FoVeOOS competition 2011
Third Challenge

Krakatoa/Why3 team

October 4, 2011

1 Solution in Why3

The Why3ML version of the challenge code, is as follows. (verbatim copy, sorry for the length file).

The first part is a theory defining a generic option type. the predicate `eq_opt` decides whether a value x is equal to the value stored in option o. it is of course false if o is None.

The second part is the module defining the code.

The main function is `duplets`, which proceeds by calling an auxiliary function `duplet` which finds one duplet.

To ease the specification, a predicate `is_duplet` is introduced, which tells that a pair (i,j) are indexes for a duplet in an array a.

To make the main function work, the auxiliary function should be able to return a duplet that is different from the first one found. This is the role of the extra parameter `except` : the duplet to find must be have a value different from this option.

The loops in the code of the auxiliary function must be annotated with proper invariants to ensures that no duplet have been found yet. This is indeed necessary only for proving that the loop cannot exit normally.

```
(*
COST Verification Competition
Please send solutions to vladimir@cost-ic0701.org

Challenge 3: Two equal elements

Given: An integer array a of length n+2 with n>=2. It is known that at
least two values stored in the array appear twice (i.e., there are at
least two duplets).

Implement and verify a program finding such two values.

You may assume that the array contains values between 0 and n-1.
```

```
*)

theory Option

  type option 'a = None | Some 'a

  predicate eq_opt (x:'a) (o:option 'a) =
    match o with
    | None -> false
    | Some v -> v=x
    end

end

module Simple

  use import int.Int
  use import Option
  use import module ref.Ref
  use import module array.Array

  (* duplet in array a is a pair of indexes (i,j) in the bounds of array
     a such that a[i] = a[j] *)
  predicate is_duplet (a:array int) (i:int) (j:int) =
     0 <= i < j < length a /\ a[i] = a[j]

  exception Break

  (* (duplet a) returns the indexes (i,j) of a
     duplet in a.
  *)
  let duplet (a:array int) (except:option int) =
     { 2 <= length a /\
       exists i j:int. is_duplet a i j /\
                       not (eq_opt a[i] except) }
     let res = ref (0,0) in
     try
      for i=0 to length a - 2 do
        invariant {
          forall k l:int. 0 <= k < i /\ k < l < length a ->
            not (eq_opt a[k] except) -> not (is_duplet a k l)
        }
         let v = a[i] in
         if eq_opt v except then () else
         for j=i+1 to length a - 1 do
           invariant {
```

```
            forall l:int. i < l < j -> not (is_duplet a i l)
          }
            if a[j] = v then
              begin
               res := (i,j);
               raise Break
              end
        done
      done;
      assert { forall i j:int. not (is_duplet a i j) };
      absurd
     with Break -> !res
     end
  { let (i,j) = result in
    is_duplet a i j /\ not (eq_opt a[i] except) }


  let duplets (a: array int) =
    { 4 <= length a /\ exists i j k l:int.
      is_duplet a i j /\ is_duplet a k l /\ a[i] <> a[k]
    }
    let (i,j) = duplet a None in
    let (k,l) = duplet a (Some a[j]) in
    ((i,j),(k,l))
    { let ((i,j),(k,l)) = result in
      is_duplet a i j /\ is_duplet a k l /\ a[i] <> a[k] }

end

(*
Local Variables:
compile-command: "why3ide challenge3_why3.mlw"
End:
*)
```

predicate `is_duplet` defines what means for a given integer to be greater or equal all values in a tree.

The proof proceeds automatically by several theorem provers. Proofs are separate d between the `duplet` function and the `duplets` function, and in each case the verification condition, a large conjunction, was split into parts. The time limit was 10 sec.

Proof obligations		Alt-Ergo 0.93	CVC3 2.2	Z3 2.19
parameter duplet	assertion	0.03	0.02	0.04
	parameter duplet	0.02	0.02	0.04
	for loop initialization	0.01	0.03	0.01
	for loop preservation	(timeout)	0.12	0.15
	assertion	0.07	0.04	0.05
	parameter duplet	0.04	0.03	0.04
parameter duplets	precondition	0.04	0.03	0.04
	precondition	0.04	0.03	0.04
	precondition	0.04	0.03	0.04
	normal postcondition	0.06	0.03	0.04

2 About a Solution using Krakatoa

Proving the same example in Java, annotated, with Krakatoa should not rise any difficulty, apart maybe the fact that we need a way to return pairs of integers.