Formal Methods for Java
Lecture 18: Verification of a Linked List in Jahob

Jochen Hoenicke
Software Engineering
Albert-Ludwigs-University Freiburg

December 21, 2012
Consider an implementation of a cyclic list with prev and next pointer:

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

class DoublyLinkedList {
    private Node first;
    private Node last;
}
```
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 $R : \alpha \Rightarrow \alpha \Rightarrow \text{bool}$ be a relation on some type $\alpha$, then \texttt{rtrancl_pt} $R$ is the reflexive transitive closure of $R$:
\texttt{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 $R : \alpha \Rightarrow \alpha \Rightarrow \text{bool}$ be a relation on some type $\alpha$, then rtrancl\_pt $R$ is the reflexive transitive closure of $R$: $\text{rtrancl\_pt} R x y$ 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 $\text{Node}.\text{next}$:

$$R = (\% x y. \ x..\text{Node}.\text{next} = y)$$  

Note: $\%$ is $\lambda$-abstraction.

The set of all nodes on the list is:

$$\text{nodes} = \{ n. \ \text{rtrancl\_pt} (\% x y. \ x..\text{Node}.\text{next} = y) \ \text{first} \ n \}$$

and the set of all values on the list is:

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

Define the successor relation using the field \( \text{Node.next} \):
\[
R = \{ (x, y) . \ x..\text{Node.next} = y \}
\]

The set of all nodes on the list is:
\[
nodes = \{n. (\text{first}, n) : \{(x,y). \ x..\text{Node.next} = y\}^*\}
\]

and the set of all values on the list is:
\[
content = \{d. \ \text{EX} \ n. \ n: \nodes \ & \ n..\text{Node.data} = d \}
\]
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:

```java
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..next1 =w}^*}";
    
    class Node {
        public Node next;
        public Node prev;
        public Object data;
        //: public ghost specvar next1 :: obj = "null";
    }
```
We introduce two axioms to relate next, prev with the new field next1:

```java
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))"
    */
```