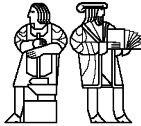


6.170 Lecture 6 Procedure abstractions



Michael Ernst
MIT EECS

Outline

Procedure specifications taxonomized; desugaring
Satisfying a specification; substitutability
Specification style; checking preconditions

Michael Ernst/John Guttag

Spring 2001

Slide 2

Procedure specifications

Liskov style
requires
modifies
effects

6.170 style:
requires
modifies
throws
effects
returns

6.170 style can be desugared into Liskov style
Liskov style can be desugared into a single logical clause

Michael Ernst/John Guttag

Spring 2001

Slide 3

Desugaring 6.170 style into Liskov style

6.170 style:
requires (unchanged)
modifies (unchanged)
throws
effects
returns
} correspond to Liskov "effects"

Example (from java.util.Vector):

```
// throws: IndexOutOfBoundsException if index < 0 || index >= size()
// effects: thispost[index] = element
// returns: thispre[index]
Object set(int index, Object element);
```

This ought to also specify that other elements are not changed

Desugaring:

```
// effects: if index < 0 || index >= size() throws IndexOutOfBoundsException
// else thispost[index] = element && returns thispre[index]
Object set(int index, Object element);
```

Michael Ernst/John Guttag

Spring 2001

Slide 4

Desugaring Liskov style

Liskov style
requires R
modifies f (suppose that fields g and h are not listed)
effects E

Single logical formula

$R \Rightarrow g_{post} = g_{pre} \ \&\& \ h_{post} = h_{pre} \ \&\& \ E$

Implementation must guarantee this formula is true

If precondition is false, formula is vacuously true, because
(false \Rightarrow X) = true

Michael Ernst/John Guttag

Spring 2001

Slide 5

Review: Transition relations

Transition relation maps procedure arguments to results

```
int increment(int i) {
    return i+1;
}
```

```
int mySqrt(double a) {
    if (Random.nextBoolean())
        return Math.sqrt(a);
    else
        return - Math.sqrt(a);
}
```

Specifications have transition relations, too

Michael Ernst/John Guttag

Spring 2001

Slide 6

Review: strength of a specification

- A stronger specification has a smaller transition relation
- A stronger specification is satisfied by fewer procedures
- A stronger specification has
 - weaker preconditions
 - stronger postcondition
 - fewer modifications

Michael Ernst/John Guttag Spring 2001 Slide 7

Substitutability

A stronger specification can always be substituted for a weaker one

This is a key concept to which we will return later

Michael Ernst/John Guttag Spring 2001 Slide 8

Satisfaction of a Specification

Let P be an implementation and S a specification

P satisfies S iff
Relation denoted by P is a subset of relation denoted by S
(A white lie)

The statement “ P is correct” is meaningless
Though often made!

If P does not satisfy S , either (or both!) could be “wrong”
“One person’s feature is another person’s bug.”
Usually better to change the program
Hate to publish amendments to library specifications

Michael Ernst/John Guttag Spring 2001 Slide 9

A Trivial Example

Specification
int anyInt (int x)
requires $x = 0$
effects returns any int

Code
int anyInt(int x) {
 return 3;
}

Does this program satisfy the specification?

Not according to the definition I just supplied
 $S = \{<0, 0>, <0, 1>, <0, -1>, \dots, <0, 3>, \dots\}$
 $P = \{<0, 3>, <1, 3>, <-1, 3>, \dots\}$

Michael Ernst/John Guttag Spring 2001 Slide 10

Satisfaction Revisited

Let $P \mid D = P$ restricted to the domain D
Remove from P all pairs whose first member is not in D

P satisfies S iff
 $P \mid (\text{Domain of } S)$ is a subset of relation denoted by S

Still a white lie.

Michael Ernst/John Guttag Spring 2001 Slide 11

Specification style

Typically have only one of effects and returns
A procedure has a side effect or is called for its value
Exception: return old value, as for **HashMap.put**

The point of a specification is to be helpful
Formalism helps, overformalism doesn't

A specification should be
coherent (not too many cases)
informative (bad example: **HashMap.get**)
strong enough (to do something useful, to make guarantees)
weak enough (to permit (efficient) implementation)

Michael Ernst/John Guttag Spring 2001 Slide 12



Checking preconditions

Checking preconditions makes an implementation more robust, provides better feedback to the client, avoids silent errors

A quality implementation checks preconditions whenever it is inexpensive and convenient to do so