Project 2: Modelling and Proving in a Higher Order Theorem Prover

Program Verification Course 12/13

1 Setup

In this assignment we will consider a simple program to verify. The emphasis is in modelling the problem in HOL, and to subsequently prove some correctness properties of the program.

Let N be some natural number. Consider the following set of processes:

$$\mathsf{Sys}_N \quad = \quad \{\mathsf{PMIN}\ (N{-}1)\ 0\} \ \cup\ \{\mathsf{PMIN}\ i\ (i{+}1) \mid i < N{-}1\ \}$$

where each $\mathsf{PMIN}\ p\ q$ operate on a common array A of integers, and it does this:

$$\mathsf{PMIN}\ p\ q \ = \ \texttt{if}\ A[q]{<}A[p]\ \texttt{then}\ A[p] := A[q]\ \texttt{else skip}$$

You can assume the array to be infinitely large so you don't have to worry about accessing the array outside its bound.

The Sys above runs non-stop. At every step of each execution, one of the processes is selected, and executed. We leave the selection scheme unspecified.

We want to verify the following properties:

1. [DESCD] Every execution step of Sys_N never increases the value of $A[i]$, for any $i < N$.
2. [EQSTABLE] For every execution step of Sys_N, if all values in $A[0..N{-}1]$ are the same before the step, they all remain the same after the step.

Note: what makes verification in a theorem prover different than model checking is that we can in principle verify the above for any value of N and any value of A.

2 Base Task

The verify the above properties in HOL, you will have to 'express' (to 'model') the problem in HOL first. The higher order feature of HOL means that it is very expressive, allowing you to model many kinds of problems. Also, there will be plenty of choices for expressing your models, e.g. you can use functions, sets, or lists. In this exercise we will explore these choices.

Being able to express is one thing, being able to prove is another, and the latter is usually more difficult. Although logicalluy The choices you made in expressing your problems may matter alot for the ease of your verification.

2.1 Preparation

Prepare a script e.g. `mysolution.smx` for writing your models and proofs. If you are a poor Windows user like myself, you may have to turn off HOL's unicode support:

```
set_trace "Unicode" 0;
```

Numeric terms are by default typed as `num`. Integers are also not loaded by HOL by default, so to use it you need to load `intLib`. However, once integers are loaded, the default type of for numerics changes to `int`.

Here is myown script preamble if you want to use it:

```
(* for Windows *)
set_trace "Unicode" 0;

load "numSimps"  ; (* simplification-set for natural number arithmetic *)
open numSimps    ; (* openning allow you to access stuffs in a loaded module
                      without having to use qualified notation like A.foo *)
load "intLib"    ; (* library for integers *)
open intLib      ;
load "stringLib" ; (* library for strings, if you need it *)
open stringLib   ;
load "stringSimps" ; (* simplification sets for string, if you need it *)
open stringSimps   ;
load "pairTheory"  ; (* definitions of FST and SND; see on-line doc. on pairTheory *)
open pairTheory    ;
load "pred_setLib" ; (* library for sets *)
open pred_setLib ;
load "pred_setSimps" ; (* simplification-set for sets *)
open pred_setSimps   ;
```

2.1.1 Relevant chapters from the HOL Description

Section 5.3, 5.5, 5.6 gives you some power tactics to do rewriting, goals solving, and induction. Check out Section 5.5.2 about simpsets (simplification sets) and how to combine them. Section 3.5.1 explains sets.

Some functions that may come handy (check out their documentation): `REWRITE_TAC` and all its variations, `EXISTS_TAC`, `by` (in `bossLib`).

2.1.2 Scripting your proofs

I do **not** want to see proofs scripted as a series of commands, e.g.:

```
g ‘FST (MINpair (x,y)) <= SND (MINpair (x,y))‘ ;
e (RW_TAC std_ss [MINpair_DEF]) ;
e (COOPER_TAC) ;
val lemma1 = it ;
```

Above is how you interactively trying to discover how to prove the goal. But the proof should be scripted like this instead:

```
val lemma = prove(
    --‘FST (MINpair (x,y)) <= SND (MINpair (x,y))‘--,
    RW_TAC std_ss [MINpair_DEF]
    THEN COOPER_TAC ) ;
```

2.2 Define PMIN

Define PMIN in HOL. It is a simple 'program'. But how do you represent/model it in HOL?

2.3 Define Sys

Sys_N is a set of processes. We can represent such a set as `set` in HOL, or as a predicate (a function of type $a \to$ `bool`), or as a list. Write three models of Sys_N using each of those mentioned choices.

2.4 Formalizing the specifications

Give HOL formulas that formally express the specifications DESCD and EQSTABLE in Section 1.

2.5 Verification

Now you can verify DESCD and EQSTABLE. Do this first for your set and predicate models [max. 7pt].

I expect the list model to be more difficult to handle. I suggest you to first prove the following lemma. If $N > 0$:

$$\begin{array}{l}\mathsf{MEM}\ P\ \mathsf{ListSys}_N \\ \Rightarrow \\ ((P = \mathsf{PMIN}\ (N{-}1)\ 0)\ \vee\ (\exists k.\ P = \mathsf{PMIN}\ k\ (k{+}1)))\end{array}$$

where MEM x s checks the membership of x in the list s, and ListSys is your list version of `Sys`.

Verification of the list model is worth max. 2pt.

3 To deliver

1. The source code containing your models and proofs.
2. A short report [max. 1 pt] explaining your models and the formalization of the specifications.