Interactive Software Verification SS 2013

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

H. Gast, L. Noschinski

SHEET 7

Lists and the Heap

Date: 19.06.2013
Hand-In: 26.06.2013, 08:30

Goals You get a deeper knowledge of Burstall's memory model.

Exercise 1 [4] Heap Basics

1. What is the meaning of p, q, and xs in the abstraction predicate list node-alloc node-next p xs q for lists?
2. What is the purpose of the parameters node-alloc and node-next?
3. For every abstraction predicate, we need a separation lemma. What is its purpose? (Look at examples and then give a general explanation)
4. Why can Burstall's memory model not be used for arbitrary C programs? Give an example for a program, which is not correctly modeled.

Exercise 2 [6] Sum of List Elements

(a) [5] The following code iterates through a list at `p` and computes the sum of all elements

```
s = 0;
while (p != null) {
    s = s + p->data;
    p = p->next;
}
```

The precondition requires that the list is allocated and binds the auxiliary variable P.

list node-alloc node-next p XS NULL $\wedge$ p = P

The postcondition ensures that the result s is really the sum of the data fields of the list elements.

s = listsum (list-data node-data XS)

The invariant is similar to `list_length` from the lecture, but we need to consider the part of the list we already visited, too.

$\exists$ ys zs.
  list node-alloc node-next P ys p $\wedge$
  list node-alloc node-next p zs NULL $\wedge$
  XS = ys @ zs $\wedge$
  s = listsum (list-data node-data ys)

Your task is to do a detailed proof to understand which lemmas are required at which point.

(b) [1] Try to find a proof which is as automatic as possible. For this, extend the current simpset step-by-step by using

declare $\cdots$ [simp]

instead of passing the theorems to every `simp` call.