TD n° 6 :Sur la terminaison des algorithmes

Rappel:

Il est important de pouvoir montrer qu'un programme termine. Or, il n'existe pas d'algorithme permettant de dire si un programme termine **toujours** ou non″. On dit en informatique que le **problème de l'arrêt** est **indécidable**.
Pour autant, cela ne signifie pas qu'on n'est pas capable de prouver que certains programmes peuvent se terminer: le problème de l'arrêt est donc **partiellement calculable**.

L'idée principale de toutes les démonstrations de la terminaison consiste à trouver une fonction que l'on appellera **fonction de terminaison** ou **convergent** de la boucle.
Une fonction de terminaison pour une boucle est une fonction qui dépend des variables de l'algorithme et dont la valeur **positive décroît strictement** à chaque répétition de la boucle.
La **condition d'arrêt** de la boucle doit impliquer le **dépassement** d'une valeur particulière de la fonction de terminaison.

Exercice 1

On suppose maintenant que l'on est capable d'écrire un algorithme, appelé **Terminator**, dont la fonction est de répondre vrai si un programme se termine et faux si un programme ne se termine pas.
Ainsi, la fonction **Terminator**(P) renvoie vrai si le programme P termine toujours et faux si P est capable de boucler.
On considère le programme appelé **Transformers** suivant :

```
Transformers
tant que Terminator(Transformers) faire
 afficher(″Continuer″)
fin
```

A quelle condition le programme **Transformers** se termine-t-il ?
(Grâce à cet exemple simple, on comprend donc que le problème de la terminaison des algorithmes n'est pas si simple.)

Exercice 2:

Intuitivement, on peut *conjecturer* que le programme ci-dessous se termine et arrive à **1**.
Pourtant, il n'existe pour le moment aucune preuve permettant d'assurer que l'on arrive **toujours** à 1: il est impossible de décider si cet algorithme se termine **toujours** ou non.

```
fonction syracuse(n)
Entrées : un entier n
Syr←n;
tant que Syr ≠1 faire
        si Syr est pair alors
        Syr ← Syr/2
        sinon
        Syr ← 3 * Syr + 1
        fin
fin
```

Néanmoins, il suffit de faire tourner cet algorithme sur quelques valeurs pour se rendre compte qu'on arrive toujours à 1 et qu'à partir de là, le comportement de la suite est **cyclique**.

Exercice 3:

Essayer d'expliquer pourquoi l'algorithme simple ci-dessous se termine.

```
trèsSimple(n :entier)
i← 0
tant que i <  exp(n) faire
     i← i + 1
fin

afficher i
```

1 – Proposer une fonction de terminaison F.
2 – Donner un minorant de F lors de l'exécution de l'algorithme. Que peut-on en déduire ?

Exercice 4:

Etudier la terminaison de l'algorithme ci-dessous.

```
fonction RechDicho(L,a)
Entrées : L[1..n] une liste triée et a un élément
Sorties : un booléen
g ← 1;
d ← n;
tant que d - g > 0 faire
        m ← Ent((d + g)=2);      /* partie entière */
        si L[m] < a alors
        g ← m+ 1
        sinon
        d ← m
        fin
fin
si L[g]=a alors
      retourner Vrai
sinon
      retourner Faux
fin
```

1-Montrer que la fonction **F(**d, g) = d - g peut être une fonction de terminaison: on montrera :

- qu'elle est positive pendant l'exécution d'une boucle,
- et qu'elle décroît strictement à chaque itération.

Pour cela on introduit **Fi**(d,g) et **Ff**(d,g) les valeurs initiales et finales de la fonction F au sein d'une répétition de la boucle.

Exercice 5:

Appliquer la méthode précédente avec les deux algorithmes suivants, en déterminant chaque fois une fonction de terminaison adaptée :

```
fonction produit(a,b)
Entrées : deux entiers a et b avec a positif
Sorties : l'entier p
p ← 0;
x ← a;
tant que x > 0 faire
        p ← p + b;
        x ← x - 1;
fin
retourner p
```

```
fonction pgcd(a,b)
Entrées : a et b deux entiers positifs
Sortie : un entier u
u←a;
v ← b;
tant que u ≠ v faire
        si u > v alors
                u ← u - v
        sinon
                v ← v - u
        fin
fin
retourner u
fin
```

Exercice 6:

Parfois, la condition d'arrêt est composée de plusieurs sous-conditions et peut être déterminée par plusieurs variables. Lorsque deux variables entrent en jeu, on peut utiliser le théorème suivant :

Soit un couple (u; v) où u et v sont des expressions contenant les variables utilisées dans une boucle. Supposons que la valeur du couple (u; v) soit décroissante à chaque répétition de la boucle pour l'ordre lexicographique : $(a,b) \leq (a',b')$ si et seulement si $a < a'$ ou $(a = a'$ et $b \leq b')$
Supposons de plus que l'on puisse encadrer u et v respectivement par $minu \leq u \leq maxu$ et $minv \leq\_ v \leq maxv$.

Alors l'exécution de la boucle termine et une bonne fonction de terminaison est :

$$f(u; v) = (u - minu) * (1 + maxv - minv) + v - minv$$

Remarque: $(u', v') < (u; v) \Rightarrow f(u'; v') < f(u, v)$

Montrer en utilisant la fonction f(i; j) = i*n + j comme fonction de terminaison que l'algorithme suivant termine.

```
Entrées : deux entiers strictement positifs m et n
i ← m- 1 ; j ← n - 1;
tant que j ≠ 0 faire
j ← j - 1;
si i ≠0 and j = 0 alors
        i ← i - 1;
        j ← n - 1;
        fin
fin
```

Exercice 7:

En utilisant ce qui précède, démontrer que l'algorithme suivant termine

```
Fonction Rech2D (b,x)
Entrées : une matrice b rectangulaire m x n
un élément x
Sorties : un booléen
i ← 1;
j ←1;
tant que (i ≤m) and (j ≤ n) and (x ≠b[i, j]) faire
        j ← j + 1;
        si j = n + 1 alors
                i ← i + 1;
                j ← 1;
        fin
fin
si i ≤ m alors
        retourner Vrai
sinon
        retourner Faux
fin
```