Terminaison des algorithmes

I Pseudo Langage

Dans ce chapitre, ainsi que les suivants, on considère un langage de programmation restreint. Les instructions que l'on retient sont les suivantes :

- affectation : on la note $X \leftarrow a$ où X est un nom de variable et a est une expression (par exemple une opération arithmétique). a peut contenir la variable X ou non.
- composition : on la note $c_1;c_2$ où c_1 et c_2 sont deux instructions. On remplacera souvent le ; par un passage à la ligne.
- instruction conditionnelle : on la notera **Si** b **alors** c_1 **sinon** c_2 **finsi**, où b est un test (comparaison, égalité) et c_1 et c_2 sont deux instructions. Le **sinon** est optionnel.
- boucle Pour : on la notera **Pour** i de p à q **faire** c **fin** où p et q sont deux entiers et c est une instruction.
- boucle Tant que : on la notera **Tant que** b **faire** c **fin** où b est un test (appelé condition d'arrêt) et c une instruction.
- On admet aussi toutes les opérations classiques (arithmétiques, sur les chaînes, les listes...)

Ce langage restreint, appelé pseudo-langage, va nous permettre d'exprimer tous les algorithmes que vous devez connaitre. On peut commencer à se familiariser avec celui-ci.

1 – On considère les algorithmes suivants. Que font-ils ?

Algorithme 1 :

```
Entrées : n :entier
Sorties : un entier
somme ← 0
pour i de 1 à n faire
    somme ← somme+1
fin
retourner somme
```

Algorithme 2 :

```
Entrées : n :entier
Sorties : une chaine de caractères
chaine ← ""
nombre ← n
tant que nombre ≠ 0 faire
    chaine ← str(nombre%2)+chaine
    nombre ← nombre//2
fin
retourner chaine
```

Algorithme 3 :

```
Entrées : L[1..n] :liste
Sorties : un entier
n ← len(L)
mini ← L[0]
pour i de 1 à n faire
    si L[i] < mini alors
        mini ← L[i]
fin
retourner mini
```

2 – Écrire un algorithme qui prend en argument une liste d'entiers et renvoie l'indice du plus grand élément de cette liste.

3 – Écrire un algorithme qui prend en argument une liste d'entiers et renvoie la moyenne des éléments de cette liste.

4 – Écrire un algorithme qui prend en argument une liste d'entiers et renvoie l'écart type de ses éléments.

5 – Écrire un algorithme qui prend en argument une liste de listes d'entiers ainsi qu'un entier, et qui renvoie un booléen indicant si l'entier fait partie des éléments de la liste.

6 – Ecrire un algorithme qui prend en argument un entier n et qui réalise le calcul de $\sum_{1\leq i,j\leq n} \min(i,j)$.

II Problème de la terminaison

Il est important de pouvoir montrer qu'un programme termine, afin de savoir si son exécution se fera sans problème.

7 – Parmi les instructions retenues, quelles sont celles qui ne peuvent pas engendrer d'exécution infinie d'un algorithme ? Quelle est la seule instruction problématique ?

On suppose maintenant que l'on est capable d'écrire un algorithme, appelé `Terminator`, dont la fonction

est de répondre vrai si un programme termine et faux si un programme ne termine pas. Ainsi, l'instruction `Terminator(P)` renvoie vrai si le programme `P` termine toujours et faux si `P` est capable de boucler.

8 – On considère le programme appelé `SarahConnor` suivant :

Algorithme 4 : SarahConnor

```
tant que Terminator(SarahConnor) faire
    rien
fin
```

À quelle condition le programme `SarahConnor` termine-t-il ?

On comprend donc que le problème de la terminaison des algorithmes n'est pas si simple. On peut énoncer le théorème suivant :

Théorème 1 Problème de l'arrêt

Il n'existe pas de programme permettant de dire si un algorithme termine toujours ou non.

On dit en théorie de l'informatique que le problème de l'arrêt est indécidable. En fait, le théorème de Rice dit même que toute propriété non triviale sur les programmes est indécidable. Bien que ce soit vrai dans le cas général, on peut cependant s'intéresser à des cas particuliers et utiliser des propriétés mathématiques pour démontrer que des algorithmes simples terminent.

Tout d'abord, donnons un contre exemple fondé sur la suite de Syracuse (algorithme 5)

Algorithme 5 : 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
```

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. Mais il n'existe pour le moment aucune théorie mathématique permettant d'assurer que l'on arrive toujours à 1 : il est impossible de dire si cet algorithme termine ou non.

III Démontrer la terminaison

On commence par étudier l'algorithme très simple suivant :

Algorithme 6 : Algorithme très naïf

```
i ← 0 tant que i < 5 faire
    i ← i + 1
fin
```

9 – Essayer d'expliquer pourquoi cet algorithme termine.

10 – On pose $f(i) = 5 - i$. Que peut-on dire de cette fonction à chaque tour de boucle ?

11 – Donner un minorant de f lors de l'exécution de l'algorithme. Que peut-on en déduire ?

Toutes les démonstrations de terminaison se dérouleront de cette façon. L'idée principale est de trouver une fonction que l'on appellera fonction de terminaison ou variant de boucle :

Définition 1 : Fonction de terminaison

Une *fonction de terminaison* pour une boucle est une fonction positive qui dépend des variables de l'algorithme et dont la valeur 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.

Ainsi, la fonction étant strictement décroissante et positive, on sait qu'elle passe sous n'improte quelle valeur en un temps fini : à ce moment là, la condition d'arrêt de la boucle se déclenche et le programme termine.

On considère l'algorithme de recherche dichotomique :

Algorithme 7 : 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
```

12 – Faire tourner l'algorithme 7 sur les deux entrées suivantes : $([1,3,5,7,8],4)$ et $([1,3,5,7,8],6)$. Que fait cet algorithme ?

13 – Montrer que la fonction $f(d,g) = d - g$ est une bonne fonction de terminaison : on montrera qu'elle est positive pendant l'exécution d'une boucle, et qu'elle décroît à chaque itération. Pour cela on introduit $f_i(d,g)$ et $f_f(d,g)$ les valeurs initiales et finales de la fonction au sein d'une répétition de la boucle.

14 – Faire le même travail avec les deux algorithmes suivants, en déterminant cette fois une fonction de terminaison adaptée :

Algorithme 8 : 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
```

Algorithme 9 : fonction pgcd(a,b)

```
Entrées : a et b deux entiers positifs
Sorties : 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
```

IV Condition d'arrêt avancée

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 par exemple utiliser le théorème suivant :

Théorème 2

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. 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 :

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

Un résultat similaire peut être établi pour tout type de tuple.

15 – Montrer ce théorème, c'est à dire que si $(u', v') <_{lex} (u, v)$, alors $f(u', v') < f(u, v)$.

16 – En s'inspirant de ce théorème, démontrer que l'algorithme suivant termine. Que fait cet algorithme ?

Algorithme 10 : Fonction Rech2D (b,x)

```
Entrées : une matrice b rectangulaire m × n
un élément x
Sorties : un booléen
i ← 1;
j ← 1;
tant que (i ≤ m) ∧ (j ≤ n) ∧ (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
```

17 – Même question avec l'algorithme suivant. On montrera que $f(i, j) = i*n+j$ est une fonction de terminaison.

Algorithme 11 : Un exemple de fonction de terminaison

```
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 ∧ j = 0 alors
        i ← i − 1;
        j ← n − 1;
    fin
fin
```

18 – Trouver dans l'algorithme suivant le couple (u, v) décroissant et donner une fonction de terminaison.

Algorithme 12 :

```
Entrées : deux entiers strictement positifs m et n
i ← 0 ; j ← 0
tant que (i < n) ∨ (j < m) faire
    si (i < 0) ∨ (j > m) alors
        j ← 0
        i ← i + j + 1
    fin
    si i > n alors
        j ← i − n
        i ← n
    fin
    j ← j + 1
    i ← i − 1
fin
```