
Nada Abdallah, Cédric Saule, Delphine Longuet
Prof. Burkhardt Wolff
Parc Orsay Université
4, rue Jacques Monod
Building H / Room 012

VERIFICATION PAR PREUVE II

Date : 5.4.2010

Exercice 1 (Exponentiation Function I)

1. Give a specification of the program below in form of a pre and post-condition
2. Verify the program with the Hoare-Method!

```
S:=1; P:=N;
while P >= 1 do
    S:= S*X;
    P:= P-1;
```

Solution :

Exercice 2 (Exponentiation Function II)

```
S:=1; P:= N;
while P >= 1 do
    if P mod 2 /= 0 then P:= P - 1; S :=S*X else SKIP;
    S:= S * S;
    P:= P div 2
```

Solution :

Exercice 3 (Binary Search in an Array (Optional))

The following program implements a binary search of a value v in a sorted array t between 0 and $N - 1$. If the value is found at index k then the result is k otherwise it is -1 .

```
int binary(int t[], int N, int v) {
    int res = -1;
    int l = 0;
    int u = N-1;
    while (res==-1 && l <= u) {
        int m = (l + u) / 2;
        if (t[m] < v) l = m + 1;
```

```

        else if (t[m] > v) u = m - 1;
            else res=m;
    }
    return res;
}

```

Verify this C-program by the Dijkstra-Method by annotating the program and computing vcg! In particular, this means :

1. Write pre and post-condition for this function (do not forget the condition for avoiding access in `t` outside the bounds)
2. Find an invariant for the loop
3. Find a variant for justifying termination of the loop

Solution : *The solution is given in ACDL2 and could be presented by gwhy*

```

/*@ axiom mean : \forall int i, int j; i <= j => i <= (i+j)/2 <= j

/*@ requires \valid_range(t,0,N-1) && 0 <= N
   @ && (\forall int k1, int k2; 0 <= k1 <= k2 < N => t[k1]<=t[k2])
   @ ensures ((0 <= \result < N && t[\result]==v)
   @          ||(\result ==-1 && \forall int k; 0 <= k < N => t[k]!=v))
   @*/
int binary(int t[], int N, int v) {
    int res = -1;
    int l = 0;
    int u = N-1;
    /*@ invariant 0 <= l && u <= N-1 && -1 <= res <= N-1
       @ && (res == -1 => \forall int k; 0 <= k < N => t[k] == v => l <= k <= u)
       @ && (res >= 0 => t[res]==v)
       @ variant u-l-res
       @ */
    while (res==-1 && l <= u) {
        int m = (l + u) / 2;
        /*@ assert l <= m <= u
           if (t[m] < v)
                {/*@ assert (\forall int k; 0<=k<=m => t[k]<v)
                   l = m + 1;}
           else if (t[m] > v)
                {/*@ assert (\forall int k; m<=k<=(N-1) => v<t[k])
                   u = m - 1;}
           else res=m;
        }
    return res;
}

```