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

SS 2017
2. 6. 2017

Functional Data Structures

Exercise Sheet 6

Exercise 6.1 Selection Sort

Selection sort (also known as MinSort) sorts a list by repeatedly moving the smallest element of the remaining list to the front.

Define a function that takes a non-empty list, and returns the minimum element and the list with the minimum element removed

fun *find_min* :: "$'a$::*linorder list* $\Rightarrow$ $'a \times {'a}$ *list*"

Show that *find_min* returns the minimum element

lemma *find_min_min*:
  assumes "*find_min xs* = (*y*,*ys*)"
  assumes "*xs*$\neq$[]"
  shows "*a*$\in$*set xs* $\Longrightarrow$ *y* $\leq$ *a*"

Show that *find_min* returns exactly the elements from the list

lemma *find_min_mset*:
  assumes "*find_min xs* = (*y*,*ys*)"
  assumes "*xs*$\neq$[]"
  shows "*mset xs* = *add_mset y* (*mset ys*)"

Show the following lemma on the length of the returned list, and register it as [*dest*]. The function package will require this to show termination of the selection sort function

lemma *find_min_snd_len_decr*[*dest*]:
  assumes "(*y*,*ys*) = *find_min* (*x*#*xs*)"
  shows "*length ys* < *length* (*x*#*xs*)"

Selection sort can now be written as follows:

fun *sel_sort* **where**
  "*sel_sort* [] = []"
| "*sel_sort xs* = (*let* (*y*,*ys*) = *find_min xs in y*#*sel_sort ys*)"

Show that selection sort is a sorting algorithm:

lemma *sel_sort_mset*[*simp*]: "*mset* (*sel_sort xs*) = *mset xs*"
lemma "*sorted* (*sel_sort xs*)"

Define cost functions for the number of comparisons of *find_min* and *sel_sort*.

fun *c_find_min* :: *"'a list ⇒ nat"*

fun *c_sel_sort*

Try to find a closed formula for *c_sel_sort*! If you do not succeed, try to find a good estimate. (Hint: Should be $O(n^2)$)

To find a closed formula: On paper:

- Put up a recurrence equation (depending only on the length of the list)
- Solve the equation (Assume that the solution is an order-2 polynomial) In Isabelle:
- Insert the solution into the lemma below, and try to prove it

lemma *c_sel_sort_cmpx*: *"c_sel_sort xs = undefined"* **oops**

Homework 6 Quicksort

Submission until Friday, 2. 6. 2017, 11:59am. We extend the notion of a sorting algorithm, by providing a key function that maps the actual list elements to a linearly ordered type. The elements shall be sorted according to their keys.

fun *sorted_key* :: *"('a ⇒ 'b::linorder) ⇒ 'a list ⇒ bool"* **where**
  "sorted_key k [] = True"
| *"sorted_key k (x # xs) = ((∀ y∈set xs. k x ≤ k y) & sorted_key k xs)"*

Quicksort can be defined as follows: (Note that we use *nat* for the keys, as this causes less trouble when writing Isar proofs than a generic *'b::linorder*)

fun *qsort* :: *"('a ⇒ nat) ⇒ 'a list ⇒ 'a list"* **where**
  "qsort k [] = []"
| *"qsort k (p#xs) = qsort k [x←xs. k x<k p]@p#qsort k [x←xs. ¬k x<k p]"*

The syntax *filter P xs* is a shortcut notation for *filter P xs.*

Show that quicksort is a sorting algorithm:

lemma *qsort_preserves_mset*: *"mset (qsort k xs) = mset xs"*
lemma *qsort_sorts*: *"sorted_key k (qsort k xs)"*

The following is a cost function for the comparsions of quicksort:

fun *c_qsort* :: *"('a ⇒ nat) ⇒ 'a list ⇒ nat"* **where**
  "c_qsort k [] = 0"
| *"c_qsort k (p#xs)*
    = c_qsort k [x←xs. k x<k p] + c_qsort k [x←xs. k x≥k p] + 2∗length xs"

Show that the number of required comparisons is at most $(\textit{length xs})^2$.

Hints:

- Do an induction on the length of the list, and, afterwards, a case distinction on the list constructors.
- It might be useful to prove $a^2+b^2 \leq (a+b)^2$ for $a\ b :: nat$
- Have a look at the lemma *sum_length_filter_compl*

lemmas *length_induct_rule* = *measure_induct_rule*[**where** *f*=*length*, *case_names shorter*]

lemma "$c\_qsort\ k\ xs \leq (length\ xs)^2$"
proof (*induction xs rule*: *length_induct_rule*)
  case (*shorter xs*) **thm** *shorter.IH*
  show *?case* **proof** (*cases xs*)
    case *Nil*
    then show *?thesis* **by** *auto*
  next
    case (*Cons x xs'*)

Insert your proof here

  qed
qed

For 3 bonus points, show that quicksort is stable. You will probably run into subgoals containing terms like $[x{\leftarrow}xs\ .\ k\ x < k\ p \wedge k\ x = a]$. Try to find a simpler form for them. (Cases on $a{<}k\ p$)!

lemma *qsort_stable*: "$[x{\leftarrow}qsort\ k\ xs\ .\ k\ x = a] = [x{\leftarrow}xs\ .\ k\ x = a]$"