CS412 Exercise sheet 8

Verification of refinements and understanding loops

1. Using the definitions given in lectures, are either of the versions of $AbSetR$ a refinement of $AbSet$? Justify using the proof obligations for refinement.
   If the conditions cannot be proved, suggest how the refinement machine might be modified to provide a valid refinement.

2. Consider the *Colours* machine and its refinement *ColoursR* given below. First, verify the refinement initialisation proof obligation for these two machines. Then, for each of the following changes, suggest whether the initialisation obligation is still satisfied. Justify your answers by expanding the formal condition.

   (a) The refinement initialisation is $colour\ :=\ red$.

   (b) The refinement initialisation is $colour\ :\in\ COLOUR$.

   (c) The abstract initialisation is $cols := \{\}$.

   (d) The abstract initialisation is $cols := \{red\}$.

3. Consider the $ColoursR2$ machine.

   (a) Show that $ColoursR2$ refines *Colours*.

   (b) Would it still be a refinement if the invariant of $ColoursR2$ were just *true*?

   REFINEMENT $ColoursR2$

   REFINES *Colours*

   INVARIANT $red \in cols$

   OPERATIONS

   $add(cc) \mathrel{\widehat{=}} skip;$
   $cc \leftarrow query \mathrel{\widehat{=}} cc := red;$
   $change \mathrel{\widehat{=}} skip$

   END

4. Show whether the following loops meet their specifications.

   (a) Here, b is an array with domain $1\,..\,10$.

   $\{true\}$
   $i, s := 10, 0;$
   WHILE $i \neq 0$ **DO** $i, s := i - 1, s + b(i)$ **END**
   INVARIANT $0 \leq i \leq 10 \wedge s = \Sigma k \bullet (i + 1 \leq k \leq 10 \mid b(k))$
   VARIANT i
   $\{s = \Sigma k \bullet (1 \leq k \leq 10 \mid b(k))\}$

(b) Here, b is an array with domain $1 \mathinner{\ldotp\ldotp} n$.

$\{1 \leq n\}$
$i := 1;$
WHILE $i \leq n \wedge x \neq b(i)$ **DO** $i := i + 1$ **END**
INVARIANT $1 \leq i \leq n + 1 \wedge x \notin b[1 \mathinner{\ldotp\ldotp} i - 1]$
VARIANT $n - i$
$\{(1 \leq i \leq n \wedge x = b(i)) \vee (i = n + 1 \wedge x \notin b[1 \mathinner{\ldotp\ldotp} n])\}$

5. Using a loop, develop a program that, given a fixed array $b[1 \mathinner{\ldotp\ldotp} n]$ with $n \geq 1$, stores in d the number of times the value x occurs in b. Verify that your program meets its specification.

The Colours machine

```
MACHINE         Colours
SETS            COLOUR = {red,blue,green}
VARIABLES       cols
INVARIANT       cols <: COLOUR
INITIALISATION  cols :: POW(COLOUR - {blue})
OPERATIONS
  add(cc) = PRE  cc:COLOUR
            THEN cols := cols \/ {cc}
            END;

  cc <-- query   = PRE   cols /= {}
                   THEN cc :: cols
                   END;

  change = ANY   newcols
           WHERE newcols <: COLOUR & newcols /= cols
           THEN  cols := newcols
           END
END

========================================================================

REFINEMENT     ColoursR
REFINES        Colours
VARIABLES      colour
INITIALISATION colour :: COLOUR - {blue}
INVARIANT      colour : cols
OPERATIONS
  add(cc) = colour :: {colour,cc};

  cc <-- query = cc:= colour;

  change =  colour :: COLOUR - {colour}
END

========================================================================
```

The AbSet machine

```
MACHINE        AbSet
VARIABLES      myset
INVARIANT      myset : POW(NAT)
INITIALISATION myset := {}
OPERATIONS

   set_add(nn) = PRE nn:NAT THEN myset := myset \/ {nn};

   oo <-- set_out  = PRE myset /= {}  THEN  oo :: myset  END

END
======================================================================

First version of AbSetR machine
-------------------------------

REFINEMENT     AbsetR
REFINES        AbSet
VARIABLES      num
INVARIANT      num : NAT &  num : myset
INITIALISATION num :: NAT
OPERATIONS

   set_add(nn) = num := nn;

   oo <-- set_out = oo := num

END
======================================================================

Second version of AbSetR machine
-------------------------------

REFINEMENT     AbsetR
REFINES        AbSet
VARIABLES      myseq
INVARIANT      myseq : iseq(NAT) & myset = ran(myseq)
INITIALISATION myseq := {}
OPERATIONS

   set_add(nn) = PRE nn /: ran(myseq) THEN  myseq := myseq <- nn END;

   oo <-- set_out =  IF myseq = <> THEN  oo := 0 ELSE oo:= first(myseq) END

END
======================================================================
```