Modelling and verifying a simple distributed system in HOL

João Paulo Pizani Flor
Department of Information and Computing Sciences,
Utrecht University - The Netherlands
e-mail: j.p.pizaniflor@students.uu.nl

April 14th, 2013

1 Introduction

This report describes the solution to one of the practical assignments (project nr. 2) of the master course "Program Verification" at Utrecht University, taught in the 3rd period of the academic year 2012/2013.

The goal of this assignment was to provide some experience in using an Automated Theorem Proving tool (in this case, http://hol.sourceforge.net) for the modelling and formal verification of software. More specifically, we were asked to model a simple distributed algorithm in HOL logic (using three possible representations) and then prove some properties over this model.

2 Modelling the algorithm itself

The system we were asked to model consisted of a set of processes Sys_n:

$$Sys_n = \{\text{PMIN}(n-1, 0)\} \cup \{\text{PMIN}(i, i+1) \mid i < n-1\}$$

All processes belonging to this set operate on a shared (global) array on integers, with the following behaviour:

$$PMIN(m, n) = \text{if } A[n] < A[m] \text{ then } A[m] = A[n] \text{ else skip}$$

The two last remarks in the specification were that the global array A could be assumed to be infinite (no concerns with boundary conditions) and that at each execution step of the system, one process would be selected and run, be nothing could be assumed about which process would be selected at which point (selection order left unspecified).

Our first step towards embedding this system in HOL was deciding how to represent the array A. We chose to represent it as a function $a : \mathbb{N} \to \mathbb{Z}$, where acessing the array at an index i corresponds to the value $a(i)$. Considering this model, an array update is a transformation between the function models, such that the "updated" element is a specialized behaviour in a case distinction by index.

This led to our model of a process `PMIN` (in HOL term syntax):

```
PMIN (p : num) (q : num) (ar : num -> int) =
    let changed = (\i. if i = p then ar q else ar i)
    in if (ar q) < (ar p) then changed else ar
```

Notice how the output array is the same as the input in the case that the `if` test fails, and the changed array maps to the same values as the original for **all indices except p**.

Having `PMIN` modelled, the next logical step was to model the set of processes Sys_n. This modelling was done in three ways, which results in three varieties (semantically equivalent) of the specifications that we later discuss. The first and most intuitive modelling is by using HOL's built-in `set` datatype. After loading the `pred_setLib` library, we gain access to definitions of set theory such as membership testing (`IN`) and set extension by inserting a single element into a set (`INSERT`), along with additional syntax in the term parser for dealing with set-builder notation. All these goodies allow us to have a concise definition of Sys_n, as follows:

```
SYS_SET (n : num) = (PMIN (n-1) 0) INSERT {PMIN i (i+1) | i < (n-1)}
```

The second way to model a set is using a boolean-valued function – that is, a function of type $(\alpha \rightarrow \text{Bool})$, called a "characteristic function" – which, given an element returns true in case the element belongs to the set. In fact, the HOL built-in set type already uses characteristic functions, so our `SYS_FUN` model looks very similar to the `SYS_SET` model already presented.

```
SYS_FUN (n : num) = \p.
    (p = PMIN (n-1) 0) ∨ (∃i. i < (n-1) ∧ p = PMIN i (i+1))
```

That is, given a natural number n, the `SYS_FUN` definition yields a boolean-valued characteristic function that returns true when the input equals `PMIN (n-1) 0` or `PMIN i (i+1)` for any $i < (n-1)$.

Lastly, we have the definition using the list built-in type from HOL. In this definition, we map the range of indices from n down to 0 to the corresponding processes. For this, we need the auxiliary recursive definition of `DOWNTOZERO`.

```
   DOWNTOZERO 0       = [0]
/\ DOWNTOZERO (SUC n) = (SUC n) :: DOWNTOZERO n

   SYS_LIST (n : num) =
       PMIN (n-1) 0 :: MAP (\i. PMIN i (i+1)) (DOWNTOZERO (n-2))
```

3 Formalizing the specifications to be verified

Given the "basic" definitions above, now we give HOL definitions formalizing the specifications to be verified. First of all, there will be three categories of HOL definitions, one category for each of the approaches we considered (set, function, list). Also, the specifications will be expressed using two auxiliary predicates, `SAME` and `DESCD`:

```
SAME (n : num) a = ∀i j. (i < n ∧ j < n) ⇒ (a i = a j)
```

The unary predicate `SAME n` tests whether, for a given array, all its elements have the same value. The definition literally states that, for every pair of inidices i and j inside the range of the array up to n, $a[i] = a[j]$.

```
NONINCR (n : num) a b = ∀i. (i < n) ⇒ (a i) ≥ (b i)
```

The binary predicate `NONINCR n` tests whether a relation holds between two arrays, namely that for every index in the range up to n, $a[i] \leq b[i]$.

3.1 Specifications using the set model

When using the set model, the specifications correspond to the following HOL definitions:

```
SYS_EXEC_SET n a_in a_out = ∃p. (p ∈ (SYS_SET n)) ∧ (p a_in = a_out)
```

The definition `SYS_EXEC_xxx n` establishes a binary relation between two arrays, and the meaning of this relation is that the second array is the result of one execution step of the algorithm, considering the first array as input.

Furthermore, the definitions of the properties themselves are:

```
EQSTABLE_SET = ∀n a b. (SAME n a) ∧ (SYS_EXEC_SET n a b) ⇒ SAME n b
```

```
DESCD_SET = ∀n a b. SYS_EXEC_SET n a b ⇒ NONINCR n a b
```

3.2 Specifications using the function model

When using the function model, the specifications correspond to the following HOL definitions:

```
SYS_EXEC_FUN n a_in a_out = ∃p. ((SYS_FUN n) p) ∧ (p a_in = a_out)
```

```
EQSTABLE_FUN = ∀n a b. (SAME n a) ∧ (SYS_EXEC_FUN n a b) ⇒ SAME n b
```

```
DESCD_FUN = ∀n a b. SYS_EXEC_FUN n a b ⇒ NONINCR n a b
```

3.3 Specifications using the list model

When using the list model, the specifications correspond to the following HOL definitions:

```
SYS_EXEC_LIST n a_in a_out = ∃p. (MEM p (SYS_LIST n)) ∧ (p a_in = a_out)
```

```
EQSTABLE_LIST = ∀n a b. (SAME n a) ∧ (SYS_EXEC_LIST n a b) ⇒ SAME n b
```

```
DESCD_LIST = ∀n a b. SYS_EXEC_LIST n a b ⇒ NONINCR n a b
```