

Cours Modélisation et vérification des systèmes informatiques
Exercices (avec les corrections)
Modélisation TLA⁺ (2)
par Dominique Méry
22 septembre 2021

Exercice 1 Soit le réseau de Petri de la figure ??.

Question 1.1 Déterminer les conditions initiales.

Question 1.2 Déterminer les relations modélisant les transitions.

Question 1.3 Valider les propriétés et les hypothèses que vous pourrez faire sur ce réseau de Petri.

◇— **Solution de l'exercice 1** _____

MODULE *petri8*

EXTENDS *Naturals, TLC*
VARIABLES *M*
CONSTANT *Places*

condition de t1

$$t1 \triangleq$$

$$\wedge M["p11"] \geq 1 \wedge M["p12"] \geq 1$$

$$\wedge M' = [[[M \text{ EXCEPT! } ["p11"] = @-1]$$

$$\text{EXCEPT! } ["p12"] = @-1]$$

$$\text{EXCEPT! } ["p21"] = @+1]$$

$$\text{EXCEPT! } ["p31"] = @+1]$$

$$t2 \triangleq$$

$$\wedge M["p21"] \geq 1$$

$$\wedge M' = [[[M \text{ EXCEPT! } ["p21"] = @-1]$$

$$\text{EXCEPT! } ["p22"] = @+2]$$

EXCEPT!["p11"] = @+1]

$t3 \triangleq$

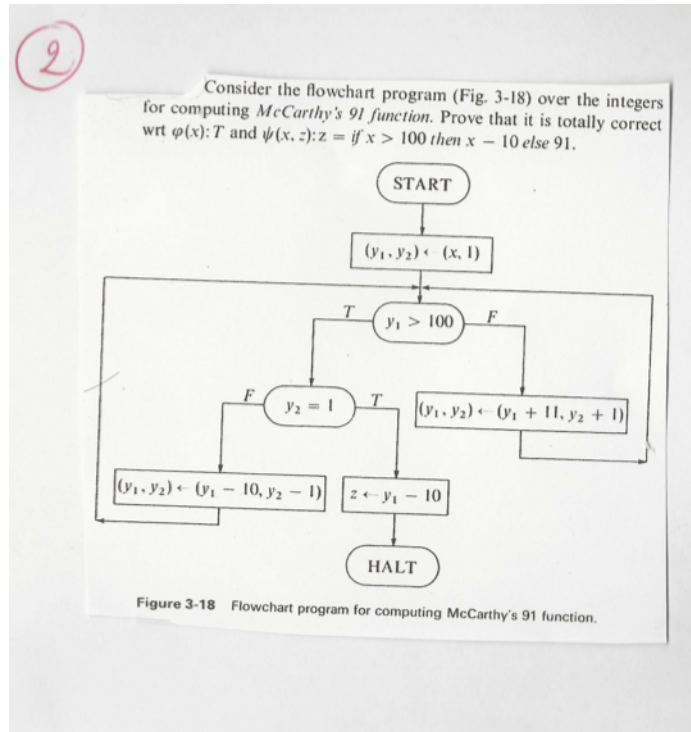
$\wedge M["p31"] \geq 1$
 $\wedge M' = [[M \text{ EXCEPT! } ["p31"] = @-1]$
 $\text{EXCEPT! } ["p33"] = @+1]$
 $\text{EXCEPT! } ["p12"] = @+1]$

$Init \triangleq M = [p \in \text{Places} \mapsto \text{IF } p \in \{"p11", "p12"\} \text{ THEN } 1 \text{ ELSE } 0]$
 $Next \triangleq t_1 \vee t_2 \vee t_3$

$Q1 \triangleq M["p22"] \neq 32$

$Q2 \triangleq M["p21"] = 1 \wedge M["p31"] = 1 \Rightarrow 2 \cdot M["p22"] = M["p33"]$

Fin 1



Exercice 2

Question 2.1 Construire un module TLA^+ modélisant les différents pas de calcul.

Question 2.2 Evaluer l'algorithme en posant des questions de sûreté suivantes :

1. l'algorithme est partiellement correct.
2. l'algorithme n'a pas d'erreurs à l'exécution.

◇ **Solution de l'exercice 2**

MODULE *maccarthy91*

EXTENDS *Naturals, TLC, Integers*
 CONSTANTS x, min, max
 VARIABLES y_1, y_2, z, c

$a \triangleq c = \text{"START"} \wedge y'_1 = x \wedge y'_2 = 1 \wedge c' = \text{"LOOP"} \wedge \text{UNCHANGED } \langle z \rangle$

$b \triangleq$
 $\wedge c = \text{"LOOP"} \wedge y_1 \leq 100$
 $\wedge y'_1 = y_1 + 11 \wedge y'_2 = y_2 + 1$
 $\wedge \text{UNCHANGED } \langle z, c \rangle$

$cc \triangleq$
 $\wedge c = \text{"LOOP"} \wedge y_1 > 100 \wedge y_2 \neq 1$
 $\wedge y'_1 = y_1 - 10 \wedge y'_2 = y_2 - 1$
 $\wedge \text{UNCHANGED } \langle z, c \rangle$

$d \triangleq$
 $\wedge c = \text{"LOOP"} \wedge y_1 > 100 \wedge y_2 = 1$
 $\wedge z' = y_1 - 10 \wedge c' = \text{"HALT"}$
 $\wedge \text{UNCHANGED } \langle y_1, y_2 \rangle$

$next \triangleq a \vee b \vee cc \vee d \vee \text{UNCHANGED } \langle y_1, y_2, z, c \rangle$

$init_1 \triangleq y_1 \in \textit{Int} \wedge y_2 \in \textit{Int} \wedge z \in \textit{Int} \wedge c = \text{"START"}$

$init \triangleq y_1 = 0 \wedge y_2 = 0 \wedge z = 0 \wedge c = \text{"START"}$

$Q1 \triangleq c \neq \text{"HALT"} \quad \text{c prned la valeur HALT}$

$Q2 \triangleq c = \text{"HALT"} \Rightarrow z = \text{IF } x > 100 \text{ THEN } x - 10 \text{ ELSE } 91$

$Qy_1 \triangleq min \leq y_1 \wedge y_1 \leq max$

$Question \triangleq Qy_1$

Fin 2

Exercice 3 Soit le schéma suivant définissant un calcul déterminant sir un nombre entier naturel est premier ou non.

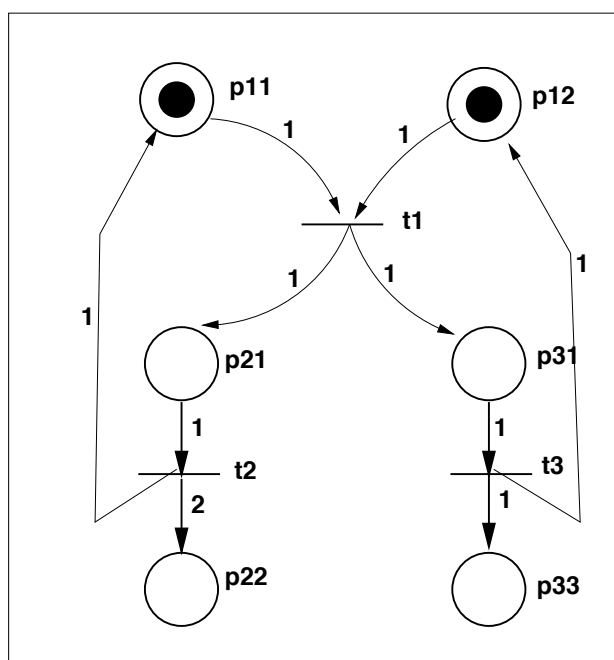
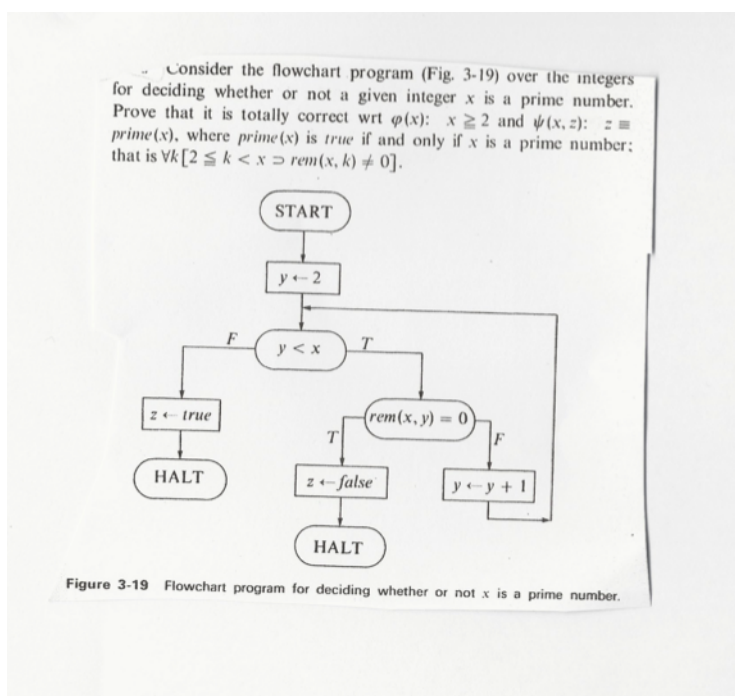


FIGURE 1 – Réseau de Petri



Question 3.1 *Ecrire un modèle TLA modélisant ce schéma de calcul.*

Question 3.2 *Ecrire un invariant à partir d'annotations que vous définirez après avoir défini des points de contrôle.*

Question 3.3 *Vérifier la correction partielle*

Question 3.4 *Vérifier l'absence d'erreurs à l'exécution.*

MODULE *Naturals*

A dummy module that defines the operators that are defined by the real *Naturals* module.

$Nat \triangleq \{ \}$
 $a+b \triangleq \{a, b\}$
 $a-b \triangleq \{a, b\}$
 $a \cdot b \triangleq \{a, b\}$
 $a^b \triangleq \{a, b\}$
 $a < b \triangleq a = b$
 $a > b \triangleq a = b$
 $a \leq b \triangleq a = b$
 $a \geq b \triangleq a = b$
 $a \% b \triangleq \{a, b\}$
 $a \div b \triangleq \{a, b\}$
 $a .. b \triangleq \{a, b\}$

MODULE *TLC*

LOCAL INSTANCE *Naturals*
 LOCAL INSTANCE *Sequences*

$Print(out, val) \triangleq val$
 $PrintT(out) \triangleq TRUE$
 $Assert(val, out) \triangleq \text{IF } val = TRUE \text{ THEN } TRUE$
 $ELSE \text{ CHOOSE } v : TRUE$
 $JavaTime \triangleq \text{CHOOSE } n : n \in Nat$
 $TLCGet(i) \triangleq \text{CHOOSE } n : TRUE$
 $TLCSet(i, v) \triangleq TRUE$

$d \rightarrow e \triangleq [x \in \{d\} \mapsto e]$
 $f @@@ g \triangleq [x \in (\text{DOMAIN } f) \cup (\text{DOMAIN } g) \mapsto$
 $\text{IF } x \in \text{DOMAIN } f \text{ THEN } f[x] \text{ ELSE } g[x]]$
 $Permutations(S) \triangleq$
 $\{f \in [S \rightarrow S] : \forall w \in S : \exists v \in S : f[v] = w\}$

In the following definition, we use *Op* as the formal parameter rather than *\prec* because TLC Version 1 can't handle infix formal parameters.

$SortSeq(s, Op(_, _)) \triangleq$
 $LET \text{Perm} \triangleq \text{CHOOSE } p \in Permutations(1 .. Len(s)) :$
 $\forall i, j \in 1..Len(s) :$
 $(i < j) \Rightarrow Op(s[p[i]], s[p[j]]) \vee (s[p[i]] = s[p[j]])$
 $IN [i \in 1..Len(s) \mapsto s[Perm[i]]]$

$RandomElement(s) \triangleq \text{CHOOSE } x \in s : TRUE$

$Any \triangleq \text{CHOOSE } x : TRUE$

$ToString(v) \triangleq (\text{CHOOSE } x \in [a : v, b : STRING] : TRUE).b$

$TLCEval(v) \triangleq v$

FIGURE 2 – Modules *Naturals.tla* et *TLC.tla*

Exercice 4 Le module *truc* permet de résoudre un problème très classique en informatique : trouver un chemin entre un sommet input et des sommets output supposés être des sommets de sortie.

Question 4.1 Pour trouver un chemin de input à l'un des sommets de output, il faut poser une question de sûreté à notre système de vérification. Donner une question de sûreté à poser permettant de trouver un chemin de input vers un sommet de output.

Question 4.2 On désire utiliser cette technique pour trouver un chemin dans un labyrinthe. Un labyrinthe est représenté par une matrice carrée de taille n . On définit ensuite pour chaque élément $\langle\langle i, j \rangle\rangle$ de la matrice les voisins communiquant à l'aide de la fonction *lab* qui associe à $\langle\langle i, j \rangle\rangle$ les éléments qui peuvent être atteints en un coup. Par exemple, le mouvement possible à partir de $\langle\langle 1, 1 \rangle\rangle$ est $\langle\langle 2, 1 \rangle\rangle$, ou le mouvement possible à partir de $\langle\langle 2, 1 \rangle\rangle$ est $\langle\langle 2, 2 \rangle\rangle$ ou $\langle\langle 1, 1 \rangle\rangle$, ou le mouvement possible à partir de $\langle\langle 2, 2 \rangle\rangle$ est $\langle\langle 2, 3 \rangle\rangle$ ou $\langle\langle 3, 2 \rangle\rangle$ ou $\langle\langle 2, 1 \rangle\rangle$, ...

```
lab == [<<x,y>> \in (nodes \X nodes) |->
      IF x=1 /\ y=1 THEN {<<2,1>>} ELSE
      IF x=2 /\ y=1 THEN {<<2,2>>}
      IF x=1 /\ y=2 THEN {} ELSE
      IF x=2 /\ y=2 THEN {<<3,2>>, <<2,3>>} ELSE
      ELSE {}

      ]
```

Modifier le module *truc* pour traiter ce problème et donner la question à poser pour trouver une sortie.

MODULE *truc*

EXTENDS *Integers, TLC*
 VARIABLES p
 CONSTANTS *input, output*

$n \triangleq 10$
 $nodes \triangleq 1..n$
 $l \triangleq [i \in 1..n \mapsto \text{IF } i = 1 \text{ THEN } \{4, 5\} \text{ ELSE}$
 $\text{IF } i = 2 \text{ THEN } \{6, 7, 10\} \text{ ELSE}$
 $\text{IF } i = 4 \text{ THEN } \{7, 8\} \text{ ELSE}$
 $\text{IF } i = 5 \text{ THEN } \{\} \text{ ELSE}$
 $\text{IF } i = 6 \text{ THEN } \{4\} \text{ ELSE}$
 $\text{IF } i = 7 \text{ THEN } \{5\} \text{ ELSE}$
 $\text{IF } i = 8 \text{ THEN } \{5, 2\} \text{ ELSE}$
 $\{\}$
 $]$

$Init \triangleq p = 1$
 $M(i) \triangleq \wedge i \in l[p]$
 $\wedge p' = i$
 $Next \triangleq \exists i \in 1..n : M(i)$

◇ **Solution de l'exercice 4**

MODULE *labyrinthe*

EXTENDS *Integers, TLC*

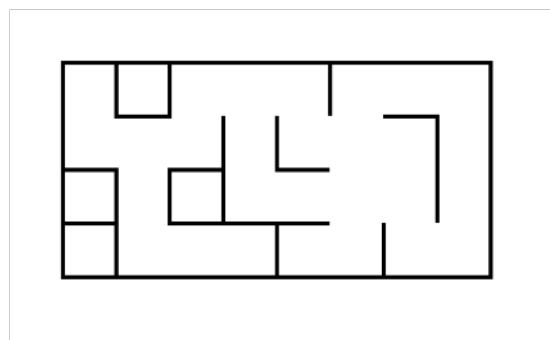
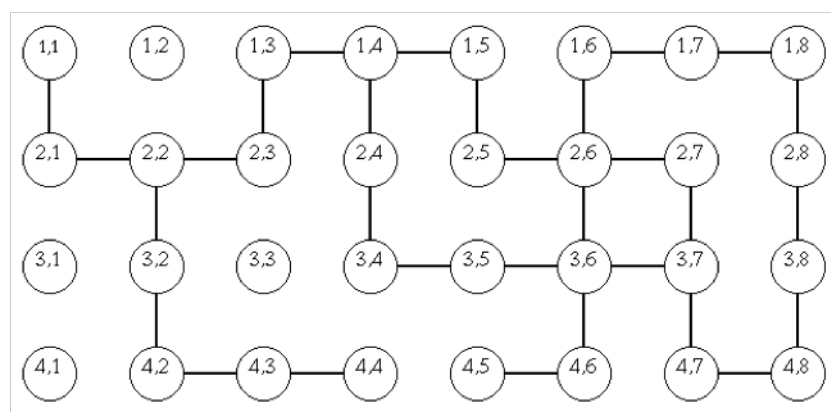


FIGURE 3 – Labyrinthe

VARIABLES p
 CONSTANTS $input, output$

$n \triangleq 10$
 $nodes \triangleq 1..n$
 $l \triangleq [i \in 1..n \mapsto \text{IF } i = 1 \text{ THEN } \{4, 5\} \text{ ELSE}$
 $\text{IF } i = 2 \text{ THEN } \{6, 7, 10\} \text{ ELSE}$
 $\text{IF } i = 4 \text{ THEN } \{7, 8\} \text{ ELSE}$
 $\text{IF } i = 5 \text{ THEN } \{\} \text{ ELSE}$
 $\text{IF } i = 6 \text{ THEN } \{4\} \text{ ELSE}$
 $\text{IF } i = 7 \text{ THEN } \{5\} \text{ ELSE}$
 $\text{IF } i = 8 \text{ THEN } \{5, 2\} \text{ ELSE}$
 $\{\}$
]

 $lab \triangleq [\langle x, y \rangle \in (nodes \times nodes) \mapsto$
 $\text{IF } x = 1 \wedge y = 1 \text{ THEN } \{\langle 1, 2 \rangle\} \text{ ELSE}$
 $\text{IF } x = 1 \wedge y = 2 \text{ THEN } \{\langle 1, 3 \rangle, \langle 2, 2 \rangle\}$
 $\text{ELSE } \{\}$
]

 $Init \triangleq p = 1$
 $M(i) \triangleq \wedge i \in l[p]$
 $\wedge p' = i$
 $Next \triangleq \exists i \in 1..n : M(i)$

 $Initlab \triangleq p = \langle 1, 1 \rangle$
 $ML(q) \triangleq \wedge q \in lab[p]$
 $\wedge p' = q$
 $Nextlab \triangleq \exists q \in nodes \times nodes : ML(q)$

 $Sortie \triangleq p \notin output$

Fin 4

Cours Modélisation et vérification des systèmes informatiques
 Exercices (avec les corrections)
 Utilisation d'un environnement de vérification Frama-c (I)
 par Dominique Méry
 22 septembre 2021

Exercice 5 Soit le petit programme suivant dans un fichier *ex1.c*

```

void swap1(int a, int b) {
    int x = a;
    int y = b;
    //@ assert x == a && y == b;
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
    //@ assert x == b && y == a;
}

```


Question 5.1 Utiliser l'outil `frama-c-gui` avec la commande `$frama-c-gui ex1.c` et cliquer sur le lien `ex1.c` apparaissant sur la gauche. A partir du fichier source, une fenêtre est créée et vous découvrez le texte du fichier.

Question 5.2 Cliquer à droite sur le mot-clé `assert` et cliquer sur *Prove annotation by WP*. Les boutons deviennent vert.

Question 5.3

```
void swap2(int a, int b) {
    int x = a;
    int y = b;
    //@ assert x == a && y == b;
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
    //@ assert x == a && y == a;
}
```

Répétez les mêmes suites d'opérations mais avec le programme suivant dans `ex2.c`.

Question 5.4 Soit le nouvel algorithme avec un contrat qui établit ce que l'on attend de cet algorithme

```
/*@
requires \valid(a);
requires \valid(b);
ensures P: *a == \old(*b);
ensures Q: *b == \old(*a);
*/
void swap3(int
           *a, int *b) {
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
}
```

Recommencer les opérations précédentes et observer ce qui a été utilisé comme outils de preuve.

◊— **Solution de l'exercice 5**

Les solutions aux questions sont dans l'énoncé lui-même et il est uniquement indispensable de faire vraiment l'exercice...

Listing 1 – schema de contrat

```
// returns the absolute value of x
int abs ( int x ) {
if ( x >=0 ) return x ;
return -x ; }
```

Listing 2 – schema decontrat

```
// returns the absolute value of x
int abs ( int x ) {
if ( x >=0 ) return x ;
return -x ; }
```

Listing 3 – schema de contrat

```
void swap1(int a, int b) {
    int x = a;
    int y = b;
    //@ assert x == a && y == b;
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
    //@ assert x == b && y == a;
}
```

Listing 4 – schema de contrat

```
/*@ requires a == b;
*/
void swap1(int a, int b) {
    int x = a;
    int y = b;
    //@ assert x == a && y == b;
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
    //@ assert x == a && y == a;
}
```

Listing 5 – schema de contrat

```
void swap2(int a, int b) {
    int x = a;
    int y = b;
    //@ assert x == a && y == b;
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
    //@ assert x == a && y == a;
}
```

Listing 6 – schema de contrat

```
#include <limits.h>
/*@
    requires INT_MIN < a+b;
    requires INT_MAX > a+b;
    requires INT_MIN < a-b;
    requires INT_MAX > a-b;
*/
```

```
void swap3(int a, int b) {
    int x = a;
    int y = b;
    //@ assert x == a && y == b;
```

```

    int tmp;
    x = x+y;
    y = x-y;
    x = x-y;

    //@ assert x == b && y == a;
}

```

Listing 7 – schema de contrat

```

#include <limits.h>
/*@
    requires INT_MIN <  a+b;
    requires INT_MAX > a+b;
    requires INT_MIN <  a-b;
    requires INT_MAX > a-b;
*/

```

```

void swap3(int a, int b) {
    int x = a;
    int y = b;
    int tmp;
    x = x+y;
    y = x-y;
    x = x-y;
}

```

Listing 8 – schema de contrat

```

/*@
    requires \valid(a);
    requires \valid(b);
    ensures P: *a == \old(*b);
    ensures Q: *b == \old(*a);
*/
void swap(int
        *a, int *b) {
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
}

```

Fin 5

Exercice 6

```

void test(int x1, int x2) {

    int y1,y2,z1,z2;
    y1 = x1;
    //@ ....;
    y2 = x2;
    //@ ....;
    y1 = y1 - (y1 - y2);
    y2 = x1 +x2 - y1;
    z1 = y1;
}

```

```

    z2 = y2;
    //@ ....;

```

```

}

```

Annoter le petit algorithme et le vérifier ensuite avec l'outil.

◊— Solution de l'exercice 6

La solution est fondée sur la redéfinition de la fonction de soustraction au sens de *on ne peut pas enlever des bonbons quand il ny en a plus...* On doit ajouter des lemmes en plus pour pouvoir prouver des propriétés invariantes. Il faut tester cet algorithme avec l'option -wp de frama-c.

Listing 9 – schema de contrat

```

//@ lemma exun: \forall integer a,b; a <= b && a >= 0 && b >= 0 ==> b-a >=0 && b-a
//@ lemma ex: \forall integer a,b,c; a <= b && a >= 0 && b >= 0 && c >= 0
&& c >= b ==> b-a >=0 && b-a <= b && b-a <= c;

```

```

#include <limits.h>
/*@ requires x <= INT_MAX;requires x <= INT_MAX;
    requires y <= INT_MAX;requires y <= INT_MAX;
    requires x-y <= INT_MAX;requires x-y <= INT_MAX;
    assigns \nothing;
    ensures \result <= x;
*/
int f(int x,int y) {
    if (x<=y) {return 0;}
    else {
        return x-y;}
}

/*@ requires x1 >= 0 && x2 >= 0;
    requires x1 <= x2;
    requires x1 <= INT_MAX;requires x1 <= INT_MAX;
    requires x2 <= INT_MAX;requires x2 <= INT_MAX;
*/

void test(int x1, int x2) {

    int y1,y2,z1,z2;
    y1 = x1;

    y2 = x2;
    y1 = f(y1,f(y1,y2));
    //@ y1 <= x1 && y1 <= x2;
    y2 = x1 +x2 - y1;
    z1 = y1;
    //@ z1 <= x1 ;
    z2 = y2;
    //@ z1 <= x1 && z1 <= x2;

}

```

Fin 6

Exercice 7 Etudier la correction des algorithmes suivants :

Question 7.1 */*@*
requires 0 <= n;
*ensures \result == n * n;*
**/*
int f(**int** n) {
 int i = 0;
 /@ assert i=0*
 int s = 0;
 /@ loop invariant i * i == s && i <= n;*
 *@ loop assigns i, s; */*
 while (i < n) {
 i++;
 s += 2 * i - 1;
 };
 return s;
}

◇— **Solution de l'exercice 7**

Listing 10 – schema de contrat

```
/*@
requires 0 <= n;
ensures \result == n * n;
*/
int f(int n) {
    int i = 0;
    //@ assert i==0;
    int s = 0;
    /*@ loop invariant i * i == s && i <= n;
    @ loop assigns i, s; */
    while (i < n) {
        i++;
        s += 2 * i - 1;
    };
    return s;
}
```

Listing 11 – schema de contrat

```
/*@
requires 0 <= n;
ensures \result == n * n;
*/
int f(int n) {
    int i = 0;
    int s = 0;
    /*@ loop invariant i <= n;
    @ loop assigns i, s; */
    while (i < n) {
        i++;
        s += 2 * i - 1;
    };
    return s;
}
```

Fin 7

Listing 12 – division

```
Exercice 8 Question 8.1 /*@ requires a >= 0 && b >= 0;
    ensures 0 <= \result;
    ensures \result < b;
    ensures \exists integer i ; a == i * b + \result;
*/
int rem(int a, int b) {
    int r = a, q = 0 ;
    /*@
        loop invariant
        a == q * b + r &&
        r >= 0;
        loop assigns r,q;
    */
    while (r >= b) { r = r - b; q = q + 1; };
    return r;
}
```

Listing 13 – division

```
Question 8.2 /*@ requires x >= 0 ;
    @*/
int f(int x){
    return x+1;
}
```

Listing 14 – division

```
Question 8.3 /*@ requires *x == 10 && *z==2**x && *y == *z+*x;
    ensures *x == 1 && *y == *x+2*10;
*/
int f(int *x, int *y, int *z){
    y = *x+*z;
    return 0;
}
```