Formal Methods for Java

Lecture 18: Verification of a Linked List in Jahob

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

December 21, 2012

Example: Doubly Linked List

Consider an implementation of a cyclic list with prev and next pointer:

```
class Node {
    public Node next;
    public Node prev;
    public Object data;
}

class DoublyLinkedList
{
    private Node first;
    private Node last;
```

Defining Interfaces using Specification Variables

```
class DoublyLinkedListSet {
    private Node first;
    private Node last;
    /*: public specvar nodes :: objset;
        public specvar content :: objset;
    ...
```

How can we define the set of nodes and data values in the linked list?

```
content == first.next*.data
```

Jahob supports reflexive transitive closure but with a different syntax:

Definition (rtrancl_pt)

Let $R : \alpha \Rightarrow \alpha \Rightarrow$ bool be a relation on some type α, then rtrancl_pt R is the reflexive transitive closure of R:
rtrancl_pt R x y holds if there is a sequence $x = x_0, \ldots, x_n = y$, $n \geq 0$ such that $R\ x_i\ x_{i+1}$ holds for $0 \leq i < n$.

Using the rtrancl_pt predicate

Definition (rtrancl_pt)

Let $R : \alpha \Rightarrow \alpha \Rightarrow$ bool be a relation on some type α, then rtrancl_pt R is the reflexive transitive closure of R:
rtrancl_pt R x y holds if there is a sequence $x = x_0, \ldots, x_n = y$, $n \geq 0$ such that $R\ x_i\ x_{i+1}$ holds for $0 \leq i < n$.

Define the successor relation using the field *Node.next*:

```
R == (% x y. x..Node.next = y)
```

Note: % is λ-abstraction.

The set of all nodes on the list is:

```
nodes == {n. rtrancl_pt (% x y. x..Node.next = y) first n}
```

and the set of all values on the list is:

```
content == {d. EX n. n: nodes & n..Node.data = d }
```

Alternative Syntax Using Sets of Tuples

Definition (rtrancl_pt)

Let $R : \alpha * \alpha$ set be a relation on some type α (as set of tuples), then R^* is the reflexive transitive closure of R: $(x, y) \in R^*$ holds if there is a sequence $x = x_0, \ldots, x_n = y$, $n \geq 0$ such that $(x_i, x_{i+1}) \in R$ holds for $0 \leq i < n$.

Define the successor relation using the field `Node.next`:

```
R == { (x,y) . x..Node.next = y}
```

The set of all nodes on the list is:

```
nodes == {n. (first, n) : {(x,y). x..Node.next = y}^*}
```

and the set of all values on the list is:

```
content == {d. EX n. n: nodes & n..Node.data = d }
```

Cyclic Versus Null-Terminated Lists

The decision procedure in Jahob works best with null-terminated lists.
Introduce a second linking structure on top of the existing list as ghost variables:

```
class Node {
    public Node next;
    public Node prev;
    public Object data;

}

class DoublyLinkedList
{
    private Node first;
    private Node last;
    /*:
      specvar nodes :: objset;
      vardefs "nodes == {x. x ~= null &
              (first,x) : {(v,w). v..next =w}^*}";
class Node {
    public Node next;
    public Node prev;
    public Object data;
    //: public ghost specvar next1 :: obj = "null";
```

Relation of the Linking Structures

We introduce two axioms to relate *next*, *prev* with the new field *next1*:

```
class DoublyLinkedList
{
  ...
   /*:
    invariant nextDef: "ALL x y. x..next = y -->
        ((x = last --> y = first) &
         (x : nodes & x ~= last --> y = x..next1))"

    invariant prevDef: "ALL x y. x..prev = y -->
        ((x = first & first ~= null --> y = last) &
         (x : nodes & x ~= first --> y..next1 = x))"
    */
```