La preuve en informatique

Preuve de terminaison

- Si la suite d'instructions n'est pas récursive et n'utilise pas de boucle, ou seulement des boucles for s’exécutant un nombre fini de fois, alors elle termine.
- Dans le cas contraire, on s’appuiera sur les propriétés suivantes :
  - Une suite d’entiers naturels strictement décroissante est finie.
  - Une suite d’entiers naturels strictement croissante tend vers $+\infty$ (et sera supérieur à n’importe quel nombre A au bout d’un certain rang).
- On appelle convergent, ou parfois « variant » de boucle, une expression du type entier naturel qui décroit strictement à chaque exécution d’une boucle ou lors de chaque appel récursif.

Preuve de terminaison : exemples d'algorithmes qui ne se terminent pas

```
def algostupide():
    n = 10
    while n > 0:
        n = n + 1
```

```
def algostupide2():
    lst = [1]
    for i in lst:
        lst.append(1)
```

Exemple de preuve de terminaison : algorithme d'Euclide

Exemple :

```
def pgcd(a,b) :
    "Calcule le pgcd de a et b"
    while b != 0 :
        a, b = b, a%b
    return a
```

Exemple de preuve de terminaison : algorithme d'Euclide

Exemple :

```
def pgcd(a,b) :
    "Calcule le pgcd de a et b"
    while b != 0 :
        a, b = b, a%b
    return a
```

Dans l'algorithme d'Euclide appliqué à deux entiers positifs *a* et *b*, la boucle s'exécute tant que $b \neq 0$, et *b* est un entier qui décroit strictement à chaque exécution, donc va forcément finir par prendre la valeur 0. L'algorithme va donc s'arrêter.

Exemple de preuve de terminaison : algorithme d'Euclide

Exemple :

```
def pgcd(a,b) :
    "Calcule le pgcd de a et b"
    while b != 0 :
        a, b = b, a%b
    return a
```

Dans l'algorithme d'Euclide appliqué à deux entiers positifs a et b, la boucle s'exécute tant que $b \neq 0$, et b est un entier qui décroit strictement à chaque exécution, donc va forcément finir par prendre la valeur 0. L'algorithme va donc s'arrêter.

b décroit strictement à chaque exécution, car b est remplacé à chaque étape par le reste r de la division euclidienne de a par b et qu'il vérifie $a = bq + r$ avec $r < b$.

Exemple de preuve de terminaison/correction : le tri à bulles

```
def tribulles(lst):
    pastrie = True
    while pastrie:
        pastrie = False
        for i in range(len(lst)-1):
            if lst[i] > lst[i+1]:
                lst[i], lst[i+1] = lst[i+1], lst[i]
                pastrie = True
    return lst
```

Exemple de preuve de terminaison/correction : le tri à bulles

```
def tribulles(lst):
    pastrie = True
    while pastrie:
        pastrie = False
        for i in range(len(lst)-1):
            if lst[i] > lst[i+1]:
                lst[i], lst[i+1] = lst[i+1], lst[i]
                pastrie = True
    return lst
```

Expliquer en français le principe de cet algorithme de tri.

Comment peut-on être sûr qu'il se termine ? et qu'il renvoie bien la liste triée ?

Exemple de preuve de terminaison : le tri à bulles

```
def tribulles(lst):
    pastrie = True
    while pastrie:
        pastrie = False
        for i in range(len(lst)-1):
            if lst[i] > lst[i+1]:
                lst[i], lst[i+1] = lst[i+1], lst[i]
                pastrie = True
    return lst
```

Comment peut-on être sûr qu'il se termine ?

On cherche un « variant de boucle », un entier qui diminue à chaque exécution de la boucle.

Exemple de preuve de terminaison : le tri à bulles

```
def tribulles(lst):
    pastrie = True
    while pastrie:
        pastrie = False
        for i in range(len(lst)-1):
            if lst[i] > lst[i+1]:
                lst[i], lst[i+1] = lst[i+1], lst[i]
                pastrie = True
    return lst
```

On considère le nombre d'inversions, c'est-à-dire le nombre de couples (m, n) où lst[m] > lst[n].

Exemple de preuve de terminaison : le tri à bulles

```
def tribulles(lst):
    pastrie = True
    while pastrie:
        pastrie = False
        for i in range(len(lst)-1):
            if lst[i] > lst[i+1]:
                lst[i], lst[i+1] = lst[i+1], lst[i]
                pastrie = True
    return lst
```

À chaque permutation, ce nombre d'inversions diminue. S'il ne change pas lors d'une exécution de la boucle for, c'est que l'algorithme n'a pas effectué de permutation, et pastrie garde la valeur False et l'algorithme se termine.

Exemple de preuve de terminaison : le tri à bulles

```
def tribulles(lst):
    pastrie = True
    while pastrie:
        pastrie = False
        for i in range(len(lst)-1):
            if lst[i] > lst[i+1]:
                lst[i], lst[i+1] = lst[i+1], lst[i]
                pastrie = True
    return lst
```

De plus, le nombre d'inversions ne peut pas diminuer indéfiniment puisque c'est un nombre entier, minoré par 0, donc l'algorithme se termine nécessairement.

Exemple de preuve de terminaison/correction : le tri à bulles

```
def tribulles(lst):
    pastrie = True
    while pastrie:
        pastrie = False
        for i in range(len(lst)-1):
            if lst[i] > lst[i+1]:
                lst[i], lst[i+1] = lst[i+1], lst[i]
                pastrie = True
    return lst
```

Lorsqu'il se termine, cela signifie que pour tout i, lst[i] < lst[i+1], donc la liste est triée.

Exemple de preuve de correction : le tri à bulles

```
def tribulles(lst):
    pastrie = True
    while pastrie:
        pastrie = False
        for i in range(len(lst)-1):
            if lst[i] > lst[i+1]:
                lst[i], lst[i+1] = lst[i+1], lst[i]
                pastrie = True
    return lst
```

On peut rédiger cette preuve avec des notations plus mathématiques en utilisant un invariant de boucle.
On note n_i le nombre d'inversions après *i* exécutions de la boucle while.

Que peut-on prendre comme invariant de boucle ?

Exemple de preuve de correction : le tri à bulles

```
def tribulles(lst):
    pastrie = True
    while pastrie:
        pastrie = False
        for i in range(len(lst)-1):
            if lst[i] > lst[i+1]:
                lst[i], lst[i+1] = lst[i+1], lst[i]
                pastrie = True
    return lst
```

On note n_i le nombre d'inversions après i exécutions de la boucle while.

On peut prendre comme invariant de boucle :
« $n_{i+1} \leq n_i$ »,
tant que $i + 1$ ne dépasse par le nombre d'exécutions de la boucle.

Exercice 2 : surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Exercice 2 : surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Que renvoie surprise(a,n) ?

Exercice 2 : terminaison de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Que peut-on prendre comme variant de boucle?

Exercice 2 : terminaison de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Que peut-on prendre comme variant de boucle?

La variable *n*.

Exercice 2 : terminaison de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Que peut-on prendre comme variant de boucle?

La variable *n*.

En effet, la valeur de l'entier naturel *n* décroit strictement à chaque exécution de la boucle, donc la boucle se termine, lorsque *n* = 0, après *k* exécutions.

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Que peut-on prendre comme invariant de boucle?

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Que peut-on prendre comme invariant de boucle?

On cherche une relation entre les variables *a*, *n*, *r* vraie à chaque exécution de la boucle, et en lien avec ce qu'on veut démontrer.

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Que peut-on prendre comme invariant de boucle?

On cherche une relation entre les variables *a*, *n*, *r*.

À la fin, on veut que le « r final » soit égal au « a^n *initial* ».

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Introduisons des notations.

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Introduisons des notations.

Soient pour tout entier naturel i inférieur ou égal à k, a_i, n_i, r_i les valeurs des variable a, n et r après la i-ième exécution de la boucle, en notant a_0, n_0 et r_0 les valeurs initiales.

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Soient pour tout entier naturel i inférieur ou égal à k, a_i, n_i, r_i les valeurs des variable a, n et r après la i-ième exécution de la boucle, en notant a_0, n_0 et r_0 les valeurs initiales.

Invariant de boucle : quelle relation en lien avec ce qu'on veut démontrer vérifient a_i, n_i, r_i à chaque exécution de la boucle?

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle : quelle relation en lien avec ce qu'on veut démontrer vérifient a_i, n_i, r_i à chaque exécution de la boucle?

On veut trouver a^n, soit $a_0^{n_0}$ avec les valeurs initiales.

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle : quelle relation en lien avec ce qu'on veut démontrer vérifient a_i, n_i, r_i à chaque exécution de la boucle?

On veut trouver a^n, soit $a_0^{n_0}$ avec les valeurs initiales.

Si *n* diminue de 1, *r* est multiplié par *a.*

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle : quelle relation en lien avec ce qu'on veut démontrer vérifient a_i, n_i, r_i à chaque exécution de la boucle?

On veut trouver a^n, soit $a_0^{n_0}$ avec les valeurs initiales.

Si *a* est élevé au carré, *n* est divisé par 2.

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle : quelle relation en lien avec ce qu'on veut démontrer vérifient a_i, n_i, r_i à chaque exécution de la boucle?

Invariant de boucle :

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle : quelle relation en lien avec ce qu'on veut démontrer vérifient a_i, n_i, r_i à chaque exécution de la boucle?

Invariant de boucle :

pour tout $i \in [[0, k]]$,

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle : quelle relation en lien avec ce qu'on veut démontrer vérifient a_i, n_i, r_i à chaque exécution de la boucle?

Invariant de boucle :

pour tout $i \in [[0, k]]$, $a_i^{n_i} r_i = a_0^{n_0}$

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle :

pour tout $i \in [[0, k]]$, $a_i^{n_i} r_i = a_0^{n_0}$

Une fois l'invariant de boucle formulé, le plus dur est fait.

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle :

pour tout $i \in [[0, k]]$, $a_i^{n_i} r_i = a_0^{n_0}$

Initialisation : Vrai pour $n = 0$.

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle :

pour tout $i \in [[0, k]]$, $a_i^{n_i} r_i = a_0^{n_0}$

Hérédité : Supposons la propriété précédente vraie pour un entier i.

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle :

pour tout $i \in [[0, k]]$, $a_i^{n_i} r_i = a_0^{n_0}$

Hérédité : Supposons la propriété précédente vraie pour un entier i.

Si n_i est impair, alors $r_{i+1} = r_i \times a$,

$n_{i+1} = n_i - 1$

et $a_{i+1} = a_i$.

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle :

pour tout $i \in [[0, k]]$, $a_i^{n_i} r_i = a_0^{n_0}$

Hérédité : Supposons la propriété précédente vraie pour un entier i.

Si n_i est impair, alors $r_{i+1} = r_i \times a$, $n_{i+1} = n_i - 1$

et $a_{i+1} = a_i$.

$a_{i+1}^{n_{i+1}} r_{i+1} = a_i^{n_i - 1} r_i \times a = a_i^{n_i} r_i = a_0^{n_0}$
d'après l'hypothèse de récurrence.

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle :

pour tout $i \in [[0, k]]$, $a_i^{n_i} r_i = a_0^{n_0}$

Hérédité : Supposons la propriété précédente vraie pour un entier i.

Si n_i est pair, alors $r_{i+1} = r_i$,

$$a_{i+1} = a_i^2,$$

$$n_{i+1} = \frac{n_i}{2}.$$

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle :

pour tout $i \in [[0, k]]$, $a_i^{n_i} r_i = a_0^{n_0}$

Hérédité : Supposons la propriété précédente vraie pour un entier i.

Si n_i est pair, alors $r_{i+1} = r_i$,

$$a_{i+1} = a_i^2,$$

$$n_{i+1} = \frac{n_i}{2}.$$

$$a_{i+1}^{n_{i+1}} r_{i+1} = \left(a_i^2\right)^{\frac{n_i}{2}} r_i = a_i^{n_i} r_i = a_0^{n_0}.$$

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle :

pour tout $i \in [[0, k]]$, $a_i^{n_i} r_i = a_0^{n_0}$

Hérédité : Supposons la propriété précédente vraie pour un entier i.

L'hérédité est vérifiée dans les deux cas.

L'invariant de boucle est donc établi pour tout entier $i \in [[0, k]]$.

Exercice 2 : correction de surprise(a,n)

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

Invariant de boucle :

pour tout $i \in [[0, k]]$, $a_i^{n_i} r_i = a_0^{n_0}$

Quand l'algorithme se termine, $n_i = 0$, et donc $r_i = a_0^{n_0}$.