Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Dr. Peter Lammich

SS 2018
18. 5. 2018

Functional Data Structures

Exercise Sheet 6

Exercise 6.1 Complexity of Naive Reverse

Show that the naive reverse function needs quadratically many *Cons* operations in the length of the input list. (Note that $[x]$ is syntax sugar for *Cons* x []!)

thm *append.simps*

```
fun reverse where
  "reverse [] = []"
| "reverse (x#xs) = reverse xs @ [x]"
```

Exercise 6.2 Simple Paths

Recall the definition of paths from last exercise sheet:

```
fun path :: "('a ⇒ 'a ⇒ bool) ⇒ 'a ⇒ 'a list ⇒ 'a ⇒ bool"
  where
  "path G u [] v ⟷ u=v"
| "path G u (x#xs) v ⟷ G u x ∧ path G x xs v"

lemma path_append[simp]: "path G u (p1@p2) v ⟷ (∃w. path G u p1 w ∧ path G w p2 v)"
  by (induction p1 arbitrary: u) auto
```

A simple path is a path without loops, or, in other words, a path where no node occurs twice. (Note that the first node of the path is not included, such that there may be a simple path from u to u.)

Show that for every path, there is a corresponding simple path.

Hint: Induction on the length of the path

thm *measure_induct_rule*[**where** *f=length*, *case_names shorter*]

```
lemma exists_simple_path:
  assumes "path G u p v"
  shows "∃p'. path G u p' v ∧ distinct p'"
```

Homework 6.1 Stability of Insertion Sort

Submission until Friday, May 25, 11:59am. Have a look at Isabelle's standard implementation of sorting: *sort_key*. (Use Ctrl-Click to jump to the definition in `~~/src/HOL/List.thy`) Show that this function is a stable sorting algorithm, i.e., the order of elements with the same key is not changed during sorting!

lemma "$[x{\leftarrow}sort\_key\ k\ xs.\ k\ x = a] = [x{\leftarrow}xs.\ k\ x = a]$"
term "$[x{\leftarrow}xs.\ P\ x]$"

Note: $\langle[x{\leftarrow}xs.\ P\ x]\ \rangle$ is syntax sugar for $\langle filter\ P\ xs\rangle$, where the filter function returns only the elements of list xs for which $P\ xs = True$.

Hint: You do not necessarily need Isar, and the auxiliary lemmas you need are already in Isabelle's library. **find_theorems** is your friend!

Homework 6.2 Quickselect

Submission until Friday, May 25, 11:59am.

From https://en.wikipedia.org/wiki/Quickselect:

Quickselect is a selection algorithm to find the kth smallest element in an unordered list. It is related to the quicksort sorting algorithm. Like quicksort, it was developed by Tony Hoare, and thus is also known as Hoare's selection algorithm. Like quicksort, it is efficient in practice and has good average-case performance, but has poor worst-case performance. Quickselect and its variants are the selection algorithms most often used in efficient real-world implementations.

Quickselect uses the same overall approach as quicksort, choosing one element as a pivot and partitioning the data in two based on the pivot, accordingly as less than or greater than the pivot. However, instead of recursing into both sides, as in quicksort, quickselect only recurses into one side — the side with the element it is searching for.

Your task is to prove correct the quickselect algorithm, which can be implemented in Isabelle as follows:

```
fun quickselect :: "'a::linorder list ⇒ nat ⇒ 'a" where
  "quickselect (x#xs) k = (let
    xs1 = [y←xs. y<x];
    xs2 = [y←xs. ¬(y<x)]
  in
    if k<length xs1 then quickselect xs1 k
    else if k=length xs1 then x
    else quickselect xs2 (k−length xs1−1)
  )"
| "quickselect [] _ = undefined"
```

Your first task is to prove the crucial idea of quicksort, i.e., that partitioning wrt. a pivot element p is correct.

lemma *partition_correct*: *"sort xs = sort [x←xs. x<p] @ sort [x←xs. ¬(x<p)]"*

Hint: Induction, and auxiliary lemmas to transform a term of the form *insort x (xs @ ys)* when you know that *x* is greater than all elements in *xs* / less than or equal all elements in *ys*.

Next, show that quickselect is correct

lemma *"k<length xs ⟹ quickselect xs k = sort xs ! k"*

Proceed by computation induction, and a case distinction according to the cases in the body of the quickselect function

proof (*induction xs k rule*: *quickselect.induct*)
  case (*1 x xs k*)

Note: To make the induction hypothesis more readable, you can collapse the first two premises of the form *?x=...* by reflexivity:

  note *IH = "1.IH"[OF refl refl]*

Insert your proof here!

next
  case *2* **then show** *?case* **by** *simp*
qed