Advanced Models and Programs
Carsten Schürmann
carsten@itu.dk

Homework 2

Please hand in a printout of your code by
Wednesday, 14 February 2007
before class meets.
Extensions can *only* be granted before the deadline
(if necessary, contact the teaching staff by email in English, please.)

Guidelines

While we acknowledge that beauty is in the eye of the beholder, you should nonetheless strive for elegance in your code. Not every program which runs deserves full credit. Make sure to state invariants in comments which are sometimes implicit in the informal presentation of an exercise. If auxiliary functions are required, describe concisely what they implement. Do not reinvent wheels, and try to make your functions small and easy to understand. Use tasteful layout and avoid long winded and contorted code.

Problem 1

Consider the following definition of a single step rotation operation on lists.

$$\texttt{rotateSingle}\ L = \begin{cases} L & \text{if } L \text{ is the empty list} \\ T\ \texttt{@}\ \texttt{[}H\texttt{]} & \text{if } L \text{ is a list with head } H \text{ and tail } T \end{cases}$$

Using this operation we can rotate a list step by step:

`rotateSingle [1,2,3,4]` $\Downarrow$ `[2,3,4,1]`

and

`rotateSingle [2,3,4,1]` $\Downarrow$ `[3,4,1,2]`.

1. Implement a generalization of this function to multiple rotation steps. The function should be of type

   ```
   rotate : int -> 'a list -> 'a list
   ```

   where the first arguments indicates how many single steps should be executed.

2. Implement a function that computes the length of a list. This function is of type

   ```
   length : 'a list -> int.
   ```

3. Show the following claim. For all lists L : $'a$ `list`, if `length` $L \Downarrow N$ then `rotate` $N\ L \Downarrow L$. You might use the following properties.

$$\begin{aligned} \texttt{nil @ } L &\equiv L \\ L \texttt{ @ nil} &\equiv L \\ L_1 \texttt{ @ (} L_2 \texttt{ @ } L_3 \texttt{)} &\equiv \texttt{(} L_1 \texttt{ @ } L_2 \texttt{) @ } L_3 \end{aligned}$$

Hint: First you need to generalize the claim, just as we did it in class.

Problem 2: Sorting

In this problem you are asked to implement a sorting function in SML. The idea behind the sorting algorithm is the following: From the list that is to be sorted, pick the first element, and make this the pivot element. Next, split the remaining list into two smaller lists, namely one that contains all elements less then the pivot, and another that contains all that are greater. Then, recursively, sort those sublists. When returning from the recursive call, paste together the sorted list of elements less then pivot, the list containing the pivot, and the sorted list of elements greater than the pivot, and you are done.

The type of your sorting function is expected to be

```
sort : 'a list -> 'a list
```

Here is an example run.

```
sort [1,6,3,7,12,5,7,8,4,3,2] ⇓  [1,2,3,3,4,5,6,7,7,8,12]
```

1. Implement your `sort` function. You will need to implement a `split` function as well that splits the input list according to the pivot. Both functions might be mutually recursive in which case you need to declare the first function using the keyword `fun`, and the second using `and`.

2. Argue why your function terminates on all inputs.

Please not write a long contorted program, a few lines is all that is needed!