Cours MOdélisation, Vérification et EXpérimentations
Exercices (avec les corrections)
Structures partiellement ordonnées, treillis complets, points-fixes (I)
par Dominique Méry
23 mai 2025

☛ *Nous supposons que les opérations suivantes sont définies pour une collection $A_0, \ldots,$ $A_i \ldots i \in \mathbb{N}$ de parties de l'ensemble E :*
— $\cup_{i\in\mathbb{N}} A_i = \{e \in E | \exists i \in \mathbb{N} : e \in A_i\}$
— $\cap_{i\in\mathbb{N}} A_i = \{e \in E | \forall i \in \mathbb{N} : e \in A_i\}$
Ces deux définitions sont correctes et légales. Elles définissent en compréhension deux ensembles.

Exercice 1 *Montrer que les structures suivantes sont des structures partiellement ordonnées inductives :*

Question 1.1 $(\mathbb{P}(E), \subseteq)$ *avec E un ensemble quelconque.*

◇— **Solution de la question 1.1**

La démonstration consiste à montrer que $\varnothing$ est le plus petit élément de $\mathbb{P}\,E$ pour l'ordre d'inclusion des parties d'un ensemble E quelconque. Puis, il faut montrer que l'union des parties d'une suite croissante de parties de E est une partie de E et est la plus grande partie de E contenant cette suite. Ces éléments sont déduits de la définition de la structure $\mathbb{P}(E)$.
Soit une suite croissante de parties $A_i, i \in 0..\infty$ de E. On définit trivialement $A = \cup_{i=0}^{i=\infty} A_i$. A est la borne supérieure de cette suite.

Fin 1.1

Question 1.2 $(E^{\perp}, \sqsubseteq)$ *tel que*

1. $E^{\perp} = E \cup \perp$ *(et $\perp \notin E$)*
2. $\sqsubseteq \subseteq E \times E$ *: soit $x \in E$ et $y \in E$ $x \sqsubseteq y \Leftrightarrow (x = \perp \vee x = y)$*

◇— **Solution de la question 1.2**

Par construction, $\perp$ est le plus petit élément de cette strure. De plus, si on considère une suite croissante de cette structure, nécessairement elle stagnera soit sur $\perp$ soit sur un meme élément qui sera le plus grand élément de cette suite et la borne supérieure.

Fin 1.2

Question 1.3 *Soient A et B deux ensembles. Montrer que la structure $(A \nrightarrow B, \sqsubseteq)$ est une structure partiellement ordonnée inductive.*

◇— **Solution de la question 1.3**

Pour deux fonctions partielles f et g, $f \sqsubseteq g$ est défini par :
— $DOM(f) \subseteq DOM(g)$ *et* $\forall x \in DOM(f) : f(x) = g(x)$.
— *ou*
— $graph(f) \subseteq graph(g)$.

La fonction $\perp_{A\nrightarrow B}$ est le plus petit élément de cette structure et est la fonction définie nulle part : $DOM(\perp_{A\nrightarrow B}) = \varnothing$.
Si $(f_i : i \in \mathbb{N})$ est une suite croissante de fonctions de cette structure, alors on peut définir la borne supérieure de cette suite f comme suit :
— $dom(f) = \bigcup_{i\in\mathbb{N}} dom(f_i)$
— $\forall x \in dom(f) : \exists i \in \mathbb{N} : x \in dom(f_i) \wedge f(x) = f_i(x)$

Le choix de i n'a pas d'importance et f est bien une fonction définie qui est la borne supérieure de cette suite. En effet, si $x \mapsto y1 \in f$ et $x \mapsto y2 \in f$, alors cela signifie que $f_i(x) = y1$ et $f_j(x) = y2$. Si on suppose que $i \leq j$, alors $f_i \sqsubseteq f_j$ selon la suite des fonctions f_k et dans ce cas, $f_i(x) = f_j(x)$. On en d'eduit que $y1 = y2$ et que f est une fonctoon partielle de A dans B.

Fin 1.3

Exercice 2 *On rappelle qu'une fonction continue au sens de la topologie de Scott est monotone croissante. Indiquer et montrer si les fonctionnelles suivantes sont monotones et/ou continues).*

1. $(F_1(f))(x) \mathrel{\widehat{=}}$ if $(\forall y \in \mathbb{Z}.f(y) = y)$ then $f(x)$ else $\perp$
2. $(F_2(f))(x) \mathrel{\widehat{=}}$ if $x \notin dom(f)$ then 0 else $\perp$
3. $(F_3(f))(x) \mathrel{\widehat{=}}$ if $x = 0$ then 1 else $f(x{+}1)$

$\perp$ est une expression qui signifie que cést une valeur indéfinie.

Question 2.1 $(F_1(f))(x) \mathrel{\widehat{=}}$ if $(\forall y \in \mathbb{Z}.f(y) = y)$ then $f(x)$ else $\perp$

◇— **Solution de la question 2.1**

*On utilise la suite croissante h_i de fonctions telles que $h_i(x) =$ **if** $x \leq i$ **then** x **else** $\perp$ **fi**. On montre que $F_1(Sup_1(h_i : i \in \mathbb{N}\})) \neq Sup_1(F_1(h_i) : i \in \mathbb{N}\})$:*

- *$Sup_1(h_i : i \in \mathbb{N}\}) = Id$ où Id est l'identité de at dans $\mathbb{N}$.*
- *$dom(F_1(h_i)) = \varnothing$*

On a montré qu'elle n'est pas continue. On montre ensuite qu'elle est croisssante.

Fin 2.1

Question 2.2 $(F_2(f))(x) \mathrel{\widehat{=}}$ if $x \notin dom(f)$ then 0 else $\perp$

◇— **Solution de la question 2.2**

On interprète F_2 sur le domaine suivante $(\mathbb{Z} \nrightarrow \mathbb{Z}, \sqsubseteq_1)$. Le symbole $\perp$ signifie que la fonction n'est pas définie au point considéré.
Soit la suite de fonctions h_i définies comme suit. On se donne un x tel que $h_k(x) \neq \perp$. On suppose que cette suute est croissante ($h_0 \sqsubseteq_1 h_1 \ldots \sqsubseteq_1 h_i \ldots$.
Puisque $(\mathbb{Z} \nrightarrow \mathbb{Z}, \sqsubseteq_1)$ est une structure partiellement ordonnée inductive (CPO), il existe une borne supérieure pour cette suite de fonctions notée $Sup_1(\{h_i : i \in \mathbb{N}\})$. $Sup_1(\{h_i : i \in \mathbb{N}\})(x) = h_k(x)$ et $F_2(Sup_1(h_i : i \in \mathbb{N}\})(x) = h_k(x)$ et $Sup_1(\{F_2(h_i) : i \in \mathbb{N}\})(x)$ est défini par :

1. $\forall j < k.F_2(h_j)(x) = 0$
2. $\forall j \geq k.F_2(h_j)(x) = \perp$

On en déduit que $Sup_1(\{F_2(h_i) : i \in \mathbb{N}\})(x) = Sup(\{0, \perp\}) = 0$.
Pour cette suite de fonctions h_k, $Sup_1(\{h_i : i \in \mathbb{N}\}) \neq F_2(Sup_1(\{h_i : i \in \mathbb{N}\})$.
La fonction F_2 n'est pas continue pour la topologie de Scott.
Pour la croissance de cette fonction, on se donne deux fonctions f et g telles que pour une valeur x $f(x) = \perp$ et $g(x) \neq \perp$. Puis on applique F_2, $F_2(f)(x) = 0$ et $F_2(g)(x) = \perp$. Donc $F_2(f)$ n'est pas plus petit que $F_2(g)$.
Donc F_2 n'est pas croissante.

Fin 2.2

Question 2.3 $(F_3(f))(x) \mathrel{\widehat{=}}$ if $x = 0$ then 1 else $f(x{+}1)$

◇— **Solution de la question 2.3**

On considère une suite croissante de fonctions $(h_k : k \in \mathbb{N})$ et on montre que $F_3(Sup_1(h_i : i \in \mathbb{N}\}) = Sup_3(F_3(h_i) : i \in \mathbb{N}\})$.

- *La suite croissante $(h_k : k \in \mathbb{N})$ admet une borne supérieure notée h*
- *La suite $(F_3(h_k) : k \in \mathbb{N})$ est croissante :*
  - *$x = 0$: $\forall k \in \mathbb{N} : (F_3(h_k)(0) = 1 = h(0)$.*
  - *$x \neq 0$: $\forall k \in \mathbb{N} : x \in dom(h_k) \Rightarrow (F_3(h_k)(x) = h_k(x{+}1) = h(x{+}1)$ et en particulier, si $x \in dom(h_{k0})$, alors la propriété est vraie pour tputes les valeurs de k plus grande que $k0$ et en particulier $x \in dom(h)$.*
- *$(F_3(h_k) : k \in \mathbb{N})$ admet une borne supéreieure notée g et elle vérifie la propriété $g \sqsubseteq F_3(h)$ puisque g est la borne supérieure de la suite $(F_3(h_k) : k \in \mathbb{N})$*
- *$dom(g)$ est l'union des domaines des fonctions $F_3(h_k)$ et $dom(g)$ est l'union des domaines de h_k. g et $F_3(h)$ ont même domaine de définition ety sont égales.*

On en déduit que la fonction F_3 est continue pour la topologie de Scott.
Dans ce cas, d'après le théorème de Kleene, elle admet un plus petit point-fixe noté μF_3. On peut montrer que la fonction $g(x) \mathrel{\widehat{=}} \text{if } x \leq 0 \text{ then } 1 \text{ else } undefined$ est un point-fixe de F_3 et donc que $\mu F_3 \sqsubseteq g$. On montre ensuite que la fonction μF_3 a même domaine que g et on en déduit que $\mu F_3 = g$

Fin 2.3

Exercice 3 *Déterminer les points-fixes des fonctionnelles suivantes et leur plus petit point-fixe, s'ils existent. On travaille dans $\mathbb{Z}$.*

1. $F_1(f)(x) \mathrel{\widehat{=}} \text{if } f(x) \equiv 0 \text{ then } 1 \text{ else } 0$ *et expliquer si le programme f1 a du sens et ce qu'il calcule :*

◇– **Solution de la question 3.0**

On considère une suite croissante de fonctions $(h_k : k \in \mathbb{N})$ et on montre que $F_1(Sup_1(h_i : i \in \mathbb{N}\}) = Sup_3(F_1(h_i) : i \in \mathbb{N}\})$. Une analyse conduit à ces observations :

— $h_i(x) = 0$ *:* $F_1(h_i)(x) = 1.$
— $h_i(x) \neq 0$ *:* $F_1(h_i)(x) = 0.$

Comme la suite des fonctions h_i est croissante, pour une valeur donnée de x, la suite des fonctions h_i ont la même valeur en x. On en déduit que les fonctions $F_1(h_i)$ sont égales quand elles sont définies pour une valeur de x et donc que cette suite est une suite croissante qui admet un borne supérieuere. On en déduit que la fonction F_1 est continue. D'après le théorème de Kleene, F_1 admet un plus petit point-fixe défini selon la construction suivante :

— ${F_1}^0 = \varnothing$ *: la fonction définie nulle part.*
— ${F_1}^{i+1} = \{x \mapsto y | {F_1}^i(x) = 0 \wedge y = 1 \vee {F_1}^i(x) \neq 0 \wedge y = 0\}$
— ${F_1}^1 = \{x \mapsto y | {F_0}^i(x) = 0 \wedge y = 1 \vee {F_1}^0(x) \neq 0 \wedge y = 0\} = \varnothing$
— ${F_1}^2 = \{x \mapsto y | {F_1}^i(x) = 0 \wedge y = 1 \vee {F_1}^1(x) \neq 0 \wedge y = 0\} = \varnothing$
— ...
— ${F_1}^i = \varnothing$

On en déduit que le plus petit-point fixe existe et est la fonction définie nulle part.
On peut définir la fonction C qui calcule la fonction indéfinie

Listing 1 – framac-mainf1.c

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


int f1(int x)
{ if (f1(x) == 0)
    { return(1);
    }
  else
    { return(0);
    }
}


int main()
{
  int val,num;
   printf("Enter a number: ");
   scanf("%f", &num);
   // Computes something ?
   val = f1(num);
```

```
    printf("....␣f1(%d)=%d\n", num, val);
    return 0;
}
```

Puis on peut vérifier que cette fonction est indéfinie en vérifiant le contrat suivant :

Listing 2 – framac-f1.c

```
/*@ requires \false;
  @ ensures \false;
*/
int f1(int x)
{ if (f1(x) == 0)
    { return(1);
    }
  else
    { return(0);
    }
}
```

Fin 3.0

2. $F_2(f)(x) \triangleq$ if $f(x) \equiv 0$ then 0 else 1

◇— **Solution de la question 3.0**

Les trois fonctions suivantes sont des points-fixes de cette fonctionnelle :

— $f_0 = \lambda x.0$
— $f_1 = \lambda x.1$
— $f_\perp = \varnothing$

On procède comme pour l'exemple prévédent.

Fin 3.0

Listing 3 – framac-mainf1.c

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


int f2(int x)
{ if (f2(x) == 0)
    { return(0);
    }
  else
    { return(1);
    }
}


int main()
{
  int val,num;
   printf("Enter␣a␣number:␣");
   scanf("%d", &num);
   // Computes something ?
   val = f2(num);
   printf("....␣f2(%d)=%d\n", num, val);
   return 0;
}
```

Puis on peut vérifier que cette fonction est indéfinie en vérifiant le contrat suivant :

Listing 4 – framac-f1.c

```
/*@ requires \false;
  @ ensures \false;
*/
int f2(int x)
{ if (f2(x) == 0)
    { return(0);
    }
  else
    { return(1);
    }
}
```

3. $F_3(f)(x) \widehat{=}$ if $x = 0$ then 1 else $f(x{+}1)$

◇– **Solution de la question 3.0**

D'après l'exercice précédent, F_3 *est continue et donc admet un plus petit point-fixe définie par la suite des approximations suivantes :*

— $F_0 = \varnothing$
— $F_1 = \{0 \mapsto 1\}$
— $F_2 = \{0 \mapsto 1, -1 \mapsto 1\}$
— ...
— $F_{i+1} = \{0 \mapsto 1, -1 \mapsto 1, \ldots, -i \mapsto 1\}$
— ...

On en déduit que $\mu F_3 = \cup_{i \in \mathbb{N}} F^i$ *et donc* $\mu F_3 = \lambda x.if\ x \leq 0\ then 1\ fi$

Fin 3.0

Listing 5 – f1.c

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

int f1(int x)
{ if (f1(x) == 0)
    { return(1);
    }
  else
    { return(0);
    }
}


int main()
{
  int val,num;
   printf("Enter a number: ");
   scanf("%f", &num);
   // Computes something ?
   val = f1(num);
   printf(".... f1(%d)=%d\n", num, val);
   return 0;
}
```

Listing 6 – f2.c

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

```
int f2(int x)
{ if (f2(x) == 0)
    { return(0);
    }
  else
    { return(1);
    }
}

int main()
{
  int val,num;
   printf("Enter␣a␣number:␣");
   scanf("%f", &num);
   // Computes something ?
   val = f2(num);
   printf("....␣f(%d)=%d\n", num, val);
   return 0;
}
```

Listing 7 – f3.c

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



int f3(int x)
{ if (x == 0)
    { return(1);
    }
  else
    { return(f3(x+1));
    }
}

int main()
{
  int val,num;
   printf("Enter␣a␣number:␣");
   scanf("%f", &num);
   // Computes something ?
   val = f3(num);
   printf("....␣f2(%d)=%d\n", num, val);
   return 0;
}
```

Exercice 4 *Soit $E = \mathbb{N} \nrightarrow \mathbb{N}$, soit la fonctionnelle $\tau \in E \nrightarrow E$ définie par :*

$$(\tau(F))(x) \mathrel{\widehat{=}} \text{ if } x = 0 \text{ then } 1 \text{ else } x \cdot F(x-1)$$

1. *Calculer $\tau(\varnothing)$, $\tau^2(\varnothing) = \tau(\tau(\varnothing))$, $\tau^3(\varnothing)$. En déduire $\tau^i(\varnothing)$ et le démontrer par récurrence.*

2. *En déduire le plus petit point fixe.*

Question 4.1 *Calculer $\tau(\varnothing)$, $\tau^2(\varnothing) = \tau(\tau(\varnothing))$, $\tau^3(\varnothing)$. En déduire $\tau^i(\varnothing)$ et le démontrer par récurrence.*

◇— **Solution de la question 4.1**

La structure partiellement ordonnée $(\mathbb{N} \nrightarrow \mathbb{N}, \subseteq)$ où $\subseteq$ est l'inclusion dénsembles, est une structure partiellement ordonnée complète (CPO).

1. $\tau^0 = \varnothing$
2. $\tau_1 = \tau(\varnothing) = \{0 \mapsto 1\}$
3. $\ldots$
4. $\tau_{i+1} = \tau(\tau^i) = \{0 \mapsto 1, i \mapsto i!\}$

On montre cela par récurrence.

Fin 4.1

Question 4.2 *En déduire le plus petit point fixe.*

◇— **Solution de la question 4.2**

On en déduit que $\mu\tau = \{i \mapsto i! | i \in \mathbb{N}\}$.

Fin 4.2

Exercice 5 *Soit la fonctionnelle $\tau \in (\int \nrightarrow \int) \nrightarrow (\int \nrightarrow \int)$:*

$$(F(f))(x) \mathrel{\widehat{=}} \text{if } x > 100 \text{ then } x{-}10 \text{ else } f(f(x{+}11))$$

Question 5.1 *Montrez que $\mu F \sqsubseteq g$ avec*

$$g(x) \mathrel{\widehat{=}} \text{if } x > 100 \text{ then } x{-}10 \text{ else } 91$$

◇— **Solution de la question 5.1**

Pour montrer que $\forall x.pppf(x) \sqsubseteq g(x)$ avec

$$g(x) \mathrel{\widehat{=}} \text{if } x > 100 \text{ then } x{-}10 \text{ else } 91$$

on utilise l'induction de point-fixe avec $P(f) = f \sqsubseteq g$.
$P(f)$ ***est inclusif***
Pour cela, on considère une chaine de fonctions de $\mathbb{Z} \nrightarrow \mathbb{Z}$ notée $f_0, f_1, \ldots, f_i, \ldots$ dont la borne supérieure est f. On suppose que pour toutes les fonctions f_i, $P(f_i)$, Par définition de la borne supérieure, $f \sqsubseteq g$. OPn en déduit donc que $P(f)$.
Application de l'induction de point-fixe

— $P(\perp)$ *:* $\perp \sqsubseteq g$.
— *Supposons $P(f)$ cést-à-dire $f \sqsubseteq g$ ou encore que f est égale à gf sur le même domaine. Montrons que $P(F(f))$ est vraie.*
Cas 1 : $x > 100$
$F(f)(x) = x{-}10 = g(x)$
Cas 2 : $x \leq 100$
$F(f)(x) = f(f(x{+}11))$ *et* $x{+}11 \leq 111$.

- ***Cas 2-1 :*** $100 < x+11 \leq 111$
  $F(f)(x) = f(f(x+11)) \stackrel{\text{def}}{=} f(x+11-1) \stackrel{\text{simp}}{=} f(x+1)$
  - ***Cas 2-1-a :*** $90 \leq x < 100$
    $f(x+1) \stackrel{\text{simp}}{=} 91 \stackrel{\text{simp}}{=} g(x)$
  - ***Cas 2-1-b :*** $x = 100$
    $f(x+1) \stackrel{\text{simp}}{=} 101-10 \stackrel{\text{simp}}{=} 91 \stackrel{\text{simp}}{=} g(x)$
- ***Cas 2-2 :*** $x+11 \leq 100$
  $F(f)(x) = f(f(x+11)) \stackrel{\text{f(x+11)=91}}{=} f(91) = 91 = g(x)$

On en déduit que $P(\mu F)$ ou encore $\mu F \sqsubseteq g$

Fin 5.1

Question 5.2 *Ecrire une fonction C calculant cette fonction et étudier sa correction.*

◇– **Solution de la question 5.2**

On peut utiliser mes contrats pour montrer l'équivalence des deux fonctions calculant la fonction de MacCarthy mais avec une difficulté pour montrer que les appels sont décroissants et convergent.

Listing 8 – framac-mc91.c

```
/*@ ensures x>100 ==> \result == x-10;
@ ensures x <= 100 ==> \result == 91;

*/
int f1(int x)
{ if (x > 100)
    { return(x-10);
    }
  else
    { return(f1(f1(x+11)));
    }
}

/*@ ensures x>100 ==> \result == x-10;
@ ensures x <= 100 ==> \result == 91;

*/

int f2(int x)
{ if (x > 100)
    { return(x-10);
    }
  else
    { return(91);
    }
}

/*@ ensures  \result == 1;

*/
int check(int x)
{ int val1,val2;
  val1 = f1(x);
  val2 = f2(x);
  if (val1 == val2)
```

```
    { return(1);
    };
  return (0);
}
```

Listing 9 – framac-mainmc91.c

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

int f1(int x)
{ if (x > 100)
    { return(x-10);
    }
  else
    { return(f1(f1(x+11)));
    }
}



int f2(int x)
{ if (x > 100)
    { return(x-10);
    }
  else
    { return(91);
    }
}

int check(int x)
{ int val1,val2;
  val1 = f1(x);
  val2 = f2(x);
  if (val1 == val2)
    { return(1);
    };
  return (0);
}

int main()
{
  int val1,val2,val3,num;
   printf("Enter a number: ");
   scanf("%d", &num);
   // Computes the square root of num and stores in root.
   val1 = f1(num);
     val2 = f2(num);
     val3 = check(num);
     printf("Et le rÃ©sultat  f1(%d)=%d et la vÃ©rification: %d et ..... %d\n", num,
   return 0;
}
```

Fin 5.2

Exercice 6 *On considère une fonction* $f5$ *définie par le code C suivant :*

```
int f5(int x)
{if (x==0)
    { return(0);}
  else
    { if  (x > 0)
        {return(2-f5(1-x));}
      else
        {/* x <0 */
          return(f5(-x));}
    }
}
```

Question 6.1 *Traduire cette définition en une définition fonctionnelle qui précisera le domaine du problème.*

$\mathcal{F}(f)(x) = $ **if** $(x == 0)$ **then** 0 **elseif** $(x > 0)$ **then** $2{-}f(1{-}x))$ **else** $f(-x)$ **fi**

Listing 10 – framac-f5.c

```
/*@ ensures  x % 2 == 0  ==> \result == 0;
  @ ensures  x % 2  != 0  ==> \result == 2;
  @ assigns \nothing;
*/
int f5(int x)
{if (x==0)
    { return(0);}
  else
    { if  (x > 0)
        {return(2-f5(1-x));}
      else
        {
          return(f5(-x));}
    }
}
```

Listing 11 – framac-mainf5.c

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

int f5(int x)
{if (x==0)
    { return(0);}
  else
    { if  (x > 0)
        {return(2-f5(1-x));}
      else
        {/* x <0 */
          return(f5(-x));}
    }
}
```

```
int main()
{
  int val,num;
   printf("Enter␣a␣number:␣");
   scanf("%d", &num);
   // Computes something ?
   val = f5(num);
   printf("....␣f2(%d)=%d\n", num, val);
   return 0;
}
```

Code Python

```
def f(n):
    if (n == 0):
        return 0
    else:
        if (n > 0):
            return 2 - f(1 - n)
        else:
            return f(-n)
print(f(6))
```

Question 6.2 *Soient les définitions suivantes où $\mathcal{F}$ désigne la fonctionnelle définie dans la question précédente :*

— $\mathcal{F}^{2n} = \{(p, v_p) | 0 \leq p < n \wedge v_p = g(p)\} \cup \{(p, v_p) | 0 > p \geq -(n-1) \wedge v_p = g(p)\} \cup \{(n, g(n))\}$
— $\mathcal{F}^{2n+1} = \{(p, v_p) | 0 \leq p < n \wedge v_p = g(p)\} \cup \{(p, v_p) | 0 > p \geq -(n-1) \wedge v_p = g(p)\} \cup \{(n, g(n)), (-n, g(-n))\}$

Montrer qu'elles sont correctes en utilisant une récurrence.

Question 6.3 *En déduire que $\mu\mathcal{F} = \mathcal{F}^0 \cup \mathcal{F}^1 \cup \mathcal{F}^2 \cup \mathcal{F}^3 \cup \ldots \cup \mathcal{F}^{2n} \cup \mathcal{F}^{2n+1} \ldots$*

Question 6.4 *Montrer que, pour tout $p \in \mathbb{N}$, $p \in \mathcal{F}^{2p} \cup \mathcal{F}^{2p+1}$*

Question 6.5 *Montrer que $\mu\mathcal{F}$ vérifie la propriété $\mu\mathcal{F} \sqsubseteq g$ où $g(x) = \textbf{if } odd(x) \textbf{ then } 2 \textbf{ else } 0 \textbf{ fi}$*

Question 6.6 *En déduire que $\mu\mathcal{F} = g$.*

Exercice 7 *Soit la fonction définie comme suit :* $F(f)(x) = \left\{ \begin{array}{l} \textbf{if } x = p \textbf{ then } p \\ \textbf{else if } x = q \textbf{ then } q \\ \textbf{else } f(x+p+q) \\ \textbf{fi} \end{array} \right.$.

On suppose que p et q sont deux constantes non nulles entières positives distinctes et que F est une fonction partielle ($\in (\mathbb{Z} \nrightarrow \mathbb{Z}) \longrightarrow (\mathbb{Z} \nrightarrow \mathbb{Z})$). On se place dans le cadre de la topologie de Scott sur léspace $(\mathbb{Z} \nrightarrow \mathbb{Z}) \longrightarrow (\mathbb{Z} \nrightarrow \mathbb{Z}), \sqsubseteq)$ où $f \sqsubseteq g$ signifie que f est moins définie que g (ou $graph(f) \subseteq graph(g)$).

Question 7.1 *Expliquez clairement pourquoi l'équation $F(f) = f$ admet un plus petit point-fixe.*

Question 7.2 *Ecrire une fonction C calculant le plus petit point-fixe μF de F.*

Question 7.3

1. *Calculer $\mu F(p)$, $\mu F(q)$, $\mu F(-p)$, $\mu F(-q)$.*
2. *Calculer pour k entier naturel, $\mu F(-(k+1) \cdot p - k \cdot q)$ et $\mu F(-(k+1) \cdot q - k \cdot p)$*

Question 7.4 *Donnez une expression simplifiée de la fonction μF. Pour cela, on pourra utiliser la caractérisation de μF par le théorème du point-fixe pour les fonctions continues au sens de Scott.*

Exercice 8 *(invariant inductif)*
On rappelle les définitions suivantes. Un modèle relationnel $\mathcal{MS}$ pour un système $\mathcal{S}$ est une structure

$$(Th(s, c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$$

où

- $Th(s, c)$ *est une théorie définissant les ensembles, les constantes et les propriétés statiques de ces éléments.*
- x *est une liste de variables flexibles.*
- VALS *est un ensemble de valeurs possibles pour x.*
- $\{r_0, \ldots, r_n\}$ *est un ensemble fini de relations reliant les valeurs avant x et les valeurs après x'.*
- $\text{INIT}(x)$ *définit lénsemble des valeurs initiales de x.*

On note $\text{NEXT} \stackrel{def}{=} r_0 \vee \ldots \vee r_n$. Une propriété S est une propriété de sûreté pour le système $\mathcal{S}$, si $\forall y, x \in \text{VALS}.Init(y) \wedge \text{NEXT}^\star(y, x) \Rightarrow x \in S$. On définit la fonction suivante F sur $\mathcal{P}(\text{VALS})$ à valeurs dans $\mathcal{P}(\text{VALS})$: $F(X) = Init \cup Next[X]$ où $Next[X]$ est lénsemble des états accessibles à partir de X par $Next$. On rappelle aussi que x peut être une variable ou une liste de variables ; VALS est donc un ensemble de valeurs ou de tuples de valeures correspondant à x.

Question 8.1 *Montrer que $(\mathcal{P}(\text{VALS}), \subseteq, \varnothing, \cup, \cap)$ est un treillis complet.*

◇— **Solution de la question 8.1**

$\subseteq$ est la relation d'inclusion des parties de lénsemble VALS. La structure $(\mathcal{P}(\text{VALS}), \subseteq)$ est évidemment une structure partiellement ordonnée. L'élément $\varnothing$ est le plus petit élément de cette structure et VALS est le plus grand élément de cette structure. Enfin, pour toute famille de parties de VALS, notée $\{A_i | i \in I\}$, il existe un plus grand élément et un plus petit élément définis comme suit :

- $Sup(A_i | i \in I) = \bigcup_{i \in I} A_i$ *: union des parties.*
- $Inf(A_i | i \in I) = \bigcap_{i \in I} A_i$ *: intersection des parties.*

Fin 8.1

Question 8.2 *Montrer que F est croissante monotone.*

◇— **Solution de la question 8.2**

Soient deux parties X et Y de $\textsc{Vals}$ *telles que $X \subseteq Y$. Soit x un élément de $F(X)$. Supposons que x n'est pas éléments de $Init$. Alors il existe z tel quue $z \in X$ et $Next(z, x)$. Puisque $z \in X$, alors $z \in Y$. On en déduit que $x \in F(Y)$. Nous avons montré que $\forall x.x \in F(X) \Rightarrow x \in F(Y)$ ou encore $F(X) \subseteq F(Y)$.*

Fin 8.2

Question 8.3 *Montrer que F admet un plus petit point-fixe noté μF.*

◇— **Solution de la question 8.3**

Puisque F est une fonction monotonte croisssante sur un treillis complet, d'après le théorème de Knaster-Tarski, il existe un plus petit point-fixe note μF pour F.

Fin 8.3

Question 8.4 *Montrer que μF est un invariant inductif de F et que cést le plus petit.*

◇— **Solution de la question 8.4**

Puisque μF est le plus petit pint-fixe de F, il est, en particulier, un point-fixe et vérifie la relation suivante :

$$F(\mu F) = \mu F = Init \cup Next[\mu F] \tag{1}$$

Léxpression $Next[\mu F]$ est l'application de la relation $Next$ à tous les éléments de μF. On dérive de l'équation 1 les deux propriétés :

1. *$Init \subseteq \mu F$*
2. *$Next[\mu F] \subseteq \mu F$*

Ces deux propriétés définissent exactement que μF est un invariant inductif. Si I est un autre invariant inductif, alors il vérifie aussi cette équation et par définition, μF est le plus petit ensemble satisfaisant cette propriété. Donc, $\mu F \subseteq I$.

Fin 8.4

Question 8.5 *Montrer que, pour toute propriété de sûreté S, $\mu F \subseteq S$.*

◇— **Solution de la question 8.5**

Une propriété S est une propriété de sûreté pour un système caractérisé par le système de transition ci-dessus, si $\forall y, x \in \textsc{Vals}.Init(y) \wedge \textsc{Next}^{\star}(y, x) \Rightarrow x \in S$. On peut noter que la formulation peut être changée comme suit sous une forme équivalente : $\forall x \in \textsc{Vals}.(\exists y.Init(y) \wedge \textsc{Next}^{\star}(y, x)) \Rightarrow x \in S$. Soit lénsemble A suivant : $A = \{a | a \in \textsc{Vals} \wedge (\exists y.Init(y) \wedge \textsc{Next}^{\star}(y, x))\}$. Si S est une propriété de sûreté, alors $A \subseteq S$. De plus, A vérifie la relation $F(A) = A$. Donc on en déduit que $\mu F \subseteq A$ puisque cést le plus petit point-fixe de F. On en déduit que si S est une propriété de sûreté, alors $\mu F \subseteq S$.

Fin 8.5

Question 8.6 *On suppose que* $\textsc{Vals}$ *est finie. Montrer qu'il existe un algorithme pour vérifier qu'une propriété S est une propriété de sûreté pour un système donné défini comme ci-dessus.*

◇— **Solution de la question 8.6**

Le calcul de μF sur un treillis fini est le calcul de la suite $(F^i)_{i \in \mathbb{N}}$ définie comme suit :

— $F^0 = \varnothing$

— $F^{i+1} = F(F_i)$, $\forall i \in \mathbb{N}$

Dans ce cas, on calcule la suite et la suite cumulée en les rangeant respectivement dans x et dans y. L'itération est bornée par $Card(T)$, puisque dans le cas contraire, on pourrait construire une suite de valeurs $Next(x_0, x_1) \ldots Next(x_i, x_{i+1}) \ldots Next(x_n, x_{n+1})$ où n est le cardinal de T avec des éléments tous distincts et cela n'est â possible.

precondition : $f \in T \longrightarrow T$
postcondition : $result = \mu.f$
local variables : $x, y \in T, i \in \mathbb{N}$

$\ell_0 : \{x, y \in T\}$
$x := \bot;$
$y := \bot;$
$i := 0;$
$\ell_{11} : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i \leq Card(T) \wedge i = 0\};$
while $i \leq Card(T)$ **do**
  $\ell_1 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i \leq Card(T)\};$
  $x := f(x);$
  $\ell_2 : \{x, y \in T \wedge x = F^{i+1} \wedge y = \bigcup_{k=0;k=i} F^k \wedge i \leq Card(T)\};$
  $y := x \sqcup y;$
  $\ell_3 : \{x, y \in T \wedge x = F^{i+1} \wedge y = \bigcup_{k=0;k=i+1} F^k \wedge i \leq Card(T)\};$
  $i := i{+}1;$
  $\ell_4 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i \leq Card(T){+}1\};$
;
$\ell_5 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i = Card(T){+}1\};$
$result := y;$
$\ell_6 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i = Card(T){+}1 \wedge result = y\};$

Algorithme 1: Calcul du point-fixe sur un treillis fini

Fin 8.6

Exercice 9

Question 9.1 *Soit le petit programme suivant annoté mais incomplet.*

```
 /*@ requires z == exp1 ;
  ensures \result == exp2;
*/

int q2(int x,int y, int z){

  /*@ assert B(x,y,z)   ; */
  x = y+1;
  /*@  assert  A(x,y,z); */
  y = x + z;
  /*@  assert  y == 3 + x ; */
  return y;
  }
```

En utilisant l'opérateur wp, proposer des assertions pour $A(x, y, z)$ et $B(x, y, z)$ et des valeurs pour les expressions expr1 et expr2, afin que le contrat soit correct.

◇— Solution de la question 9.1

On utilise la calcul wp à partir de la postcondition et nous appliquons successivement en stoppant au début de q2.

```
/*@ requires z == exp1 ;
  ensures \result == exp2;
*/

int q2(int x,int y, int z){
    /*@ assert y+1+exp1 == 3 + y+1  && y+1+exp1 == exp2; */
  /*@ assert B(x,y,z)  ; */
    /*@ assert y+1+z == 3 + y+1  && y+1+z == exp2; */
  x = y+1;
  /* @ assert A(x,y,z); */
   /*@ assert x+z == 3 + x  && x+z == exp2; */
  y = x + z;
  /*@ assert y == 3 + x  && y == exp2; */
  return y;
  }
```

En particulier, nous avons que $y == exp2$ *avant de renvoyer la valeur de y.*
La condition de début de la fonction q2 est

```
  /*@  assert  y+1+exp1  == 3 + y+1   &&  y+1+exp1 == exp2; */
```

On en déduit que exp1 = 3 et exp2= y+4 où y est la valeur de y au point d'appel soit `\old(y)`.
On en déduit le contrat suivant.

```
/*@ requires z == 3  ;
  ensures \result == \old(y)+4;
*/

int q2(int x,int y, int z){

  /* @ assert B(x,y,z)  ; */
  /*@ assert z == 3   ; */

  /*@ assert y+1+z == 3 + y+1 ; */
  x = y+1;
  /* @ assert A(x,y,z); */
  /*@ assert x+z == 3 + x; */
  y = x + z;
  /*@ assert y == 3 + x ; */
  return y;
  }
```

Fin 9.1

Question 9.2 *Soit le petit programme suivant annoté mais incomplet.*

```
/*@ requires A(x,y,z)  ;
  ensures \result == 6 ;
*/

int q2(int x,int y, int z){

  /* @ assert B(x,y,z)  ; */
   x = y+1;
  /* @ assert C(x,y,z); */
   y = x + z;
  /* @ assert D(x,y,z); */
```

```
  return y;
  }
```

En utilisant l'opérateur wp, proposer des assertions pour $A(x,y,z)$, $B(x,y,z)$, $C(x,y,z)$ et $D(x,y,z)$, afin que le contrat soit correct.

Question 9.3 *Soit le petit programme suivant annoté mais incomplet.*

```
/*@ requires A;
ensures \result == 0;
  assigns \nothing;
*/

int f(int c) {
     //@ assert 49 == 49 && 2*c ==  2*c &&  49+2*c+1 ==  (c+1)*(c+1);
  int x = 49;
     //@ assert x == 49 && 2*c ==  2*c &&  x+2*c+1 ==  (c+1)*(c+1);
  int z = 2*c;
    //@ assert x == 49 && z ==  2*c &&  x+z+1 ==  (c+1)*(c+1);
  int y = (2*c+1)*(2*c+1);
    // @ assert B;
    //@ assert x == 49 && z ==  2*c &&  x+z+1 ==  (c+1)*(c+1);
    y= x+z+1;
    //@ assert x == 49 && z ==  2*c && y ==  (c+1)*(c+1);
    return(0);
}
```

En utilisant l'opérateur wp, proposer des assertions pour A et B, afin que le contrat soit correct.

Question 9.4 *Soit le petit programme suivant annoté mais incomplet.*

```
/*@ requires  A(x,y,z)   ;
  ensures \result == 12 ;
*/

int q2(int x,int y, int z){

  /* @ assert B(x,y,z)   ; */
  x = y+z;
  /* @  assert  C(x,y,z); */
  y = x + 1;
  /* @  assert  D(x,y,z); */
  return y;
  }
```

En utilisant l'opérateur wp, proposer des assertions pour $A(x,y,z)$, $B(x,y,z)$, $C(x,y,z)$ et $D(x,y,z)$, afin que le contrat soit correct.

Question 9.5 *Soit le petit programme suivant annoté mais incomplet.*

```
33/*@ requires A;
ensures \result == 0;
  assigns \nothing;
*/
```

```
int f(int c) {
    //@ assert (2*c == 14 ==> 49 == 49 && 2*c == 2*c && 49+2*c +1 == (c+1)*(c+1))
&& (2*c != 14 ==> 49 == 49 && 2*c == 2*c && 49+1== 50);
if
  int x = 49;
  int z = 2*c;

      //@ assert (z == 14 ==> x == 49 && z == 2*c && x+z+1 == (c+1)*(c+1));
//@ assert (z != 14 ==> x == 49 && z == 2*c && x+1== 50);

      //@ assert (z == 14 ==> x == 49 && z == 2*c && x+z+1 == (c+1)*(c+1))
&& (z != 14 ==> x == 49 && z == 2*c && x+1== 50);
if  z == 14
              {
    //@ assert x == 49 && z == 2*c && x+z+1 == (c+1)*(c+1);
  int y = (2*c+1)*(2*c+1);
    // @ assert B
;    //@ assert x == 49 && z == 2*c && x+z+1 == (c+1)*(c+1);
    y= x+z+1;
    //@ assert x == 49 && z == 2*c && y == (c+1)*(c+1);

              }
  else
    {
        // @ assert C;
    //@ assert x == 49 && z == 2*c && x+1== 50;
    y= x+1;
    //@ assert x == 49 && z == 2*c && y == 50;

    }
      //@ assert (x == 49 && z == 2*c && y == (c+1)*(c+1)) || (x == 49 && z ==
2*c && y == 50);

    return(0);
}
```

En utilisant l'opérateur wp, proposer des assertions pour A et B, afin que le contrat soit correct.