**TP 7 - Preuves par récurrence**

# 1 Définir des fonctions récursives

*Avant de commencer à faire des preuves par récurrence, nous allons avoir besoin de voir quelques points supplémentaires de syntaxe en LEAN.*

## 1.1 Définition de fonctions par correspondance de motifs

*En LEAN, il est possible – dans beaucoup de situations – de définir une fonction en faisant une disjonction de cas sur l'entrée.*

*Par exemple, on peut définir une fonction* $f : \mathbb{N} \to \mathbb{N}$ *qui renvoie* 0 *si l'entrée* $n$ *vaut* 0*, et* $n-1$ *dans les autres cas, de la façon suivante :*

```
def moins_un : Nat→Nat
  | 0 => 0     -- si l'entree vaut 0, on renvoie 0
  | n + 1 => n     -- si l'entree est de la forme n + 1, avec n dans Nat, on renvoie n.
```

**1.** Tapez le code précédent, et affichez la valeur prise par la fonction sur divers entiers naturels (commande `#eval`).

*Dans l'exemple précédent, on a une disjonction entre deux possibilités sur la valeur de l'entrée. On peut aussi écrire des fonctions plus compliquées faisant appel à plus de cas possibles.*

**2.** Définir une fonction `exemple : Nat→Nat`, qui prend en entrée un entier naturel, et qui renvoie 1 si l'entier vaut 0, 0 si l'entier vaut 1, et $2n$ si l'entier est de la forme $n+2$. Affichez les valeurs prises par cette fonction sur quelques exemples.

*Il n'est pas possible de mettre n'importe quel type de formule comme possibilité pour l'entrée, mais LEAN fonctionne bien pour des formules comme* `n+1`*,* `n+2`*, etc. comme on l'a fait ci-dessus.*

*Attention, il ne faut pas oublier de traiter tous les cas possibles en entrée :*

**3.** Que se passe-t-il si l'on essaie le code suivant ?

```
def exemple_fautif : Nat→Nat
  | 0 => 12
  | n + 3 => 3 * n
```

## 1.2 Fonctions récursives

*En LEAN, il est autorisé pour une fonction de s'appeler elle-même : si c'est le cas, on dit que la fonction est* récursive. *En utilisant la syntaxe vue à la section précédente, cela permet de définir facilement beaucoup de fonctions intéressantes.*

**4.** Que fait la fonction suivante ?

```
def factorielle : Nat→Nat
  | 0 => 1
  | n + 1 =>  (n+1) * (factorielle n)
```

Calculez quelques valeurs de cette fonction.

**5.** Définissez une fonction `somme_entiers : Nat→Nat` telle que `somme_entiers n` renvoie $\sum_{j=0}^{n} j$. De même, définissons une fonction `somme_carres` qui à $n \in \mathbb{N}$ associe la somme $\sum_{j=0}^{n} j^2$.

**6.** Définissez une fonction `est_pair : Nat→Bool` qui à $n \in \mathbb{N}$ associe `true` si $n$ est pair, et `false` si $n$ est impair.
*(Indication : faites comme à la question* **2** *et traitez d'abord les cas de base où $n = 0$ ou $n = 1$)*

*Attention, LEAN n'acceptera pas de fonction récursive s'il y a un risque de déclencher une boucle infinie :*

**7.** A votre avis, quel est le problème avec la fonction suivante ? Que se passe-t-il si on essaie de la taper en LEAN ?

```
def boucle_infinie : Nat→Nat
  | 0      =>  0
  | n + 1  =>  boucle_infinie (n+1)
```

# 2 Preuves par récurrences

*Reportez vous à l'aide-mémoire : une section a été ajoutée concernant les preuves par récurrence.*

*On va se servir de LEAN pour démontrer la formule donnant la somme des $n$ premiers entiers. Pour cela, on va procéder en deux étapes :*

**8.** En utilisant la fonction `somme_entiers` définie à la question **5.**, écrire l'énoncé d'un théorème `thm_somme_entiers_v2`, affirmant que pour tout $n \in \mathbb{N}$, on a :

$$2\sum_{j=0}^{n} j = n(n+1).$$

Démontrer le théorème, et écrire en parallèle la preuve au papier. Dans l'étape d'hérédité, vous pourrez utiliser un bloc « `calc` » pour écrire le calcul pertinent (vous avez le droit d'utiliser les commandes `ring` ou `simp`).

*(Indication : dans l'étape d'hérédité, commencez par réécrire la fonction* `somme` *avec la commande* `rw` *: cela fera apparaître sa définition par récurrence. Vous pouvez ensuite écrire le bloc* `calc`*. Si vous vous débrouillez bien, trois lignes suffisent !)*

**9.** Ecrire maintenant un théorème `somme_entiers_final` affirmant que pour tout $n \in \mathbb{N}$, on a

$$\sum_{j=0}^{n} j = \frac{n(n+1)}{2}.$$

Dans la preuve, après avoir introduit $n \in \mathbb{N}$, vous pouvez utiliser un bloc `calc`, et essayer de faire apparaître `2* somme n` à la première ligne. Vous pourrez ensuite utiliser le théorème de la question **8**.

**10.** Procéder exactement de la même manière que précédemment pour démontrer la formule donnant la somme des $n$ premiers carrés, puis des $n$ premiers cubes.

**11.*** Utiliser ce qui précède pour écrire en LEAN une preuve du résultat suivant : pour tout $n \in \mathbb{N}$, on a :

$$1^3 + 2^3 + 3^3 + \ldots + n^3 = (1 + 2 + 3 + \ldots + n)^2.$$

*On maintenant voir comment employer LEAN pour faire une récurrence double.*

**12.** Définissez une fonction récursive `suite_double : Nat → Int`, telle que pour tout entier `n`, `suite_double n` renvoie le $n$-ième terme de la suite définie par récurrence de la façon suivante :

$$\begin{cases} u_0 &= 2 \\ u_1 &= 5 \\ u_{n+2} &= 5u_{n+1} - 6u_n. \end{cases}$$

**13.** Définissez un prédicat `P` portant sur un entier $n \in \mathbb{N}$ de sorte que « `P n` » soit la proposition suivante :

$$P(n) : \quad u_n = 3^n + 2^n.$$

**14.** Définissez un théorème `recurrence_double` affirmant que pour tout $n \in \mathbb{N}$, la proposition

$$P(n) \wedge P(n+1)$$

est vraie. Démontrer ensuite ce théorème, et écrire la preuve au papier en parallèle.

*(Petit conseil : faites le calcul final sur papier avant de le faire sur machine. Encore une fois, vous pourrez utiliser un bloc `calc`. Si vous vous débrouillez bien, les tactiques `ring` et `simp` suffiront. Vous pouvez aussi utiliser la tactique `omega` à la place de `simp` : c'est aussi une commande de simplification, qui est assez efficace.)*

**15.** Enfin, écrire et démontrer un théorème `formule_suite` affirmant que pour tout $n \in \mathbb{N}$, on a $u_n = 3^n + 2^n$.

**16*.** Imitez la méthode précédente pour montrer que la suite définie par récurrence de la façon suivante :

$$\begin{cases} v_0 &= 3 \\ v_1 &= 1 \\ v_2 &= 9 \\ v_{n+3} &= v_{n+2} + 4v_{n+1} - 4v_n. \end{cases}$$

a pour terme général

$$v_n = 1 + 2^n + (-2)^n.$$