Section: Master 1 SIGL

Faculty of Mathematics and Computer Science
Department of Computer Science

Formal Verification and Specification
Lab Session (TP) 01

1. **RegulMach:**
   Implement in Atelier B the Reglum machine presented in the course.
2. **Eclude:**
   Define an abstract machine of a the Ecludian division
3. **PGCD:**
   Define an abstract machine of a the PGCD problem and its refinement in Method B
4. **8-bit calculator:**
   Define a an abstract machine and its implementation for 8-bit calculator. The calculator has two registers for storing values, one i principal (*Rp*), and other secondary (Rs). The calculator performs five(05) main operations:
   - *Increment:* which increments the principle register
   - *Decrement:* which decrements the principle register
   - *storeRp:* stores a value into *Rp*
   - *storeRs:* stores a value into *Rs*
   - *comp:* which compares between *Rs* and *Rp*

Answers

1. **RegulMach:**
   Implement in Atelier B the Reglum machine presented in the course.

   Answer: See course.

2. **Eclude:**
   Define an abstract machine of a the Ecludian division

```
MACHINE
Ecld
OPERATIONS
reste, quot <-- calculReste (divis , divid ) =
PRE
   divis : NAT ∧ divid : NAT ∧ divis > 0
  ∧ divis <= divid
THEN
ANY vq, vr WHERE
        vq : NAT
        ∧ vr : NAT
        ∧ divid = vq*divis + vr
    THEN
    quot := vq
    || reste := vr
    END
END
END
```

Figure 1: Abstract machine Eclud

   Answer:

3. **PGCD:**
   Define an abstract machine of a the PGCD problem and its refinement in Method B

```
MACHINE
    pgcd1 /* PGCD of two integers*/
    /* pgcd(x,y) is d | x mod d = 0 ∧ y mod d = 0
    ∧ ∀ other divisors dx d > dx
    ∧ ∀ other divisors dy d > dy */
OPERATIONS
rr <-- pgcd(xx,yy) = /* specification of pgcd */
PRE
    xx : INT & xx >= 1 & xx < MAXINT
    & yy : INT & yy >= 1 & yy < MAXINT
THEN
    ANY dd WHERE
        dd : INT
        & (xx - (xx/dd)*dd) = 0 /* d is a divisor of x */
        & (yy - (yy/dd)*dd) = 0 /* d is a divisor of y */
        /* and the other common divisors are < d */
        & !dx.((dx : INT & dx < MAXINT
        & (xx- (xx/dx)*dx) = 0 & (yy-(yy/dx)*dx)=0) => dx < dd)
        THEN
        END
END
END
```

Figure 2: Abstract machine PGCD

```
IMPLEMENTATION pgcd_i
REFINES pgcd1

OPERATIONS
rr <-- pgcd (xx, yy) = /* operation refined */
BEGIN
VAR cd, rx, ry, cr IN
cd := 1;
WHILE ( cd < xx & cd < yy) DO
    rx := xx - (xx/cd)*cd ; ry := yy - (yy/cd)*cd;
   IF (rx = 0 & ry = 0)
    THEN /* cd divise x et y, possible PGCD */
    cr := cd /* possible rr */
    END;
 cd := cd + 1 ; /* look for bigger */
INVARIANT
xx : INT & yy : INT & rx : INT & rx < MAXINT
& ry : INT & ry < MAXINT & cd < MAXINT
& xx = cr*(xx/cr) + rx & yy = cr*(y/cr) + ry
VARIANT
    xx - cd
END
END
END
END
```

Figure 3: Implementation PGCD

Answer:

4. **8-bit calculator:**
   Define a an abstract machine and its implementation for 8-bit calculator. The calculator has two registers for storing values, one i principal (*Rp*), and other secondary (Rs). The calculator performs five(05) main operations:

   - *Increment:* which increments the principle register
   - *Decrement:* which decrements the principle register
   - *storeRp:* stores a value into *Rp*
   - *storeRs:* stores a value into *Rs*
   - *comp:* which compares between *Rs* and *Rp*

```
MACHINE Calculette8
 VARIABLES rp, /* registre principal */ rs /* registre secondaire */
 INVARIANT rp : NAT & rs : NAT
     & 0 <= rp & rp <= 255 /* 8 bits */
     & 0 <= rs & rs <= 255 /* 8 bits */
 INITIALISATION rp :=0 || rs := 0
 OPERATIONS
 inc1 = /* incrementer le registre principal de 1 */
     PRE rp +1 <= 255 THEN
      rp := rp + 1
 END;
 dec1 = /* decrementer le registre principal de 1 */
      PRE rp - 1 >= 0 THEN
      rp := rp - 1
      END;
 storeRP(vv) =
      PRE vv : NAT & 0<= vv & vv <= 255 THEN
      rp := vv
 END;
  storeRS(val) =
  PRE val : NAT & val <= 0 & val <= 255 THEN
  rs := val
  END;
  res <-- cmp = /* comparer les contenus de RP et RS */
  res := bool(rs = rp);
  res <-- getRP = /* recuperer la valeur de RP */
  res := rp ;
  res <-- getRS = /* recuperer la valeur de RS */
  res := rs
 END
```

Figure 4: Abstract machine Claculette_8

```
IMPLEMENTATION Calculette8_i REFINES Calculette8
CONCRETE_VARIABLES rp , rs
INITIALISATION rp := 0 ; rs := 0
OPERATIONS
inc1 = BEGIN rp := rp + 1 END;
dec1 = BEGIN rp := rp - 1 END;
storeRP ( vv ) = BEGIN rp := vv END;
storeRS ( val ) = BEGIN rs := val END;
res <-- cmp = res := bool ( rs = rp );
 res <-- getRP = res := rp;
 res <-- getRS = res := rs
END
```

Figure 5: Abstract machine Claculette_8

Answer: