Validation des systèmes embarqués

Durée : 3h
Documents : tous documents autorisés

Veuillez respecter les notations introduites dans l'énoncé.
Les parties I et II sont indépendantes, veuillez les rédiger sur des copies séparées.
Des explications pertinentes en français seront **très** appréciées, et il est indispensable de **justifier** vos réponses.

I - Modélisation et interprétation abstraite (11 points)

Exercice 1 : Expression de propriétés et questions de validation automatique (3 points)

On s'intéresse ici à l'expression de propriétés temporelles dans des programmes impératifs séquentiels. La figure 1 donne un exemple de programme C qui utilise l'instruction `assert P`. Cette instruction a pour effet de stopper le programme (avec un message d'erreur compréhensible) si l'exécution atteint ce point de programme avec la propriété `P` fausse.

`progassert.c :`

```
1. #include <stdio.h>
2. #include <assert.h>
3. int main () {
4.   int i ;
5.
6.   printf("Donnez un entier : \n");
7.   scanf ("%d", &i);
8.   assert (i==42);
9. }
```

Si l'on tape par exemple 13 lors de l'exécution du programme, on a droit au message :

```
Donnez un entier :
13
a.out: progassert.c:8: main:
Assertion ‘i==42’ failed.
```

Figure 1: Exemple de programme C utilisant `assert`

▷ **Question 1 (1.5 points) :**

On considère un programme séquentiel écrit en C et manipulant une variable entière `x`. On désire exprimer que, lors des passages successifs du programme à un certain point `XX`, les valeurs de `x` sont strictement croissantes (le point `XX` peut être par exemple à l'entrée d'une boucle).
— Est-ce une propriété de sûreté (safety) ou de vivacité (liveness) ? Justifiez.
— Expliquez de manière générale comment modifier le programme en utilisant `assert` pour que, lors des exécutions, on obtienne un message d'erreur si la propriété ci-dessus est violée.

▷ **Question 2 (1.5 points) :**

— Donnez un exemple de programme non trivial, et sa transformation.
— Dessinez le graphe de contrôle de votre programme, pour montrer comment une instruction `assert` donne des états d'erreur.
— Pensez-vous qu'une abstraction de type *signes* ou *intervalles* puisse permettre de vérifier cette propriété de croissance d'une variable ? Justifiez.

Exercice 2 : Interprétation abstraite (8 points)

Dans toute cette partie, on considère des programmes à deux variables numériques `x` et `y`, comme celui de la figure 2.

```
get (b) ; -- une valeur booléenne inconnue
if (b) then
   x := 1 ; y := 1 ;
else
   x := 80 ; y := 20 ;
end if ;
while true loop
   x := x - 30 ;
   y := y + 30 ;
end loop
```

Figure 2: Exemple de programmes à deux variables numériques

On s'intéresse à des abstractions des ensembles de couples de valeurs de `x` et `y` sous la forme :

$$\begin{cases} m_1 \leq x - y \leq m_2 \\ p_1 \leq x + y \leq p_2 \\ m_1, p_1 \in \mathbb{Z} \cup \{-\infty\} \\ m_2, p_2 \in \mathbb{Z} \cup \{+\infty\} \end{cases}$$

La figure 3 est une illustration graphique des domaines du plan représentés par des abstractions de cette forme.

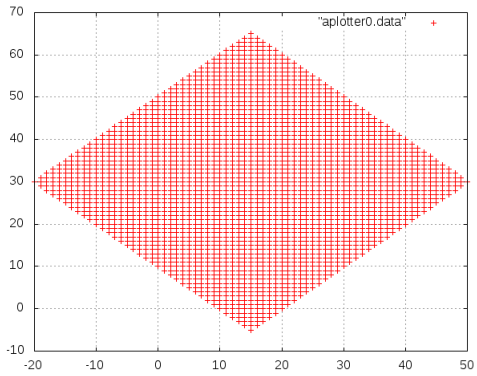


Figure 3: Domaine représenté par $m_1 = -50, m_2 = 20, p_1 = 10, p_2 = 80$

▷ **Question 3 (1.5 points) :**

— Donner quelques exemples d'ensembles de couples, et leur abstraction sous la forme (m_1, m_2, p_1, p_2) définie ci-dessus.
— Définir un ordre sur ces valeurs abstraites : $(m_1, m_2, p_1, p_2) \leq^\sharp (m'_1, m'_2, p'_1, p'_2)$.
— Définir l'union de deux valeurs abstraites (m_1, m_2, p_1, p_2) et (m'_1, m'_2, p'_1, p'_2).

On étudie les calculs abstraits associés à ce type d'information. On attache à un état de programme un quadruplet (m_1, m_2, p_1, p_2) signifiant que l'ensemble des valeurs possibles des variables x et y en cet état est inclus dans l'ensemble décrit par $m_1 \leq x - y \leq m_2 \ \wedge \ p_1 \leq x + y \leq p_2$.

On s'intéresse maintenant à la propagation de ce type d'information à travers des affectations à x ou y, et à travers des conditions sur x et y.

▷ **Question 4 (3 points) :**

Décrire l'effet des affectations suivantes sur un quadruplet (m_1, m_2, p_1, p_2), c'est-à-dire donner un quadruplet (m'_1, m'_2, p'_1, p'_2) tel que, si $m_1 \leq x-y \leq m_2 \wedge p_1 \leq x+y \leq p_2$ avant l'affectation, alors $m'_1 \leq x - y \leq m'_2 \ \wedge \ p'_1 \leq x + y \leq p'_2$ après l'affectation. Parmi toutes les solutions (dont $(-\infty, +\infty, -\infty, +\infty)$) on donnera bien sûr la *plus forte post-condition*, c'est-à-dire le quadruplet qui décrit la plus petite zone du plan réel. `k, k'` sont des constantes entières.

- `{ x := k ; y := k' ; }`
- `x := x + k`
- `y := y + k`
- `x := -x`

▷ **Question 5 (2 points) :**

Même question pour l'effet des conditions : si l'on sait que $m_1 \leq x-y \leq m_2 \wedge p_1 \leq x+y \leq p_2$, et que d'autre part une condition booléenne $C(x, y)$ est vraie, il est possible que cela renforce la connaissance sur x et y, ou bien que ce soit contradictoire, etc. Pour chacune des conditions ci-dessous, indiquer si la condition est compatible avec l'information donnée par (m_1, m_2, p_1, p_2), et dans ce cas donner un quadruplet (m'_1, m'_2, p'_1, p'_2) qui exprime la nouvelle information sur x et y :

- `x = y`
- `x` $\geq$ `y`

▷ **Question 6 (1.5 points) :**

Associer des valeurs abstraites aux états de contrôle du programme de la figure 2 (dessiner le graphe de contrôle en séparant bien les transitions qui portent des affectations, et les transitions qui portent des conditions). Discutez la terminaison de l'analyse.

II - Model-checking (11 points)

Exercice 3 : SMV (5 points)

Le code en SMV de la figure 4 représente une machine à états.

▷ **Question 7 (5 points) :**

- Quels sont les états atteignables ? Donner le graphe d'états de cette machine.
- Donner la fonction de transition.
- Quels états satisfont les propriétés CTL suivantes :
  - `AX S[2]`
  - `EX S[2]`
- Les propriétés suivantes sont-elles satisfaites dans l'état initial ?
  - `AG ~S[1]| S[2]|~(S[3]^S[4])` (le symbole `^` représente le ou exclusif)
  - `A (S[4]-> S[4] U S[3])`

```
module main (I1, I2, Y)
 {
/* E/S */
 INPUT I1,I2: BOOLEAN;
 OUTPUT Y: BOOLEAN;
/* SIGNAL */

S: ARRAY 1..4 OF BOOLEAN;

init(S):= [0000];
next(S):=
  SWITCH(S){
    [0000]: CASE{
            I1&I2:[1111];
            I1&~I2:[1000];
            ~I1&I2:[0001];
            ~I1&~I2: [0000];
            };
    [0001]: CASE{
            I1&I2:[0011];
            I1&~I2:[0001];
            ~I1&I2:[0010];
            ~I1&~I2: [0000];
            };
    [0010]: CASE{
            I1&I2:[0011];
            I1&~I2:[0010];
            ~I1&I2:[0100];
            ~I1&~I2: [0000];
            };
    [0011]: CASE{
            I1&I2:[1111];
            I1^I2:[0011];
            ~I1&~I2: [0000];
            };
    [0100]: CASE{
            I1&I2:[1100];
            I1&~I2:[0010];
            ~I1&I2:[0100];
            ~I1&~I2: [0000];
            };
    [1000]: CASE{
            I1&I2:[1100];
            I1&~I2:[0100];
            ~I1&I2:[1100];
            ~I1&~I2: [0000];
            };
    [1001]: CASE{
            I1|I2:[1100];
            ~I1&~I2: [0000];
            };
    [1010]: CASE{
            I1|I2:[1001];
            ~I1&~I2: [0000];
            };
    [1100]: CASE{
            I1^I2:[1100];
            ~I1&~I2: [0000];
            };
    [1111]: CASE{
            I1|I2:[1111];
            ~I1&~I2: [0000];
            };
   };
  Y:= ~(S=[1111])&~(S=[0000])?1:0;
 }
```

Figure 4: Exemple de code SMV

Exercice 4 : Protocole du bit alterné (6 points)

Le protocole du bit alterné est un ensemble de règles permettant d'envoyer de façon fiable des messages à un récepteur en utilisant des voies de transmission qui peuvent perdre les messages. Les messages sont numérotés pour pouvoir être identifiés, mais la numérotation est cyclique avec un cycle 0,1 (d'où le nom du protocole). Pour signifier qu'il a bien reçu un message, le récepteur envoie à l'émetteur un accusé de réception qui porte le numéro du message dont il confirme la réception. Pour pallier une perte éventuelle d'un message ou d'un accusé, l'émetteur utilise un timeout qui est armé pour une durée donnée, à chaque émission d'un message. Si le timeout se déclenche avant que l'accusé de réception ne soit reçu par l'émetteur, alors ce dernier renvoie le message. La réception par le récepteur d'un message déjà reçu provoque de la part de celui-ci une réémission de l'accusé de réception correspondant.

La figure 6 donne une implémentation de ce protocole en PROMELA. Seuls les numéros des messages et les accusés de réception sont envoyés.

La figure 5 représente l'automate de l'émetteur. La figure 8 montre une trace d'exécution de ce protocole.

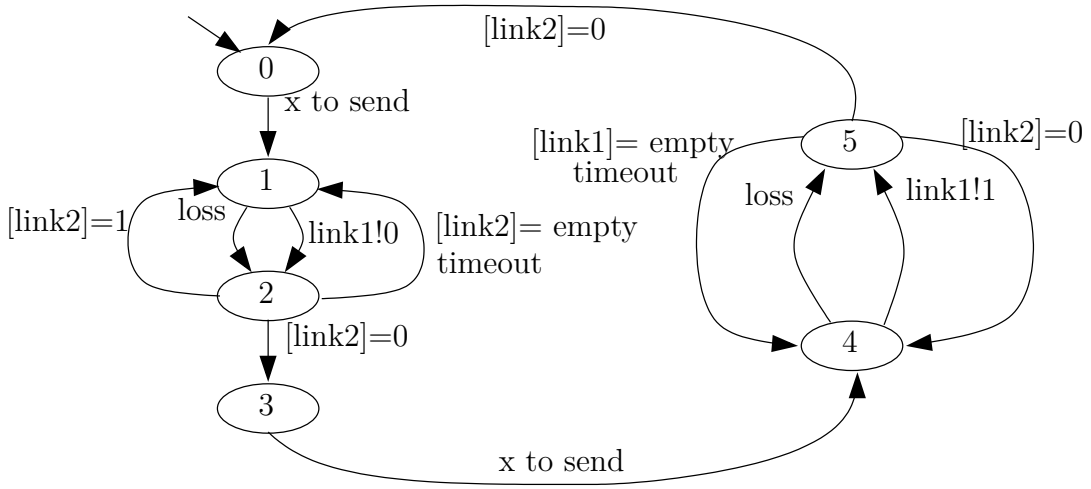


Figure 5: Automate de l'émetteur

▷ **Question 8 (1.5 points) :**

Donner l'automate du récepteur.

▷ **Question 9 (2.5 points) :**

La figure 8 donne une trace d'exécution possible de ce protocole et la figure 7 montre le sequence chart associé à cette trace. Que contiennent les cannaux de communication `link1` et `link2` apres l'étape `11` de la trace d'exécution ? Que se passe-t-il si l'on supprime une transition perte `(loss)` ? Donner une trace d'exécution faisant apparaître le problème.

▷ **Question 10 (2 points) :**

Ecrire en LTL une propriété montrant que le récepteur utilise toujours le même bit tant que l'émetteur n'a pas correctement acquitté le message.

```
/* canaux de transmission, sans perte */
chan link1 = [1] of {bool} ;
chan link2 = [1] of {bool} ;


active proctype Emitter () {
   byte state = 0 ;
   byte x = 0 ;

   do

     :: atomic { state == 0 -> printf ("x to send %d\n",0) ;
                               state = 1  ;}
     :: atomic { state == 1 -> printf("put 0 on link1\n") ;
                               link1!0 ; state = 2 ; }
     :: atomic { state == 1 -> printf ("link1 loses\n") ;
                               state = 2 ; }  /* perte */
     :: atomic { state == 2 && !link2?[x] ->
                               printf ("timeout Emitter\n") ;
                               state = 1 ; }  /* timeout */
     :: atomic { state == 2 && link2?[0] ->
                               printf ("get 0 from link2\n") ;
                               link2?x ; state = 3 ; }
     :: atomic { state == 2 && link2?[1] ->
                               printf ("get 1 from link2\n") ;
                               link2?x ; state = 1 ; }
     :: atomic { state == 3 -> printf ("x to send %d\n",1) ;
                               state = 4  ;}
     :: atomic { state == 4 -> printf("put 1 on link1\n") ;
                               link1!1 ; state = 5 ; }
     :: atomic { state == 4 -> printf ("link1 loses\n") ;
                               state = 5 ; }  /* perte */
     :: atomic { state == 5 && !link2?[x] ->
                               printf ("timeout Emitter\n") ;
                               state = 4 ; }  /* timeout */
     :: atomic { state == 5 && link2?[0] ->
                               printf ("get 0 from link2\n") ;
                               link2?x ;
                               state = 4 ; }
     :: atomic { state == 5 && link2?[1] ->
                               printf ("get 1 from link2\n") ;
                               link2?x ; state = 0 ; }
   od
}
```

```
active proctype Receiver () {

    byte state = 0 ;
    byte x     = 0 ;

   do
     :: atomic { state == 1 -> printf ("y received %d\n",0) ;
                               state = 2  ;}
     :: atomic { state == 2 -> printf("put 0 on link2\n") ;
                               link2!0 ; state = 3 ; }
     :: atomic { state == 2 -> printf ("link2 loses\n") ;
                               state = 3 ; }  /* perte */
     :: atomic { state == 3 && !link1?[x] ->
                               printf ("timeout Receiver\n") ;
                               state = 2 ; }   /* timeout */
     :: atomic { state == 3 && link1?[0] ->
                               printf("get 0 from link1\n") ;
                               link1?x ;state = 2 ; }
     :: atomic { state == 3 && link1?[1] ->
                               printf("get 1 from link1\n") ;
                               link1?x ; state = 4  ; }
     :: atomic { state == 4 -> printf ("y received %d\n",1) ;
                               state = 5  ;}
     :: atomic { state == 5 -> printf("put 1 on link2\n") ;
                               link2!1 ; state = 0 ; }
     :: atomic { state == 5 -> printf ("link2 loses\n") ;
                               state = 0 ; }  /* perte */
     :: atomic { state == 0 && !link1?[x] ->
                               printf ("timeout Receiver\n") ;
                               state = 5 ; }   /* timeout */
     :: atomic { state == 0 && link1?[0] ->
                               printf("get 0 from link1\n") ;
                               link1?x  ;   state = 1 ; }
     :: atomic { state == 0 && link1?[1] ->
                               printf("get 1 from link1\n") ;
                               link1?x  ; state = 5 ; }
   od
}
```

Figure 6: Code du protocole du bit alterné en PROMELA. Une séquence regroupant plusieurs instructions à l'aide de la primitive `atomic` sera exécutée comme une seule instruction, sans entrelacement avec les autres processus.

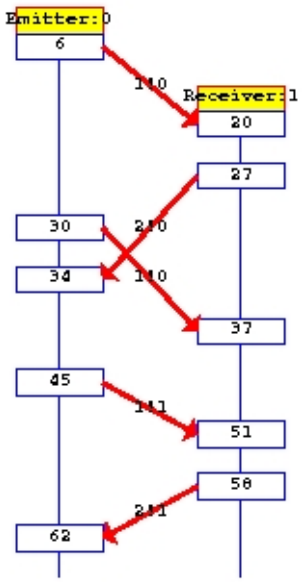


Figure 7: Sequence chart

```
Starting Emitter with pid 0
  0: proc  - (:root:) creates proc  0 (Emitter)
Starting Receiver with pid 1
  0: proc  - (:root:) creates proc  1 (Receiver)
  1: (Emitter) (state 55) [((state==0))]
x to send 0
  2: (Emitter) (state 2) [printf('x to send %d\\n',0)]
  3: (Emitter) (state 3) [state = 1]
  4: (Emitter) (state 55) [((state==1))]
put 0 on link1
  5: (Emitter) (state 6) [printf('put 0 on link1\\n')]
  6: (Emitter) (state -) [values: 1!0]
  6: (Emitter) (state 7) [link1!0]
  7: (Emitter) (state 8) [state = 2]
  8: (Emitter) (state 55) [(((state==2)&&!(link2?[x])))]
timeout Emitter
  9: (Emitter) (state 15) [printf('timeout Emitter\\n')]
 10: (Emitter) (state 16) [state = 1]
 11: (Emitter) (state 55) [((state==1))]
link1 loses
 12: (Emitter) (state 11) [printf('link1 loses\\n')]
 13: (Emitter) (state 12) [state = 2]
 14: (Emitter) (state 55) [(((state==2)&&!(link2?[x])))]
timeout Emitter
 15: (Emitter) (state 15) [printf('timeout Emitter\\n')]
 16: (Emitter) (state 16) [state = 1]
 17: (Emitter) (state 55) [((state==1))]
put 0 on link1
 18: (Emitter) (state 6) [printf('put 0 on link1\\n')]
 19: (Receiver)(state -) [1?0]
 19: (Receiver)(state 55) [(((state==0)&&link1?[0]))]
 20: (Receiver)(state -) [values: 1?0]
 20: (Receiver)(state 46) [link1?x]
get 0 from link1
 21: (Receiver)(state 47) [printf('get 0 from link1\\n')]
 22: (Receiver)(state 48) [state = 1]
 23: (Receiver)(state 55) [((state==1))]
y received 0
 24: (Receiver)(state 2) [printf('y received %d\\n',0)]
 25: (Receiver)(state 3) [state = 2]
 26: (Receiver)(state 55) [((state==2))]
 27: (Receiver)(state -) [values: 2!0]
 27: (Receiver)(state 6) [link2!0]
put 0 on link2
 28: (Receiver)(state 7) [printf('put 0 on link2\\n')]
 29: (Receiver)(state 8) [state = 3]
 30: (Emitter) (state -) [values: 1!0]
 30: (Emitter) (state 7) [link1!0]
 31: (Emitter) (state 8) [state = 2]
 32: (Emitter) (state -) [2?0]
 32: (Emitter) (state 55) [(((state==2)&&link2?[0]))]
get 0 from link2
 33: (Emitter) (state 19) [printf('get 0 from link2\\n')]
 34: (Emitter) (state -) [values: 2?0]
 34: (Emitter) (state 20) [link2?x]
 35: (Emitter) (state 21) [state = 3]
 36: (Receiver)(state -) [1?0]
 36: (Receiver)(state 55) [(((state==3)&&link1?[0]))]
 37: (Receiver)(state -) [values: 1?0]
 37: (Receiver)(state 19) [link1?x]
get 0 from link1
 38: (Receiver)(state 20) [printf('get 0 from link1\\n')]
 39: (Receiver)(state 21) [state = 2]
 40: (Emitter) (state 55) [((state==3))]
x to send 1
 41: (Emitter) (state 29) [printf('x to send %d\\n',1)]
 42: (Emitter) (state 30) [state = 4]
 43: (Emitter) (state 55) [((state==4))]
put 1 on link1
 44: (Emitter) (state 33) [printf('put 1 on link1\\n')]
 45: (Emitter) (state -) [values: 1!1]
 45: (Emitter) (state 34) [link1!1]
 46: (Emitter) (state 35) [state = 5]
 47: (Receiver)(state 55) [((state==2))]
link2 loses
 48: (Receiver)(state 11) [printf('link2 loses\\n')]
 49: (Receiver)(state 12) [state = 3]
 50: (Receiver)(state -) [1?1]
 50: (Receiver)(state 55) [(((state==3)&&link1?[1]))]
 51: (Receiver)(state -) [values: 1?1]
 51: (Receiver)(state 24) [link1?x]
get 1 from link1
 52: (Receiver)(state 25) [printf('get 1 from link1\\n')]
 53: (Receiver)(state 26) [state = 4]
 54: (Receiver)(state 55) [((state==4))]
y received 1
 55: (Receiver)(state 29) [printf('y received %d\\n',1)]
 56: (Receiver)(state 30) [state = 5]
 57: (Receiver)(state 55) [((state==5))]
 58: (Receiver)(state -) [values: 2!1]
 58: (Receiver)(state 33) [link2!1]
put 1 on link2
 59: (Receiver)(state 34) [printf('put 1 on link2\\n')]
 60: (Receiver)(state 35) [state = 0]
 61: (Emitter) (state -) [2?1]
 61: (Emitter) (state 55) [(((state==5)&&link2?[1]))]
 62: (Emitter) (state -) [values: 2?1]
 62: (Emitter) (state 51) [link2?x]
get 1 from link2
 63: (Emitter) (state 52) [printf('get 1 from link2\\n')]
 64: (Emitter) (state 53) [state = 0]
 65: (Emitter) (state 55) [((state==0))]
x to send 0
 66: (Emitter) (state 2) [printf('x to send %d\\n',0)]
 67: (Emitter) (state 3) [state = 1]
```

Figure 8: Trace d'exécution

Ordre :

$$(m_1, m_2, p_1, p_2) \leq^\sharp (m'_1, m'_2, p'_1, p'_2)$$
$$\iff$$
$$m_1 \geq m'_1 \ \wedge m_2 \leq m'_2 \ \wedge p1 \geq p'1 \ \wedge p2 \leq p'2$$

Union :

$$(m_1, m_2, p_1, p_2) \cup^\sharp (m'_1, m'_2, p'_1, p'_2)$$
$$=$$
$$(min(m_1, m'_1), max(m_2, m'_2), min(p1, p'1), max(p2, p'2))$$

Les affectations :

- `x:=k ; y := k'` : $(m'_1, m'_2, p'_1, p'_2) = (k - k', k - k', k + k', k + k')$
- `x := x+k` : $(m'_1, m'_2, p'_1, p'_2) = (m_1 + k, m_2 + k, p_1 + k, p_2 + k)$
- `y := y+k` : $(m'_1, m'_2, p'_1, p'_2) = (m_1 - k, m_2 - k, p_1 + k, p_2 + k)$
- `x := -x` : $(m'_1, m'_2, p'_1, p'_2) = (-p_2, -p_1, -m_2, -m_1)$

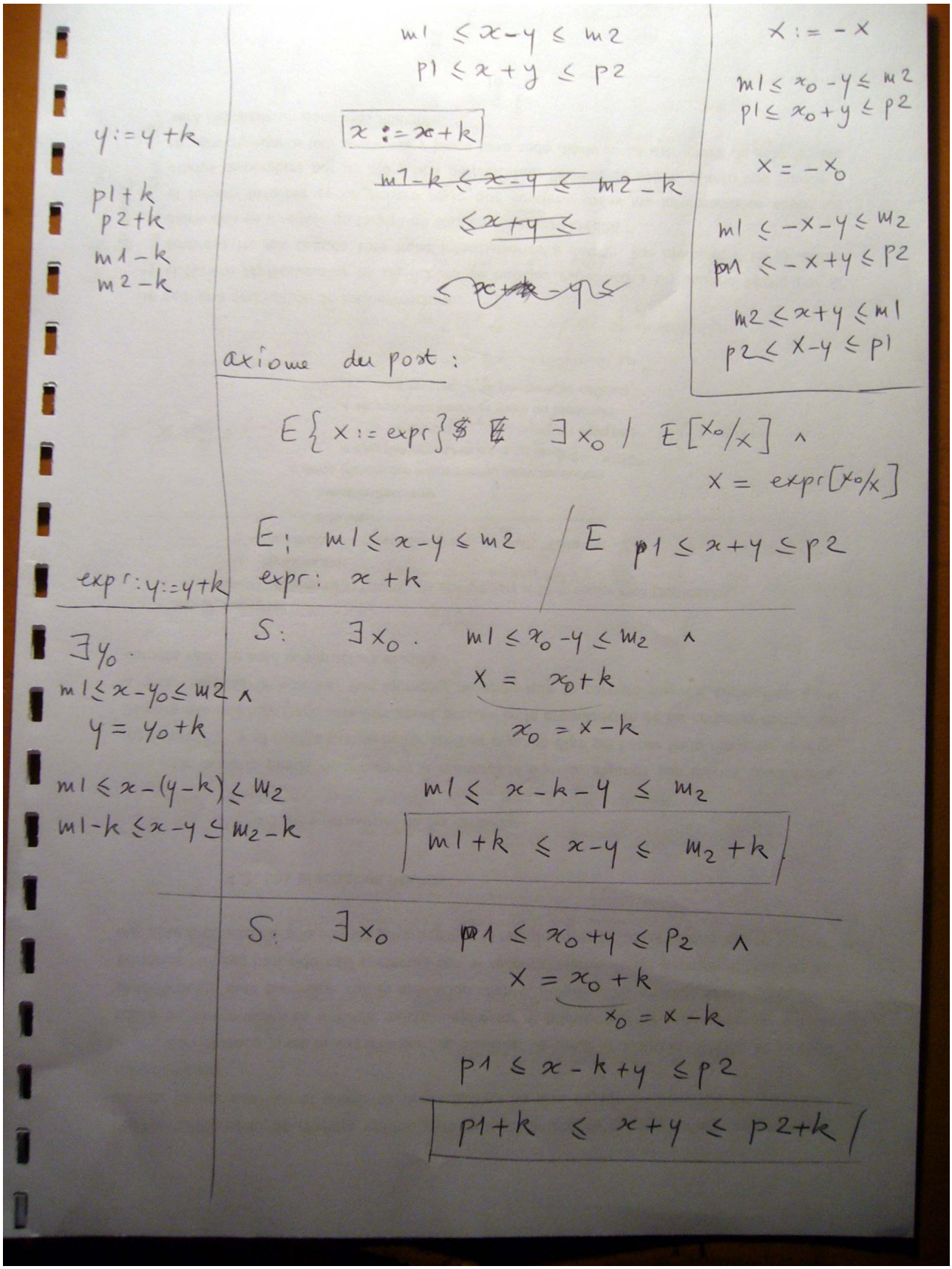
Exemple, détail du calcul pour `x' := -x` :

$$m_1 \leq x - y \leq m_2$$
$$m_1 + y \leq x \leq m_2 + y$$
$$-m_2 - y \leq x' = -x \leq -m_1 - y$$
$$-m_2 \leq x' + y \leq -m_1$$

$$p_1 \leq x + y \leq p_2$$
$$p_1 - y \leq x \leq p_2 - y$$
$$-p_2 + y \leq x' \leq -p_1 + y$$
$$-p_2 \leq x' - y \leq -p_1$$

Les conditions :

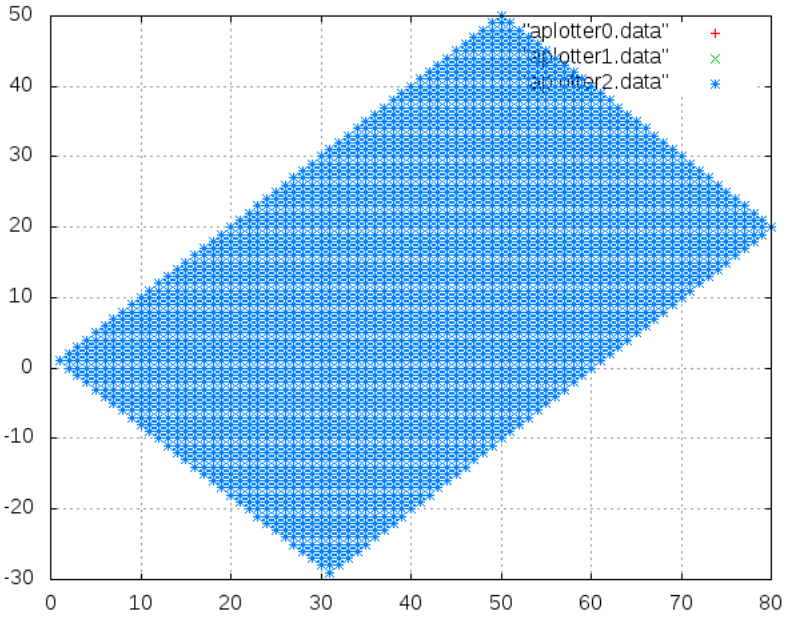
- `x=y` : cette condition est représentable comme $(0, 0, -\infty, +\infty)$. L'intersection avec un (m_1, m_2, p_1, p_2) est facile : si $0 \in [m1, m2]$ il y a une intersection non vide, la condition est possible, et ça renforce en $(0, 0, p1, p2)$. Si $0 \notin [m1, m2]$, il n'y a pas d'intersection, la condition n'est pas possible, la valeur abstraite associée à l'état but est le domaine vide, représentable par exemple avec $(2, 1, 2, 1)$.
- `x` $\geq$ `y` : représentable par $(0, +\infty, -\infty, +\infty)$, en faisant l'intersection avec un (m_1, m_2, p_1, p_2), on trouve $(max(0, m_1), min(+\infty, m_2), max(-\infty, p1), min(+\infty, p_2)) = (max(0, m_1), m_2, p_1, p_2)$.



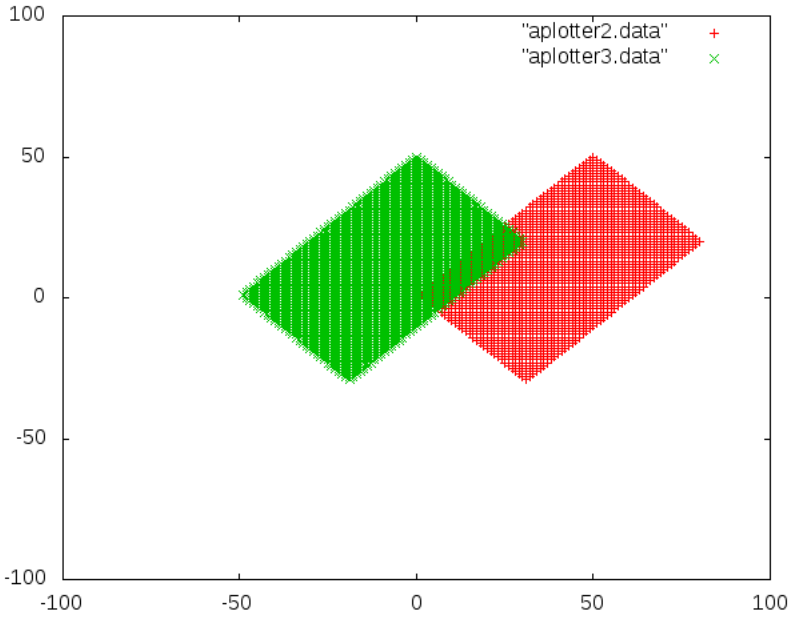
Le programme :

```
get (b) ; -- une valeur inconnue
if (b) then
   x := 1 ; y := 1 ;
else
   x := 80 ; y := 20 ;
end if ;
while ... loop
   x := x - 50 ;
   y := y + 30 ;
end loop
```

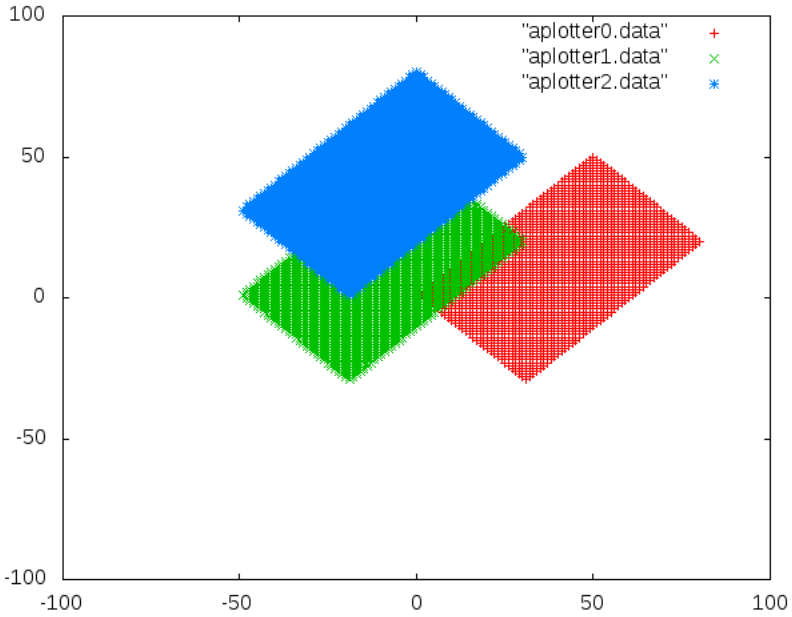
Le point `(1,1)` : $(0, 0, 2, 2)$
Le point `(80,20)` : $(60, 60, 100, 100)$
Union : $(0, 60, 2, 100)$.



On déplace le dernier avec `x := x - 50 ;`, ça donne un décalage à gauche $(-50, 10, -48, 50)$:



On déplace le dernier avec `y := y + 30 ;`, ça donne un décalage vers le haut $(-80, -20, -18, 80)$:

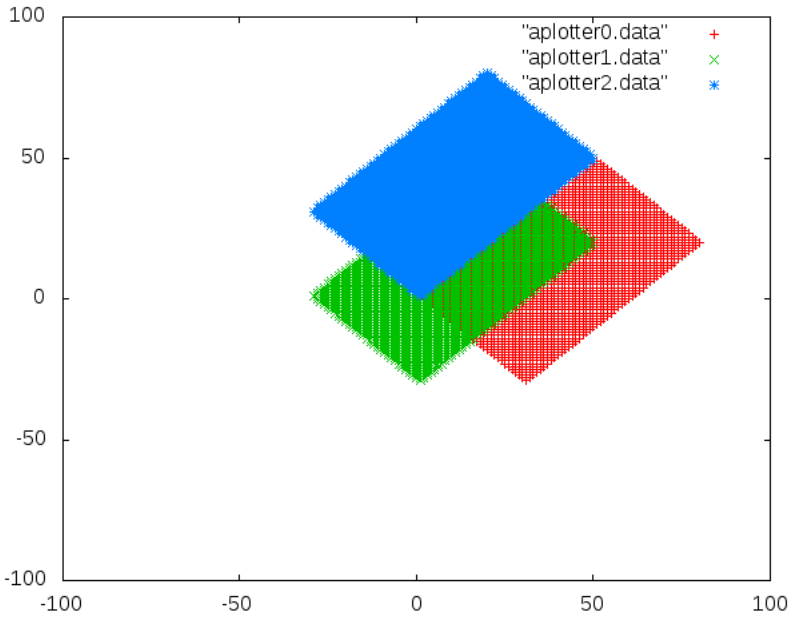


Si on prend le programme :

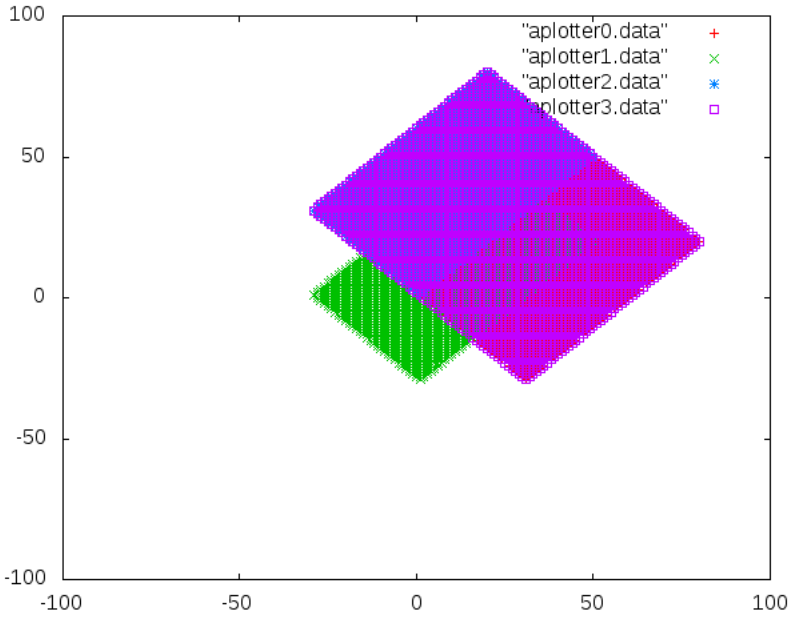
```
get (b) ; -- une valeur inconnue
if (b) then
   x := 1 ; y := 1 ;
else
   x := 80 ; y := 20 ;
end if ;
  // 0 60 2 100
while ... loop
   x := x - 30 ;
```

```
  // -30 30 -28 70
   y := y + 30 ;
  // -60 0 2 100
end loop
  // ( 0 60 2 100 ) U (-60 0 2 100) = (-60 60 2 100)
// si on refait :
x := x -30
// ca donne : (-90, 30, -28, 70)
y := y+30
// ca donne : (-120, 0, 2 100)
```

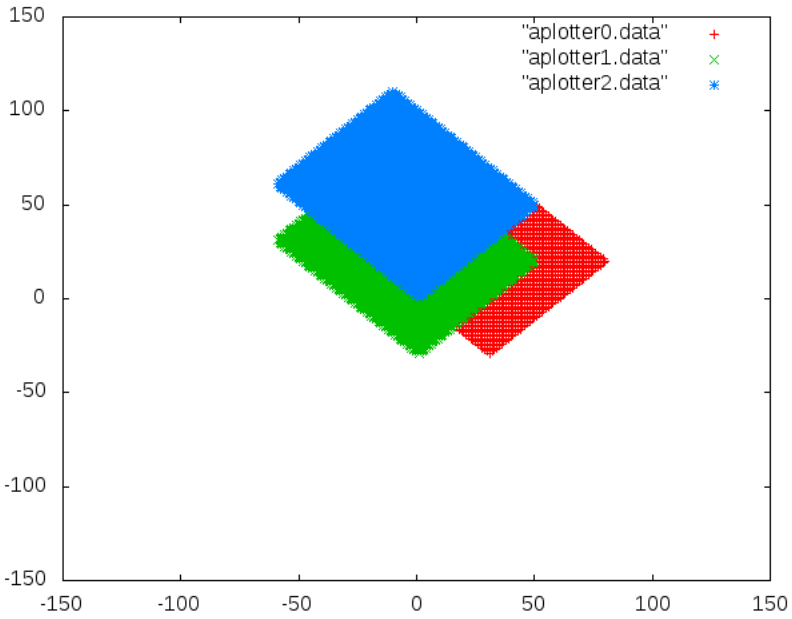
le décalage à gauche et celui vers le haut font que ça se recolle :



Du coup, en tournant dans la boucle, on fait l'union et ça donne :



Si on refait un tour dans la boucle, ça va donner :



Est-ce que ça converge vers une bande diagonale gauche ?