

# ITP Exercise 1

due Friday 28th April

## 1 Setting up Environment

We will use the HOL theorem prover<sup>1</sup>. For the practical sessions you will need to be able to use HOL on your own machine. Therefore, please set up the following software.

### 1.1 Standard ML

You will need to have Standard ML available. Please install PolyML 5.6<sup>23</sup> or later.

### 1.2 HOL

Please install a recent version of the HOL theorem prover. I recommend installing the most recent version from the git repository. If for some reason you don't want to do this, the latest release should be fine as well. Installation instructions can be found on HOL's webpage<sup>4</sup>.

### 1.3 Emacs

In the lecture GNU Emacs<sup>5</sup> 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 the *hol-mode* for Emacs. It is distributed with HOL, but needs setting up in Emacs. Please set it up and familiarise yourself with its basic usage. Documentation can be found on HOL's webpage<sup>6</sup>. We will write SML programs all the time. Please install the SML mode<sup>7</sup> to enable syntax highlighting for ML in Emacs. Information on both the SML and the HOL mode can also be found in HOL's interaction manual<sup>8</sup>.

## 2 SML

Let's refresh our knowledge about Standard ML. Moreover, these exercises are aimed at getting familiar with Emacs and using the HOL mode. So, please use Emacs with HOL mode as a user-interface and treat HOL like a ML REPL.

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

---

<sup>1</sup><https://hol-theorem-prover.org>

<sup>2</sup>webpage <http://www.polyml.org>

<sup>3</sup>download link <https://github.com/polyml/polyml/releases/tag/v5.6>

<sup>4</sup>see <https://hol-theorem-prover.org/#get>

<sup>5</sup><https://www.gnu.org/software/emacs/>

<sup>6</sup>see <https://hol-theorem-prover.org/hol-mode.html>

<sup>7</sup><https://elpa.gnu.org/packages/sml-mode.html>

<sup>8</sup><https://hol-theorem-prover.org/HOL-interaction.pdf>

## 2.1 Our Own Lists

Of course SML comes with a decent list library. However, as an exercise implement your own list datatype and implement the following list operations for your own 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 with pen and paper that for your implementation `append l [] = l` holds for all `l`.
2. Similarly, prove  $\forall l1\ l2. \text{length } (\text{append } l1\ l2) = \text{length } l1 + \text{length } l2$ .
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 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.

An example implementation of the function `make_change : int list -> int -> int list list` should show for example the following outputs. Notice however, 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

$$\forall cs\ n. \quad n > \text{sum } cs \implies \text{make\_change } cs\ n = []$$

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