Formal Methods for Java

Lecture 17: Framing in the Key Prover

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

June 28, 2017

Motivating Example

```
public class Cell {
    private int value;

    public Cell() {}

    public void get() {
        return value;
    }

    public void set(int v) {
        this.value = v;
    }
}
```

```
public class Caller {
    //@ ensures \result = 5;
    public int m() {
        Cell c1 = new Cell();
        c1.set(5);
        Cell c2 = new Cell();
        c2.set(10);
        return c1.get();
    }
}
```

Find pre-/post-conditions, invariants for `Cell` sufficient to show the post-condition of `m()`.

What are possible problems?

- Does `c1.get()` return the value set by `c1.set()`?
- Does creating a new cell change the value of `c1.get()`?
- Does calling `c2.set()` change the value of `c1.get()`?
- Does calling `c2.set()` affect the invariant of `c1`?

Does get() return the value set by set()?

If `get()` is pure (`assignable \nothing`), we can use `get()` in the post-condition of `set()`.

```
public class Cell {
    private int value;

    /*@ assignable \nothing;
      @ ensures \result == get(); @*/
    public void get() {
        return value;
    }

    /*@ ensures get() == v; @*/
    public void set(int v) {
        this.value = v;
    }
}
```

Does new Cell() change the value?

Say that `new Cell()` doesn't change anything.

```
public class Cell {
    private int value;

    /*@ assignable \nothing; @*/
    Cell() { }

    /*@ assignable \nothing;
      @ ensures \result == get(); @*/
    public void get() {
        return value;
    }

    /*@ ensures get() == v; @*/
    public void set(int v) {
        this.value = v;
    }
}
```

Does c2.set() change the value?

Say what *c2.set()* changes.
But also say that *c1.get()* doesn't depend on *c2.value*.

```
public class Cell {
    private int value;

    /*@ normal_behavior
      @ assignable \nothing; @*/
    Cell() { }

    /*@ normal_behavior
      @ accessible this.value;
      @ assignable \nothing;
      @ ensures \result == get(); */
    public void get() {
        return value;
    }

    /*@ normal_behavior
      @ assignable this.value;
      @ ensures get() == v; @*/
    public void set(int v) {
        this.value = v;
    }
}
```

But value should not be visible

Public pre- and post-condition should not reveal internals.

Use a public model variable instead.

```
public class Cell {
    private int value;
    //@ public \locset footprint;
    //@ accessible footprint: footpring;
    //@ represents footprint = value;

    /*@ normal_behavior
      @ assignable \nothing; @*/
    Cell() { }

    /*@ normal_behavior
      @ accessible footprint;
      @ assignable \nothing;
      @ ensures \result == get(); */
    public void get() {
        return value;
    }

    /*@ normal_behavior
      @ assignable footprint;
      @ ensures get() == v; @*/
    public void set(int v) {
        this.value = v;
    }
}
```

locset footprint

```
//@ public \locset footprint;
//@ accessible footprint: footprint;
//@ represents footprint = value;
```

- `\locset` is a set of locations (fields of objects).
- Can be used in `assignable` and `accessible`.

What does `accessible footprint : footprint` mean?

$\Rightarrow$ The locations in *footprint* only change if their values change.

- For example, in function declared `assignable footprint`.

Footprints should be disjoint

```
c1.set(10);     // assignable c1.footprint;
// ensures c1.get() == 10;
c2.set(10);     // assignable c2.footprint;
return c2.get(); // accessible c1.footprint;
```

How do we know that $c1.footprint$ and $c2.footprint$ do not intersect?

```
//@ ensures \fresh(footprint);
public Cell()

//@ ensures footprint == old.footprint;
public void set(int x)
```

Alternatively, if `set()` changes the footprint:

```
//@ ensures \new_elems_fresh(footprint);
public void set(int x)
```

Footprints as Alternative to Datagroups

```
public interface PriorityQueue {
  //@ public instance model \locvar footprint;

  /*@ public normal_behavior
    @  assignable footprint;
    @  ensures \new_elems_fresh(footprint);
    @*/
  public void enqueue(Comparable o);
  ...
}
```

Implementation then uses represents:

```
public class Heap implements PriorityQueue {
  public Comparable[] data;
  //@ represents footprint = data, data.*
  ...
}
```

List Interface

```
public interface List {
    //@ public model instance \locset footprint;
    //@ public accessible footprint: footprint;

    /*@ public normal_behaviour
      @   accessible footprint;
      @   ensures size() >= 0;
      @*/
    public /*@pure@*/ int size();


    /*@ public normal_behaviour
      @   requires 0 <= index && index < size();
      @   accessible footprint;
      @   ensures \result == get();
      @
      @ also public exceptional_behaviour
      @   requires index < 0 || size() <= index;
      @   signals_only IndexOutOfBoundsException;
      @*/
    public /*@pure@*/ Object get(int index);
    ...
}
```

List Interface (add)

```
public interface List {
    ...
  /*@ public normal_behaviour
    @   assignable footprint;
    @   ensures size() == \old(size()) + 1 && get(size() - 1) == o;
    @   ensures (\forall int i; 0 <= i && i < size() - 1; get(i) == \old(g
    @   ensures \new_elems_fresh(footprint);
    @*/
  public void add(Object o);
}
```