

&lt;

Cours Modélisation et vérification des systèmes informatiques  
 Exercices (avec les corrections)  
 Examen écrit et TP Noté du 27 novembre 2025  
 par Dominique Méry  
 4 décembre 2025

**Exercice 1** Soit le contrat suivant :

```
variables int x, int y, int z
requires P(x0, y0, z0)
ensures Q(x0, y0, z0, xf, yf, zf)
begin
  0 : x = 10 ∧ y = z+x ∧ z = 2·x
  y := z+x
  1 : x = 10 ∧ y = x+2·10
end
```

**Question 1.1** Compléter le module `ex25_1.tla` en définissant la précondition et la postcondition.

**Question 1.2** Compléter l'action `step0_1` et vérifier que le module est sans blocage.

**Question 1.3** Compléter le module en vérifiant que le contrat est correct c'est-à-dire que la propriété de correction partielle est satisfaite ou pas.

Listing 1 – labelex251

```
----- MODULE ex25_1 -----
EXTENDS Integers ,TLC
CONSTANTS x0,y0,z0 (* x0,y0,z0 are the initial values *), un
P(u,v,w) == u=10 /\ v=u+w /\ w=2*u
Q(u0,v0,w0,u,v,w) ==

-----
ASSUME P(x0,y0,z0)

-----
VARIABLES x,y,z,pc

-----
step0_1 == pc="0" /\ y'=z+x /\ pc'="1" /\ x'=x /\ z'=z
skip == UNCHANGED <<x,y,z,pc>>

-----
Init ==
Next == step0_1 \/ skip

-----
safety pc == pc="1" =>
=====
```

Listing 2 – labelex251

```
----- MODULE ex25_1 -----
EXTENDS Integers ,TLC
CONSTANTS x0,y0,z0 (* x0,y0,z0 are the initial values *), un
P(u,v,w) == u= 10 /\ v = w+u /\ w=2*u
Q(u0,v0,w0,u,v,w) == P(u0,v0,w0) /\ u = 10 /\ v = u+2*10
```

```

-----  

ASSUME P(x0,y0,z0)  

-----  

VARIABLES x,y,z,pc  

-----  

step0_1 ==  

    /\ pc="0"  

    /\ y'=z+x  

    /\ /\ pc ="1"  

    /\ UNCHANGED <<x,z>>  

skip == UNCHANGED <<x,y,z,pc>>  

-----  

Init == x=x0 /\ y=y0 /\ z=z0 /\ pc="0"  

Next == step0_1 /\ skip  

-----  

safetypc == pc="1" => Q(x0,y0,z0,x,y,z)
=====
```

**Exercice 2** Soit le contrat suivant :

```

variables int x,int y,int z
requires P(x0,y0,z0)
ensures Q(x0,y0,z0,xf,yf,zf)
begin
  0 : x = 1 /\ y = 3 /\ x+y = 12
  x := y+x
  1 : x = 567 /\ y = 34
end

```

**Question 2.1** Compléter le module `ex25_2.tla` en définissant la précondition et la postcondition.

**Question 2.2** Compléter l'action `step0_1` et vérifier que le module est sans blocage.

**Question 2.3** Compléter le module en vérifiant que le contrat est correct c'est-à-dire que la propriété de correction partielle est satisfaite ou pas.

Listing 3 – labelex251

```

----- MODULE ex25_2 -----
EXTENDS Integers ,TLC
CONSTANTS x0,y0 (* x0,y0 are the initial values *), un
pre(u,v) ==
post(u0,v0,u,v) ==

-----  

ASSUME pre(x0,y0)  

-----  

VARIABLES x,y,pc  

-----  

step0_1 ==  

skip == UNCHANGED <<x,y,pc>>
```

---

```

Init ==
Next == step0_1 \& skip

```

---

```
safetypc ==
```

---

Listing 4 – labelex251

```

----- MODULE ex25_2 -----
EXTENDS Integers ,TLC
CONSTANTS x0,y0,z0 (* x0,y0  are the initial values *), un
P(u,v,w) == u= 1 /\ v = 3
Q(u0,v0,w0,u,v,w) == P(u0,v0,w0) /\ u=567 /\ v=34

```

---

```
ASSUME P(x0,y0,z0)
```

---

```
VARIABLES x,y,z,pc
```

---

```
step0_1 ==
  /\ pc="0"
  /\ x '=y+x
  /\ pc ="1"
  /\ UNCHANGED <>y,z>>
```

```
skip == UNCHANGED <>x,y,z,pc>>
```

---

```
Init == x=x0 /\ y=y0 /\ z=z0 /\ pc="0"
Next == step0_1 \& skip
```

---

```
safetypc == pc="1" => Q(x0,y0,z0,x,y,z)
```

---

```
=====
```

**Exercice 3** Soit le programme C suivant :

Listing 5 – summation.c

```
#include <stdio.h>
```

```

int ffS(int n) {
    int ps = 0;
    int k = 0;
    int ok=k, ops = 0;
    while (k < n) {
        ok=k;ops=ps;
        k = ok + 1;
        ps = ops + k;
    };
    return ps;
}
int fS(int n) {
    int ps = 0;
```

```

int k = 0;
while (k < n) {
    ps = ps + k+1;
    k = k + 1;
}
return ps;
}

int main()
{
    for (int i = 0; i < 11; ++i){
        printf("Value_for_z=%d_is_%d\n", i ,fS(i));
    }
    return 0;
}

```

**Question 3.1** Ecrire les données en entrée de ce programme et ce qui est calculer.

**Question 3.2** Donner la liste des variables utiles pour exprimer le calcul à partir des valeurs.

**Question 3.3** Soit le code suivant extrait du programme ci-dessus :

Listing 6 – labelsummation.c

```

int ps = 0;
int k = 0;
int r;
inloop: while (k < x) {
    k = k + 1;
    ps = ps + k+1;
}
outloop: r = ps ;

```

Traduire ce code sous la forme d'actions suivant les étiquettes inloop et outloop.

**Question 3.4** Vérifier la correction partielle et l'absence des erreurs à l'exécution.

Listing 7 – labelsummation.c

```

----- MODULE summation -----
EXTENDS Integers ,TLC
CONSTANTS x0 ,r0 (* x0 is the input *), un
sum(u) == (u*(u+1)) \div 2
P(u,v) == u \geq 0
Q(u0,v0,u,v) == P(u0,v0) /\ v = sum(u)
-----
ASSUME P(x0,r0)
-----
VARIABLES x ,r ,k ,ps ,pc
-----
```

```

inloop == pc="inloop" /\ k < x /\ k'=k+1 /\ ps'=ps+k+1 /\ UNCHANGED <<x,r,pc>>
outloop == pc="inloop" /\ ~(k < x) /\ pc'="outloop" /\ UNCHANGED <<x,r,ps,k>>
final == pc="outloop" /\ r'=ps /\ pc'="done" /\ UNCHANGED <<x,ps,k>>
-----
Init == x=x0 /\ ps=0 /\ k=0 /\ r=un /\ pc="inloop"
Next == inloop \& outloop \& final \& UNCHANGED <<x,r,k,ps,pc>>
-----
safety pc==pc="done" => r==sum(x)
=====
```

**Exercice 4** Soit le programme suivant :

Listing 8 – average.c

```

int a(int n, int m) {
    int r;
    r = (m+n)/2;
    return r;
}
```

**Question 4.1** Ecrire une relation entre les valeurs initiales  $n_0$ ,  $m_0$  et  $r_0$  et les valeurs finales de  $n_f$ ,  $m_f$  et  $r_f$  des variables  $n, m, r$ . Traduire cette relation dans un module TLA<sup>+</sup>.

Listing 9 – average.tla

```

----- MODULE average -----
EXTENDS Integers , TLC
CONSTANTS n0,m0,r0 , un
-----
ave(x,y) == (x+y) \div 2
-----
ASSUME n0 \in Int /\ m0 \in Int
-----
VARIABLES n,m,r , pc
-----
Computing == pc="start"/\pc'="done"/\r'=ave(n,m) /\ UNCHANGED <<n,m>>
Skipping == UNCHANGED <<n,m,r , pc>>
-----
Init == n=n0 /\ m=m0 /\ r=un /\ pc="start"
Next == Computing \& Skipping
-----
safety pc == pc="done" => r = ave(n,m)
=====
```

**Question 4.2** L'exécution du programme summation.c conduit à une valeur inattendue pour certaines valeurs de  $n_0$  et de  $m_0$ . Analayser cette question avec l'outil TLA<sup>+</sup>.

Listing 10 – average.tla

```

----- MODULE averagebis -----
EXTENDS Integers , TLC
CONSTANTS n0,m0,r0 , un , maxi
mini== -maxi-1
-----
```

```

ave(x,y) == (x+y) \div 2
rte(x) == x#un => x \in mini..maxi
-----
ASSUME n0 \in Int /\ m0 \in Int
-----
VARIABLES n,m,r,pc
-----
Computing == pc=="start"/\pc'=="done"/\r'= ave(n,m) /\ UNCHANGED <<n,m>>
Skipping == UNCHANGED <<n,m,r,pc>>
-----
Init == n=n0 /\ m=m0 /\ r=un /\ pc=="start"
Next == Computing \ / Skipping
-----
safetypc == pc=="done" => r = ave(n,m)
safety rte == rte(n) /\ rte(m) /\ rte(r)
=====
```

**Exercice 5** Soient les deux annotations suivantes. Pour chacune de ces annotations, traduire les annotations en un module TLA<sup>+</sup> et vérifier ou non la correction de cette annotation.

### Question 5.1

On suppose que  $p$  est un nombre premier :

$$\begin{aligned} \ell_1 : & x = 2^p \wedge y = 2^{p+1} \wedge x \cdot y = 2^{2 \cdot p + 1} \\ & x := y + x + 2^x \\ \ell_2 : & x = 5 \cdot 2^p \wedge y = 2^{p+1} \end{aligned}$$

### Question 5.2

$$\begin{aligned} \ell_1 : & x = 1 \wedge y = 12 \\ & x := 2 \cdot y \\ \ell_2 : & x = 1 \wedge y = 24 \end{aligned}$$