Software Engineering - Exercise Sheets 4, 5

Martin Mehlmann
(mehlmann@informatik.uni-freiburg.de)

May 27, 2009

Exercise Sheet 4

Sheet 4 - Exercise 1

```
/* A stack with a fixed maximum capacity */
public class Stack<X>
{

    int topIx;         // index in content of the top element
    final X[] content; // array that stores elements of the stack

    @Inv{topIx < content.length ∧ isEmpty()?topIx = −1 : topIx ≥ 0}

    @Pre{capacity > 0}
    public Stack(int capacity)
    {
        this.content = (X[]) new Object[capacity];
        this.topIx = -1;
    }
    @Post{isEmpty()∧!isFull()}
```

Sheet 4 - Exercise 1

```
@Pre{!isEmpty()}
public X top()
{
    return this.content[this.topIx];
}

@Pre{!isEmpty()}
public X pop()
{
    X res = this.content[this.topIx];
    this.topIx--;
    return res;
}
@Post{!isFull() ∧ old.top() = pop}
```

Sheet 4 - Exercise 1

```
@Pre{!isFull()}
public void push(X x)
{
    this.topIx++;
    this.content[this.topIx] = x;
}
@Post{!isEmpty() ∧ top() == x}

public boolean isEmpty()
{
    return this.topIx == -1;
}
public boolean isFull()
{
    return (this.topIx ==
            (this.content.length - 1));
}
public static void main(String[] args) { ... }
```

Sheet 4 - Exercise 2 (a)

```
interface Map<K,V>
{
    @Pre{key ≠ null}
    boolean containsKey(K key);

    @Pre{key ≠ null}
    V get(K key);
    @Post{containsKey(key) ∨ get = null}

    @Pre{key ≠ null}
    void put(K key, V value);
    @Post{containsKey(key) ∧ get(key) = value}
}
```

Sheet 4 - Exercise 2 (b)

Correct specialization, `Map` now supports `null` keys.

```
interface MapWithNull<K,V> extends Map<K,V>
{
    boolean containsKey(K key);

    V get(K key);
    @Post{containsKey(key) ∨ get = null}

    void put(K key, V value);
    @Post{containsKey(key) ∧ get(key) = value}
}
```

Sheet 4 - Exercise 2 (b)

Incorrect specialization, `Map` disallows `null` values.

```
interface MapNoNullValues<K,V> extends Map<K,V>
{
    @Pre{key ≠ null}
    boolean containsKey(K key);

    @Pre{key ≠ null}
    V get(K key);
    @Post{get ≠ null ∨ !containsKey(key)}

    @Pre{key ≠ null ∧ value ≠ null}
    void put(K key, V value);
    @Post{containsKey(key) ∧ get(key) = value}
}
```

Precondition of `put` is stronger than the corresponding precondition in `Map`.

Exercise Sheet 5

Sheet 5 - Exercise 1

(i) $\{\mathit{true}\}$ `x := 0;` $\{\mathit{false}\}$

The triple is not partially correct cause it cannot be derived in the hoare calculus

(ii) $\{\mathit{false}\}$ `x := 0;` $\{\mathit{true}\}$

The triple is partially correct:

$\{\mathit{false}\}$
$\Longrightarrow$
$\{\mathit{true}\}$
`x:= 0`
$\{\mathit{true}\}$

Sheet 5 - Exercise 1

(iii) $\{x \geq y\}$ `y := y + 1;` $\{x = y - 1\}$

The triple is not partially correct cause it cannot be derived in the hoare calculus

(iv) $\{x = y\}$ `y := y + 1;` $\{x \geq y - 1\}$

The triple is partially correct:

$\{x = y\}$
$\Longrightarrow$
$\{x \geq y\}$
`y:= y + 1`
$\{x \geq y - 1\}$

Sheet 5 - Exercise 1

(v) $\{a = x, b = y\}$

```
a := a + b;
b := a - b;
a := a - b;
```

$\{a = y, b = x\}$

The triple is partially correct:

$\{a = x, b = y\}$

$\Longrightarrow$

$\{(a + b) - (a + b - b) = y, a + b - b = x\}$

```
a := a + b;
```

$\{a - (a - b) = y, a - b = x\}$

```
b := a - b;
```

$\{a - b = y, b = x\}$

```
a := a - b;
```

$\{a = y, b = x\}$

Sheet 5 - Exercise 1

(vi) $\{true\}$

```
int x;
if (x % 2 == 0)
  h := x / 2;
else
  h := (x - 1) / 2;
```

$\{2 * h \leq x \leq 2 * h + 1\}$

Sheet 5 - Exercise 1

The triple is partially correct:

$\{true\}$

```
if (x % 2 = 0)
```

$\{x\%2 = 0\}$

$\Longrightarrow$

$\{2 * (x/2) = x\}$

$\Longrightarrow$

$\{2 * (x/2) \leq x \leq 2 * (x/2) + 1\}$

```
h := x/2
```

$\{2 * h \leq x \leq 2 * h + 1\}$

```
else
```

$\{x\%2 \neq 0\}$

$\Longrightarrow$

$\{2 * ((x - 1)/2) = x - 1\}$

$\Longrightarrow$

$\{2 * ((x - 1)/2) \leq x \leq 2 * ((x - 1)/2) + 1\}$

```
h:= (x - 1)/2
```

$\{2 * h \leq x \leq 2 * h + 1\}$

$\{2 * h \leq x \leq 2 * h + 1\}$

Sheet 5 - Exercise 1

State a program S with a single variable x such that $\{y = 5\}\ S\ \{y = 23\}$ is partially correct. The hoare triple is partially correct for the program

```
while(true) x := x + 1;
```

$\{y = 5\}$
```
while (true)
```
  $\{y = 5, \mathit{true}\}$
```
  x := x + 1;
```
  $\{y = 5\}$
$\{y = 5, \mathit{false}\}$
$\Longrightarrow$
$\{y = 23\}$

Total correctness does not hold, cause the program does not terminate.

Sheet 5 - Exercise 2

Which of the following assertions are invariants for the `while` loop of the program? Give a proof.

(i) *true true* is an invariant:

$\{true\}$
while (a < x)
  $\{true, a < x\}$
  $\Longrightarrow$
  $\{true\}$
  a := a + 1;
  $\{true\}$
  b := b + a;
  $\{true\}$
$\{true, a \geq x\}$

Sheet 5 - Exercise 2

(ii) *false false* is an invariant:

$\{false\}$

```
while (a < x)
```

$\{false, a < x\}$

$\Longrightarrow$

$\{false\}$

```
a := a + 1;
```

$\{false\}$

```
b := b + a;
```

$\{false\}$

$\{false, a \geq x\}$

Sheet 5 - Exercise 2

(iii) $x \geq a \wedge a \geq a_0$

$x \geq a \wedge a \geq a_0$ is an invariant:

$\{x \geq a, a \geq a_0\}$

while `(a < x)`

$\{x > a, a \geq a_0, x \geq a\}$

$\Longrightarrow$

$\{x > a, a \geq a_0\}$

$\Longrightarrow$

$\{x \geq a + 1, a + 1 \geq a_0\}$

`a := a + 1;`

$\{x \geq a, a \geq a_0\}$

`b := b + a;`

$\{x \geq a, a \geq a_0\}$

$\{x \geq a, a \geq a_0, x \leq a\}$

Sheet 5 - Exercise 2

(iv) $b = a(a+1)/2$
$b = a(a+1)/2$ is an invariant:

$\{b = a * (a+1)/2\}$
```
while (a < x)
```
$\{b = a * (a+1)/2, a < x\}$
$\Longrightarrow$
$\{b + a + 1 = a * (a+1)/2 + a + 1, a < x\}$
$\Longrightarrow$
$\{b + a + 1 = (a * a + 3a + 2)/2, a < x\}$
$\Longrightarrow$
$\{b + a + 1 = (a+1) * (a+2)/2\}$
```
a := a + 1;
```
$\{b + a = a * (a+1)/2\}$
```
b := b + a;
```
$\{b = a * (a+1)/2\}$
$\{b = a * (a+1)/2, a >= x\}$