using programs should not depend upon bug!
 Port to new platform
 Improve efficiency

“I could really speed up foo, but the last time I changed foo, bar stopped working -- and I don't understand bar worth a ...”

© Michael Ernst/John Guttag Spring 2001 Slide 9

An (Abstract) Example

Specification: reference manual for Java
Implementation: compiler and runtime environment

Locality
 Write programs without understanding the compiler
 Write compiler without understanding programs that will use it

Modifiability
 Programs port across compilers
 Compiler writers free to add optimizations

© Michael Ernst/John Guttag Spring 2001 Slide 10

Specifying Procedures

Procedure - code written in a formal executable language
 Perhaps, but not necessarily, Java
 Some library routines may be hand coded in assembler

Specification - written in a specification language
 Usually not formal (or executable)
 Most often highly stylized natural language

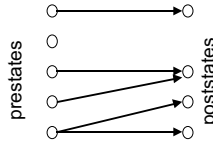
Each of these is tangible text
Each denotes something less tangible
Can think of each as denoting a relation
 A set (perhaps infinite) of ordered pairs
 Described using a characteristic function
 Mapping from <inputs, outputs> → Boolean

© Michael Ernst/John Guttag Spring 2001 Slide 11

What Specifications Denote

Think of specification as denoting a transition relation
 Spec: <State, Arguments>, <State, Results> → Boolean
prestates *poststates*

Why a relation?
 Undesirable to constrain abstraction to a single mapping
 E.g., choose an element from a set and delete it



© Michael Ernst/John Guttag Spring 2001 Slide 12

General Form of a Specification of a Procedure

<qualifiers> <return> <name> <formals>
<requires>
<modifies>
<effects>

```
public static Boolean isPrime (int i)
  requires i > 0
  modifies nothing
  effects returns true if i is a prime number and false
  otherwise
```

requires \Rightarrow (modifies \wedge effects)
Recall that $\forall X, X \Rightarrow \text{true}, \text{false} \Rightarrow X$
 $x \Rightarrow y$ is the same as $y \vee \text{not}(x)$

© Michael Ernst/John Guttag Spring 2001 Slide 13

An Example

```
int find(int[] a, int value)
  requires value occurs in a
  modifies nothing
  effects returns an i such that a[i] = value
```

requires (the precondition)
Client must ensure prestate satisfies the precondition

Modifies - No side effects allowed here

effects
Defines possible poststates using values from the prestate
Implementer must guarantee the postcondition
If precondition holds
Suppose precondition does not hold?

© Michael Ernst/John Guttag Spring 2001 Slide 14

requires \Rightarrow (modifies \wedge effects)


When is specification S1 weaker than S2?
When $\forall P, (P \text{ satisfies } S2) \Rightarrow (P \text{ satisfies } S1)$

We can weaken a specification by
Making requires harder to satisfy
Adding things to modifies clause
Making effects easier to satisfy

What is the strongest (most constraining) requires clause?
true

What is the strongest (most constraining) effects clause?
false

What is the strongest (most constraining) modifies clause?
modifies nothing



© Michael Ernst/John Guttag Spring 2001 Slide 15