La preuve en informatique

Malgré de nombreux tests réussis, les applications professionnelles sont souvent publiées avec divers bugs qui ont échappé aux tests. Le succès d'un jeu de tests ne garantit pas la correction d'un programme.

L'idéal est de prouver le fonctionnement du code avec :

- une preuve de terminaison : preuve qu'une boucle non bornée (while) ou une fonction récursive se termine.
- une preuve de correction : preuve qu'un algorithme donne bien le résultat attendu

La combinaison des deux permet d'affirmer que le code fonctionne.

1. Preuve de terminaison

On s'appuiera souvent 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 **variant** l'expression qui va garantir l'arrêt du code, souvent un entier naturel strictement décroissant.

Remarque : On ne sait pas toujours prouver qu'une boucle s'arrête. Un des exemples les plus célèbres est la conjecture de Syracuse, problème non résolu à ce jour, qui mobilisa tant les mathématiciens durant les années 1960, en pleine guerre froide, qu'une plaisanterie courut selon laquelle il faisait partie d'un complot soviétique visant à ralentir la recherche américaine.

2. Preuve de correction

Là aussi, la difficulté vient des boucles, et des appels récursifs. Pour le reste, il suffit de suivre l'algorithme ligne par ligne.
On s'appuie sur un **invariant** : une proposition qui reste vraie à chaque exécution d'une boucle (ou à chaque appel récursif). On l'exprime en général en fonction du nombre n d'exécutions de la boucle et on utilise une preuve par récurrence.

Plusieurs rédactions sont possibles. Le plus souvent, on ajoute un indice aux noms de variables affectées par la boucle : ainsi, on note a_n la valeur de la variable a lors de la n-ième exécution de la boucle (ou lors du n-ième appel de la fonction dans le cas d'une fonction récursive).

3. Une fonction itérative

```
def fact(n):
    f = 1
    while n >=1 :
        f = f * n
        n = n - 1
    return f
```

Preuve de terminaison
La boucle while s'exécute tant que $n \geq 1$ et n est un entier qui est décrémenté à chaque exécution. Elle va donc se terminer.

Preuve de correction

Notons f_k et n_k les valeurs des variables f et n après k exécutions de la boucle while. Alors les valeurs initiales de ces variables sont $f_0 = 1$ et $n_0 = n$.

Un invariant de boucle est, pour $0 \leq k \leq n$: « $n_k = n - k$ et $f_k = \frac{n!}{(n-k)!}$ ».

On le prouve par récurrence sur k.

Initialisation : pour $k = 0$, on a bien $n_0 = n$ et $f_0 = \frac{n!}{n!} = 1$.

Hérédité : supposons que pour un entier k compris entre 0 et $n - 1$ on a $n_k = n - k$ et $f_k = \frac{n!}{(n-k)!}$.

Alors, après une exécution de plus de la boucle while, on a $f_{k+1} = \frac{n!}{(n-k)!} \times n_k = \frac{n!}{(n-k)!} \times (n-k) = \frac{n!}{(n-(k+1))!}$

et $n_{k+1} = n_k - 1 = n - (k + 1)$.

Donc, la propriété est vraie au rang $k + 1$.

Par récurrence, la propriété est vraie pour tout entier k compris entre 0 et n.

En particulier, la boucle se termine lorsque $n_k = 0$, donc après $k = n$ exécutions.

On a alors $f_n = \frac{n!}{(n-n)!} = \frac{n!}{0!} = n!$.

4. Une fonction récursive

```
def fact(n):
    if n==0:
        return 1
    else:
        return n*fact(n-1)
```

Preuve de terminaison

La variable n est un entier qui est décrémenté à chaque nouvel appel. Il va donc forcément finir par prendre la valeur 0, ce qui va mettre fin aux appels récursifs.

Preuve de correction

Prenons comme invariant pour $n \in \mathbb{N}$: l'appel de fact(n) renvoie n!

Prouvons-le par récurrence.

Initialisation : pour $n = 0$, fact(0) renvoie 1, qui est bien égal à 0 !

Hérédité : supposons que fact(n) renvoie n! pour un entier naturel n.

Alors n+1 ≠ 0 donc fact(n+1) renvoie (n+1) × fact(n) = (n+1) × n! = (n+1)!

L'initialisation et l'hérédité sont vérifiées, donc par récurrence, pour tout $n \in \mathbb{N}$, fact(n) renvoie bien n!.

En NSI, on n'attend pas que vous sachiez rédiger une preuve de correction, d'autant plus que certains élèves de NSI ne suivent pas la spécialité mathématiques et n'ont jamais vu de raisonnement par récurrence. Vous devez néanmoins connaitre et comprendre la notion de variant (servant à justifier la terminaison d'un script itératif ou récursif) et la notion d'invariant servant à prouver la correction d'un algorithme.