ITPPV Homework 1

due 23:59, Tuesday January 28, 2020

1 Setting up the Environment

We will use the HOL4 theorem prover[1] in the course. For the homeworks, you will need to be able to use HOL4 on your own machine. Therefore, please set up the following software.

1.1 Standard ML

You need to have an implementation of Standard ML (SML). Please install PolyML[2] 5.8[3].

1.2 HOL4

Please install version Kananaskis-13 of the HOL4 theorem prover[4]. Installation instructions can be found on HOL4's website[5].

1.3 Emacs

In the lecture GNU Emacs[6] will be used as a user-interface. Please install a recent version of Emacs. Please make sure you use Emacs and not XEmacs.

1.4 HOL-mode and SML mode

We will use *hol-mode* for Emacs. It is distributed with HOL4, but needs setting up in Emacs. Please set it up and familiarise yourself with its basic usage. Documentation can be found on HOL4's website[7]. We will write SML programs often. Please install the SML mode[8] to enable syntax highlighting for SML in Emacs. Information on both the SML and the HOL mode can also be found in HOL4's interaction manual[9].

2 Standard ML

Let's refresh our knowledge about Standard ML. The exercises below are aimed at getting familiar with Emacs and using the SML mode, as preparation for using the HOL mode.

To learn more about the HOL Emacs mode, you can have a look at the HOL4 interaction manual linked above. If you need a brush-up on SML syntax, we recommend reading something compact like `https://learnxinyminutes.com/docs/standard-ml/`. If you need more material, the book *ML for the Working Programmer* by Larry Paulson is a good introduction.

[1]`https://hol-theorem-prover.org`
[2]webpage `http://www.polyml.org`
[3]download link `https://github.com/polyml/polyml/releases/tag/v5.8`
[4]download link `https://github.com/HOL-Theorem-Prover/HOL/releases/tag/kananaskis-13`
[5]see `https://hol-theorem-prover.org/#get`
[6]https://www.gnu.org/software/emacs/
[7]see `https://hol-theorem-prover.org/hol-mode.html`
[8]`https://elpa.gnu.org/packages/sml-mode.html`
[9]`https://hol-theorem-prover.org/HOL-interaction.pdf`

2.1 Your Own Lists

SML comes with a decent list library. Nevertheless, as an exercise, define your own list datatype and implement the following list operations for that datatype:

- `length`
- `append` (`@`)
- `rev`
- `revAppend`
- `exists`

If you don't know what these functions should do, you can find documentation of the Standard ML Basis Library at e. g., `http://sml-family.org`. In addition, implement a function

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

which is supposed to construct a list of the given length that only contains the given element. For example, `replicate "a" 3` should return the list `["a", "a", "a"]`.

1. Prove on pen and paper, using structural induction on your list datatype, that for your implementation, `append l [] = l` holds for all `l`.
2. Similarly, prove using pen and paper that:
   ∀`l1 l2. length (append l1 l2) = length l1 + length l2`.
3. There are strong connections between `append`, `revAppend` and `rev`. One can for example define `revAppend` by `revAppend l1 l2 = append (rev l1) l2`. Write down similar definitions for `rev` and `append` using only `revAppend`.

2.2 Making Change

In the following, let's use the standard SML list library again. Write a program that, given the coins and notes you have in your wallet, lists *all* the possible ways to pay a certain amount. Represent the coins you have as a list of integers. If a number occurs twice in this list, you have two coins with this value. The result should be returned in the form of a list of lists. For simplicity, the output may contain duplicates. The function should have the following signature:

```
make_change : int list -> int -> int list list
```

An implementation of the function can for example have the outputs below. Note that the output of your implementation is allowed to contain duplicates and use a different order of the lists.

-
  ```
  make_change [5,2,2,1,1,1] 6 =
      [[5, 1], [2, 2, 1, 1]]
  ```
- `make_change [5,2,2,1,1,1] 15 = []`

-
  ```
  make_change [10,5,5,5,2,2,1,1,1] 10 =
      [[10], [5, 5], [5, 2, 2, 1], [5, 2, 1, 1, 1]]
  ```

Write down as formally as you can some properties of `make_change`. An example property is

$$\forall \texttt{cs n.}\ \ \texttt{n > sum cs} \Longrightarrow \texttt{make\_change cs n = []}$$

where `sum` is defined by `val sum = foldl (op+) 0` and we assume that `cs` contains no number less than 0.