1. Introduction

This draft presents an introduction to the use of **PVS** for validating algorithms.

`./tex2code.pl < logicalalgo.tex`

2. The maximum of a set of numbers

An algorithm for computing the maximum of a set of numbers is presented in [**?**]. Misra presents both operational and non-operational descriptions for that same algorithm, using the logic of **UNITY**. That description essentially establishes relation between the states of program to be defined. That description is reproduced below, followed by a **PVS** encoding of it.

3. A **UNITY** description

Fact: An execution of a program can be represented as a sequence of states.

- **state**: a state of a program is an assignment of values to all its variables.
- **action:** a temporal logic expression defining a relation between *old states* and *new* states. For example,

  $$p \ \mathbf{co} \ q$$

  that states (using Hoare triples)

  $$\forall \alpha . \{p\} \ \alpha \ \{q$$

  that for any action of the system in a state where p holds, a new state will result where q will hold.

  An action is a logical relation constraining the *transitions*, or *updates* in the language of [**?**]. (I'll use the terms *transition* and *update* to designate synonyms for the usual *step* of a program, and say that actions constraining the transitions/steps of a program.)

A logical description of a program defines a set of conditions that any implementation of it should satisfy. Misra, in [**?**, **?**], uses a formal system (called UNITY), which combines a temporal logic to present logical descriptions of programs and a language (whose semantics is expressed in the logic of UNITY) for representing operational descriptions of programs. As an example, he introduces a program for computing the maximum of a set of numbers as given in 1. Notice the use of the operator **co** described above.

Algorithm 1.

program *max*
initially $v = 0$
$\alpha :: v = m$ **co** $v = m \vee (v \in S \wedge v > m)$
$\beta :: m \in S \leadsto v \geq m$

In a logical description, the steps to be performed are described through predicates that represent logical relation between states. In a sense, 1 above describes a family of programs. The programs in that family should obey the following conditions:

- **initially** $v = 0$
  Before any computation v must be initialized to 0.
- $v = m$ **co** $v = m \vee (v \in S \wedge v > m)$
  which states that $\forall m. \forall \alpha : \{v = m\}\ \alpha\ \{v = m \vee (v \in S \wedge v > m)$
  For any *transition* in this program, if v equals m before the transition, then after it either v is still equal to m or it has been changed to some value from S and $v > m$.
- $m \in S \leadsto v \geq m$
  that establishes that for any m if $m \in S$ in a given state, then eventually v will be greater or equal than m.

Property α is a safety property: v never decreases and, when it changes, it moves to a new value from S.

Property β ensures progress. It wouldn't help if v wad initiallized to 0 and then never changed. α would still hold, but the maximun in S would never be obtained. β enforces that S will be completely scanned: eventually any value in S will have been compared to v.

4. A **PVS** DESCRIPTION

The algorithm is coded here as **PVS** theory. The first line defines that the theory is called *max*.

```
max : THEORY
  BEGIN
```

Next two lines define:

- **v**: that identify the maximum at the end of computation.
- **S**: the set whose maximum is being searched.

```
  v : nat;
  S  : setof[nat];
```

UNITY's **co** operator relates a state with its successor. A state will be understood as a map from variables to values [**?**]. However, because **PVS** doesn't have temporal operators, and a relation on sequences of states must be expressed, the concept of state will be modeled through a function mapping pairs of integers to integers:

- **st** represents the state function. Given a non-negative integer representing the states (states are numbered from 0), and the variable v (another non-negative integer), it returns the value of v in the respective state.

```
 st : [nat,nat->nat];
```

The axiom **INIT** defines the initial state (thus, 0) of the computation. In state 0, v has value 0.

```
 INIT: AXIOM st(0,v)=0;
```

Axiom **NEXT** relates the value of v in state i (say m) to its value in the succeeding state $(i+1)$, which must be either m or some number greater than m and contained in S.

```
 NEXT: AXIOM forall (i:nat,m:nat):
                 st(i,v)=m IMPLIES
                    (st(i+1,v)=m OR
                       (st(i+1,v)>m AND S(st(i+1,v))))
```

Next the *formula* **STABLE** is introduced. A formula is a conjecture that must be proved. This conjecture in particular states a *stability* property: that if v is greater than some n in a state, it will still be grater than n in the succeeding state.

```
 STABLE: FORMULA forall(i:nat,n:nat): st(i,v)>=n IMPLIES
                                        st(i+1,v)>=n



END max

```

The **PVS** proof follows.

Verbose proof for `STABLE`.

`STABLE:`

$$\{1\} \quad \forall\ (i\text{: nat},\ n\text{: nat})\text{: st}(i,\ v)\ \geq\ n \supset \text{st}(i+1,\ v)\ \geq\ n$$

Skolemizing,

`STABLE:`

$$\begin{array}{|ll}
 & \\ \hline
\{1\} & \mathrm{st}(i',\ v) \geq n' \supset \mathrm{st}(i'+1,\ v) \geq n'
\end{array}$$

Applying disjunctive simplification to flatten sequent,
`STABLE:`

$$\begin{array}{|ll}
\{\text{-}1\} & \mathrm{st}(i',\ v) \geq n' \\ \hline
\{1\} & \mathrm{st}(i'+1,\ v) \geq n'
\end{array}$$

Using n_0 to name and replace $\mathrm{st}(i',\ v)$,
`STABLE:`

$$\begin{array}{|ll}
\{\text{-}1\} & \mathrm{st}(i',\ v) = n_0 \\
\{\text{-}2\} & n_0 \geq n' \\ \hline
\{1\} & \mathrm{st}(i'+1,\ v) \geq n'
\end{array}$$

Forward chaining on NEXT,
we get 2 subgoals:
`STABLE.1:`

$$\begin{array}{|ll}
\{\text{-}1\} & \mathrm{st}(i'+1,\ v) = n_0 \\
\{\text{-}2\} & \mathrm{st}(i',\ v) = n_0 \\
\{\text{-}3\} & n_0 \geq n' \\ \hline
\{1\} & \mathrm{st}(i'+1,\ v) \geq n'
\end{array}$$

Trying repeated skolemization, instantiation, and if-lifting,
This completes the proof of `STABLE.1`.
`STABLE.2:`

$$\begin{array}{|ll}
\{\text{-}1\} & \mathrm{st}(i'+1,\ v) > n_0 \\
\{\text{-}2\} & S(\mathrm{st}(i'+1,\ v)) \\
\{\text{-}3\} & \mathrm{st}(i',\ v) = n_0 \\
\{\text{-}4\} & n_0 \geq n' \\ \hline
\{1\} & \mathrm{st}(i'+1,\ v) \geq n'
\end{array}$$

Trying repeated skolemization, instantiation, and if-lifting,
This completes the proof of `STABLE.2`.
Q.E.D.

5. A first abstract data type

```
stacks [t: TYPE+] : THEORY
  BEGIN

  stack : TYPE+
  s : VAR stack
```

```
  empty : stack
  nonemptystack?(s) : bool = s /= empty
  nonemptystack_exists: AXIOM EXISTS s: nonemptystack?(s)

  push: [t, stack -> (nonemptystack?)]
  pop: [(nonemptystack?) -> stack]
  top: [(nonemptystack?) -> t]

  x,y: VAR t

  push_top_pop: AXIOM
    nonemptystack?(s) IMPLIES push(top(s), pop(s)) = s

  pop_push: AXIOM pop(push(x, s)) = s

  top_push: AXIOM top(push(x, s)) = x

  pop2push2: THEOREM pop(pop(push(x, push(y, s)))) = push(y,s)

END stacks
```