Interactive Software Verification SS 2013

https://www21.in.tum.de/teaching/isv/SS13

H. Gast, L. Noschinski

SHEET 8

List Manipulations

(Page 1 / 2)

Date: 18.12.2012
Hand-In: 25.12.2012, 23:50

Goals Manipulate lists on the heap and use ghost variables.

Exercise 1 [3] Prepend a List Node

The cons operation (insert a new element at the beginning of a list) is easy to implement on the heap. One just needs to set the `next`-pointer of the new node to the begin of the existing list. Moreover, the old list stays unchanged, which makes this operation so useful for functional languages.

```
pre: ”list node-alloc node-next p XS NULL ∧ node-alloc q”
post: ”list node-alloc node-next q (q # XS) NULL ∧
      list node-alloc node-next p XS NULL”
{*
   q→next = p;
*}
```

Unfortunately, the given pre-condition is to weak. Insert the necessary additional assumptions and prove the correctness. (Have a look at the “Cheat Sheet” from the lecture).

Exercise 2 [4] Test for an Element

The following program tests whether the value *v* is contained in the list *p*. Prove the program correct. For this, annotate the program with ghost variables, so you do not have to provide witnesses for existential quantifiers manually.

```
pre: ”list node-alloc node-next p XS NULL ∧ DS = list-data node-data XS”
post: ”r ≠ 0 ↔ v ∈ set DS”
{*
    r = 0;
    /*@ True */
    while (p != null && r == 0) {
        if (p→data == v) {
            r = 1;
        }
    p = p→next;
    }
*}
```

Exercise 3 [5] Insert an Element (Hard)

Verify the following code which inserts a node q at the position pos of the list p and returns the modified list in r.

```
if (pos == 0) {
  q->next = p;
  r = q;
} else {
  r = p;
  prev = p;
  p = p-> next;
  pos = pos - 1;
  while (0 < pos) {
    prev = p;
    p = p-> next;
    pos = pos - 1;
  }
  prev->next = q;
  q->next = p;
}
```