TD n° 2 : Preuve d'algorithmes

EXERCICE 1 On considère la fonction puissance du cours, légèrement modifiée pour rendre les preuves plus aisées :

PYTHON
```
def puissance(k, n):
    """Calcule k^n pour deux entiers k, n >= 0."""
    i = n
    res = 1
    while i > 0:
        res = k * res
        i -= 1
    return res
```

1. Trouver un variant pour la boucle et justifier que ce programme termine.
2. Montrer que « res $\times k^i = k^n$ et $i \geq 0$ » est un invariant de la boucle.
3. Justifier la correction de ce programme.

EXERCICE 2 La fonction suivante vérifie l'appartenance d'un élément à une liste :

PYTHON
```
def appartient(element, liste):
    n = len(liste)
    i = 0
    while i < n and liste[i] != element:
        i += 1
    return i < n
```

1. Trouver un variant de boucle qui permet de montrer la terminaison de cette fonction.
2. Montrer que « $i \leq n$ et element $\notin$ liste$[:i]$ » est un invariant de la boucle.
3. Montrer la correction de la fonction.

EXERCICE 3 Cette fonction qui renvoie le maximum d'une liste non vide :

PYTHON
```
def maximum(liste):
    """Renvoie l'élément maximal d'une liste non vide."""
    n = len(liste)
    maxi = liste[0]
    for i in range(1, n):
        if liste[i] > maxi:
            maxi = liste[i]
    return maxi
```

1. Justifier la terminaison de cette fonction.
2. Montrer que « maxi $= \max_{k \in [\![0, i-1]\!]}$ liste$[k]$ » est un invariant de la boucle **for**.
3. Justifier la correction de cette fonction.
4. Réécrire cette fonction avec une boucle **while**. Montrer sa terminaison et sa correction.

EXERCICE 4 Écrire en PYTHON avec une boucle **for** sur les indices, puis avec une boucle **while**, et justifier dans les deux cas la terminaison et la correction, des fonctions :

1. `nb_occurrences(element, liste)` qui compte le nombre d'occurrences d'un élément dans une liste ;
2. `miroir(mot)` qui renvoie le mot miroir (*i.e.* lu à l'envers) d'une chaîne de caractères. Comment écrire cette fonction en une ligne avec des tranches ?
3. `fibonacci(n)` qui renvoie le terme d'indice n de la suite de Fibonacci ;
4. `est_fibonacci(n)` qui vérifie si l'entier n appartient à la suite de Fibonacci (boucle **while** seulement).

EXERCICE 5 *Exercice corrigé*

Justifier la terminaison et la correction des algorithmes ci-dessous. On énoncera précisément et on prouvera les variants et invariants utilisés. On détaille le tout premier exemple. Il faut savoir le faire, même si en général on ne donnera pas autant de détails.

1. Fonction qui détermine le rang du dernier terme strictement positif de la suite récurrente définie par $u_{n+1} = \frac{1}{2}u_n - 3n$ de premier terme $u_0 > 0$ donné en argument.

PYTHON

```
def rang_dernier_strictement_positif(u_0):
    u = u_0
    n = 0
    # u = u_0 et pour i < 0, u_i > 0
    while u > 0:
        # u = u_n et pour i < n, u_n > 0
        # donc u_n > 0, donc pour i < n + 1, u_n > 0
        u = u/2 - 3*n
        # u = u_{n + 1} et pour i < n + 1, u_n > 0
        n += 1
        # u = u_n et pour i < n, u_n > 0
    return n - 1
```

On propose l'invariant « $u = u_n$ et $\forall i \in [\![0; n-1]\!],\ u_i > 0$ » pour la boucle `while`.

- **Initialisation :** Avant d'entrer dans la boucle, $n = 0$, $u = u_0$ et la condition $\forall i \in \emptyset,\ u_i > 0$ est bien sûr vérifiée.
- **Conservation :** Supposons l'invariant vrai au début d'un passage dans la boucle et donc $u = u_n$ et $\forall i \in [\![0; n-1]\!], u_n > 0$ Comme on entre dans la boucle, $u > 0$, c'est-à-dire $u_n > 0$ et donc $i \in [\![0; n]\!], u_n > 0$. Comme $u = u_n$, $\frac{u}{2} - 3u$ vaut u_{n+1} et donc, après l'affectation, $u = u_{n+1}$. Après décrément de n, on retrouve donc bien les deux conditions de l'invariant.
- **Correction :** À la sortie de la boucle, l'invariant donne $u = u_n$ et $\forall i \in [\![0; n-1]\!], u_n > 0$. La négation de la condition de boucle donne $u \leq 0$ et donc $u_n \leq 0$. Ceci montre que n est le premier indice tel que u_n soit négatif (et donc $n > 0$ car $u_0 > 0$). De plus, si pour $k \in \mathbb{N}$, $u_k \leq 0$ alors $u_{k+1} = \frac{u_k}{2} - 3k \leq 0$ et on montre que $\forall i \geq k,\ u_i \leq 0$. On a donc bien montré donc que $n-1$ est le rang du dernier terme strictement positif.

☞ **Remarque 1**

On a simplement montré que si l'algorithme termine, alors il renvoie le bon résultat.

Pour le variant, on montre d'abord qu'il existe $N > 0$ tel que $u_N \leq 0$. En effet, si $\forall n \in \mathbb{N}, u_n > 0$, comme $\forall n \in \mathbb{N}, u_{n+1} - u_n = -\frac{u_n}{2} - 3n < 0$, la suite serait strictement décroissante. Pour $n \geq \frac{u_0}{6}, u_{n+1} \leq \frac{u_n}{2} - \frac{u_0}{2} \leq 0$, ce qui contredit l'hypothèse. On propose alors comme variant $N - n$. Pour montrer que c'est bien un variant, on a besoin de l'invariant $u = u_n$, qu'il faut donc avoir traité avant.

- **Terminaison :** Montrons que $N - n$ est un variant. Il est clair que cette quantité décroît strictement (de 1) à chaque passage dans la boucle. D'après l'invariant, $\forall i \in [\![0; n-1]\!], u_i > 0$ et donc $n \leq N$ et on a donc bien $N - n \in \mathbb{N}$.

2. Fonction qui calcule le logarithme itéré défini pour $x > 0$ par :

$$\log^*(x) = \min\left\{i \geq 0 \mid \log_2^{(i)}(x) \leq 1\right\}$$

PYTHON

```
def log_itere(x):
    # x_0 = x
    k = 0
    # k >=0, x = x_0
    while x > 1:
        # k >= 0, x = log_2^(k)(x_0), pour i < k,
        ↪ log2^(i)(x_0) > 1
        # Comme x > 1, on a pour i < k + 1, log2^(k)(x_0)
        ↪ > 1
        x = math.log2(x)
        # x = log_2^(k + 1)(x_0)
        k += 1
        # k >= 0, x = log_2^(k)(x_0), pour i < k,
        ↪ log2^(i)(x_0) > 1
    return k
```

Attention, ici, on modifie l'argument. Il faut noter sa valeur initiale pour pouvoir écrire l'invariant. On pose x_0 la valeur initiale de x. Invariant : « $k \geq 0$ et $x = \log_2^{(k)}(x_0)$ et $\forall i \in [\![0, k-1]\!], \log_2^{(i)}(x_0) > 1$. La terminaison revient à montrer que la fonction $\log^*$ est bien définie. Pour $x \in]0;1]$, $\log^*(x) = 0$, pour $x \in]1;2]$, $\log^*(x) = 1$. Pour $x > 2$ on montre par exemple que $\log_2(x) < x - 1$ et donc que pour $i \geq 1$, si $\log_2^{(i)}(x) > 1$, alors $\log_2^{(i)}(x) < x - i$. Donc, si $i = \lfloor x \rfloor$ alors $\exists k \leq i, \log_2^{(k)}(x) \leq 1$. On peut alors prendre comme variant $\lfloor x_0 \rfloor - k$, par exemple.