UNIVERSITÉ Grenoble Alpes

INF201
Algorithmique et Programmation Fonctionnelle
Cours 4: Fonctions et types récursifs

Année 2018 - 2019

Les précédents épisodes de INF 201

- Types de base : `bool`, `int`, `float`, `char`
- `if` ... `then` ... `else` ... expression conditionnelle
- identificateurs (locaux et globaux)
- définition, utilisation et composition de fonctions
- types avancés : synonyme, énuméré, produit, somme
- filtrage et pattern-matching

Plan

Définition de fonctions en OCaml

Rappels

```
let fct_name (p1:t1) (p2:t2) ... (pn:tn) : t = expr
```

Exemple :

```
let max2 (x: int) (y: int) : int = if x > y then x else y
```

Définition de fonctions en OCaml

Rappels

```
let fct_name (p1:t1) (p2:t2) ... (pn:tn) : t = expr
```

Exemple :

```
let max2 (x: int) (y: int) : int = if x > y then x else y
```

Une fonction OCaml

- des paramètres formels (optionnels) : `x` et `y`
- une valeur = l'expression correspondant au corps de la fonction (qui permet de calculer le résultat)

  `if x > y then x else y`

- un type, qui dépend :
  - du type des paramètres de la fonction
  - du type du résultat

  $\texttt{int} \rightarrow \texttt{int} \rightarrow \texttt{int}$

- un nom (optionnel) : `max2`

Exemple :

- (`max2` 5 9) (les parenthèses peuvent *parfois* être omises)
- (`max2` (`min2` 3 8) 9)
- (`max2` (5 − 4) (8 + 3))

Exemple :

- (`max2` 5 9) (les parenthèses peuvent *parfois* être omises)
- (`max2` (`min2` 3 8) 9)
- (`max2` (5 − 4) (8 + 3))

Plus généralement :

```
let fct_name (p1:t1) (p2:t2) ... (pn:tn) : t = expr
```

- le type de `fct_name` est `t1` → `t2` → ... → `tn` → `t`
- appel de la fonction `fct_name` à *n* paramètres :

  (`fct_name e1 e2 ... en`)

  où :

  - `e1`,...,`en` sont les paramètres effectifs
  - chaque paramètre effectif `ei` a pour type `ti`
  - Le type de (`fct_name e1 e2 ... en`) est `t`

Fonctions: SPECIFICATION et IMPLEMENTATION

Il est très important de distinguer 2 concepts/étapes lors de la définition d'une fonctions (et d'un programme en général)

Fonctions: SPECIFICATION et IMPLEMENTATION

Il est très important de distinguer 2 concepts/étapes lors de la définition d'une fonctions (et d'un programme en général)

Spécification:

Une description de ce qui est attendu

- à un certain niveau d'abstraction
- doit être suffisamment précis
- proche d'une description "mathématique"
- peut utiliser des exemples *pertinents*

Un contrat

Entrées $\longrightarrow$ Fonction $\longrightarrow$ Sorties

Consiste en:

- 1 description
- 1 signature (son **type**)
- des exemples

Fonctions: SPECIFICATION et IMPLEMENTATION

Il est très important de distinguer 2 concepts/étapes lors de la définition d'une fonctions (et d'un programme en général)

Spécification:

Une description de ce qui est attendu

- à un certain niveau d'abstraction
- doit être suffisamment précis
- proche d'une description "mathématique"
- peut utiliser des exemples *pertinents*

Un contrat

Entrées $\longrightarrow$ Fonction $\longrightarrow$ Sorties

Consiste en:

- 1 description
- 1 signature (son **type**)
- des exemples

Implémentation:

La description de comment le réaliser

- le code OCaml

Entrées $\longrightarrow$ Fonction $\longrightarrow$ Sorties

Fonctions: SPECIFICATION et IMPLEMENTATION

Il est très important de distinguer 2 concepts/étapes lors de la définition d'une fonctions (et d'un programme en général)

Spécification:

Une description de ce qui est attendu

- à un certain niveau d'abstraction
- doit être suffisamment précis
- proche d'une description "mathématique"
- peut utiliser des exemples *pertinents*

Un contrat

Entrées $\longrightarrow$ $\boxed{\text{Fonction}}$ $\longrightarrow$ Sorties

Consiste en:

- 1 description
- 1 signature (son **type**)
- des exemples

Implémentation:

La description de comment le réaliser

- le code OCaml

Entrées $\longrightarrow$ $\boxed{\text{Fonction}}$ $\longrightarrow$ Sorties

Définir une fonction : Spécification PUIS Implémentation

Plan

Pourquoi la récursivité ?

Exemples de pbs que l'on ne sait pas encore résoudre en OCaml

- calculer la moyenne de n entiers lus dans un fichier
- calculer les nombres premiers compris entre 1 et n
- définir une image composée de n figures géomètriques
- définir un type polynôme

$\rightarrow$ **ce qu'il nous manque ?**

Pourquoi la récursivité ?

Exemples de pbs que l'on ne sait pas encore résoudre en OCaml

- calculer la moyenne de n entiers lus dans un fichier
- calculer les nombres premiers compris entre 1 et n
- définir une image composée de n figures géomètriques
- définir un type polynôme

$\rightarrow$ **ce qu'il nous manque ?**

1. décrire des exécutions comportant un nombre variable de calculs
2. définir des objets de taille importante/variable

Pourquoi la récursivité ?

Exemples de pbs que l'on ne sait pas encore résoudre en OCaml

- calculer la moyenne de n entiers lus dans un fichier
- calculer les nombres premiers compris entre 1 et n
- définir une image composée de n figures géomètriques
- définir un type polynôme

$\rightarrow$ **ce qu'il nous manque ?**

1. décrire des exécutions comportant un nombre variable de calculs
2. définir des objets de taille importante/variable

En programmation impérative (langages Python [INF101], C [INF203]) :

instruction itérative (`while`, `for`) et tableaux

En programmation fonctionnelle :

la récursivité . . .

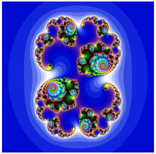
A propos de récursivité

Qu’est-ce que la récursivité, qu’est-ce qu’une définition récursive ?

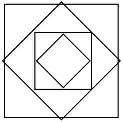
A propos de récursivité

Qu'est-ce que la récursivité, qu'est-ce qu'une définition récursive ?

Exemple : Quelques objets récursifs



$$u_{n+1} = F(u_n)$$

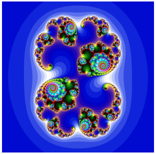


Images sous licence Creative Common

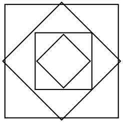
A propos de récursivité

Qu'est-ce que la récursivité, qu'est-ce qu'une définition récursive ?

Exemple : Quelques objets récursifs



$$u_{n+1} = F(u_n)$$



Fonctions récursives

- la valeur du résultat est obtenue en exécutant plusieurs fois une même fonction sur des données différentes :

$$f(f(f(...f(x_0)...)))$$

- généralisation des suites récurrentes
- un élément de base de la programmation fonctionnelle . . .
- et beaucoup (beaucoup) d'autres applications en informatique !

Exemple : Définition récursive de la factorielle

$$\begin{cases} U_0 = 1 \\ U_n = n \times U_{n-1}, n \geq 1 \end{cases} \qquad \begin{cases} 0! = 1 \\ n! = n \times (n-1)!, n \geq 1 \end{cases}$$

- Cette définition fournit un résultat pour tout entier ≥ 0 :
  $\rightarrow$ elle est bien fondée . . .
  contre-exemple : $U_0 = 1$ et $U_n = n \times U_{n+1}$, $n \geq 1$
- On peut montrer sa correction par récurrence sur $\mathbb{N}$:

$$\forall n.\ U_n = n!$$

Fonctions récursives en OCaml

Un 1er exemple

Exemple : Définition récursive de la factorielle

$$\begin{cases} U_0 = 1 \\ U_n = n \times U_{n-1}, n \geq 1 \end{cases} \qquad \begin{cases} 0! = 1 \\ n! = n \times (n-1)!, n \geq 1 \end{cases}$$

- Cette définition fournit un résultat pour tout entier ≥ 0 :
  $\rightarrow$ elle est bien fondée ...
  contre-exemple : $U_0 = 1$ et $U_n = n \times U_{n+1}$, $n \geq 1$
- On peut montrer sa correction par récurrence sur $\mathbb{N}$:

$$\forall n.\ U_n = n!$$

Exemple : Fonction factorielle en OCaml

```
let rec fact (n:int):int =
  if n=0 then 1
  else n * fact(n−1)
```

```
let rec fact (n:int):int =
  match n with
    0 → 1
    | n → n * fact(n−1)
```

Définir une fonction récursive

Spécification: description, signature, exemples, et équations de récurrence

Implémentation: code de la fonction en OCaml

```
let rec fct_name (p1:t1) (p2:t2) ... (pn:tn):t = expr
```

où `expr` peut contenir zéro, un ou plusieurs appels à `fct_name`.
On distinguera différentes sous-expressions de `expr` :

- les cas de base : aucun appel à `fct_name`
- les cas récursifs : un ou plusieurs appel(s) à `fct_name`

Définir une fonction récursive

Spécification: description, signature, exemples, et équations de récurrence

Implémentation: code de la fonction en OCaml

```
let rec fct_name (p1:t1) (p2:t2) ... (pn:tn):t = expr
```

où `expr` peut contenir zéro, un ou plusieurs appels à `fct_name`.
On distinguera différentes sous-expressions de `expr` :

- les cas de base : aucun appel à `fct_name`
- les cas récursifs : un ou plusieurs appel(s) à `fct_name`

Les règles de typage sont les mêmes que pour le fonctions non récursives.

Remarque

- `t1`, .., `tn` peuvent être des types quelconques
- Une fonction récursive ne peut pas être anonyme

Exemples de définitions de fonctions récursives

Exemple : Somme des entiers de 0 **à** n

description + profil + exemples

$$\begin{cases} \sum_{i=0}^{0} i & = 0 \\ \sum_{i=0}^{n} i & = n + \sum_{i=0}^{n-1} i \quad \text{si } n > 0 \end{cases}$$

Exemples de définitions de fonctions récursives

Exemple : Somme des entiers de 0 à n

description + profil + exemples

$$\begin{cases} \sum_{i=0}^{0} i & = 0 \\ \sum_{i=0}^{n} i & = n + \sum_{i=0}^{n-1} i \quad \text{si } n > 0 \end{cases}$$

```
let rec sum (n : int) : int =
 match n with
  | 0 -> 0
  | n -> n + sum (n - 1)
```

Exemples de définitions de fonctions récursives

Exemple : Somme des entiers de 0 à n

description + profil + exemples

$$\begin{cases} \sum_{i=0}^{0} i & = 0 \\ \sum_{i=0}^{n} i & = n + \sum_{i=0}^{n-1} i \quad \text{si } n > 0 \end{cases}$$

```
let rec sum (n : int) : int =
  match n with
   | 0 → 0
   | n → n + sum (n − 1)
```

Exemple : Division entière

description + profil + exemples

$$a/b = \begin{cases} 0 & \text{si } a < b \\ 1 + (a - b)/b & \text{si } b \leq a \end{cases}$$

Exemples de définitions de fonctions récursives

Exemple : Somme des entiers de 0 à n

description + profil + exemples

$$\begin{cases} \sum_{i=0}^{0} i & = 0 \\ \sum_{i=0}^{n} i & = n + \sum_{i=0}^{n-1} i \quad \text{si } n > 0 \end{cases}$$

```
let rec sum (n : int) : int =
 match n with
  | 0 → 0
  | n → n + sum (n − 1)
```

Exemple : Division entière

description + profil + exemples

$$a/b = \begin{cases} 0 & \text{si } a < b \\ 1 + (a - b)/b & \text{si } b \leq a \end{cases}$$

```
let rec div (a : int) (b: int) : int =
 if a < b then 0
 else 1 + div (a − b) (b)
```

Essayons . . .

Exercice : reste de la division entière

Définir une fonction qui calcule le reste de la division entière

Exercice : suite de Fibonacci

Implémenter une fonction qui renvoie le n^{ieme} terme de Fibonacci où n est donné comme paramètre. La suite de Fibonnaci est défini comme :

$$fib_n = \begin{cases} 1 & \text{si } n = 0 \text{ ou } n = 1 \\ fib_{n-1} + fib_{n-2} & \text{si } n > 1 \end{cases}$$

Exercice

Exercice: la fonction puissance (2 versions)

$$\begin{cases} x^0 & = 1 \\ x^n & = x * x^{n-1} \quad \text{si } 0 < n \end{cases} \qquad \begin{cases} x^0 & = 1 \\ x^n & = (x * x)^{n/2} \quad \text{si } n \text{ est pair} \\ x^n & = x * (x * x)^{\frac{n-1}{2}} \quad \text{si } n \text{ est impair} \end{cases}$$

- Donnez 2 implémentations de la fonction `power`: `int` $\rightarrow$ `int` $\rightarrow$ `int` en vous basant sur ces 2 définitions équivalentes.
- Quelle est la différence entre ces deux versions ?

Exercice

Exercice: la fonction puissance (2 versions)

$$\begin{cases} x^0 & = 1 \\ x^n & = x * x^{n-1} \quad \text{si } 0 < n \end{cases} \qquad \begin{cases} x^0 & = 1 \\ x^n & = (x * x)^{n/2} \quad \text{si } n \text{ est pair} \\ x^n & = x * (x * x)^{\frac{n-1}{2}} \quad \text{si } n \text{ est impair} \end{cases}$$

- Donnez 2 implémentations de la fonction `power`: `int` $\rightarrow$ `int` $\rightarrow$ `int` en vous basant sur ces 2 définitions équivalentes.
- Quelle est la différence entre ces deux versions ?

Plan

Terminaison

Pensez vous que l'exécution de cette fonction termine ?

$$mac(n) = \begin{cases} n - 10 & \text{si } n > 100 \\ mac(mac(n + 11)) & \text{si } n \leq 100 \end{cases}$$

Terminaison

Pensez vous que l'exécution de cette fonction termine ?

$$mac(n) = \begin{cases} n - 10 & \text{si } n > 100 \\ mac(mac(n+11)) & \text{si } n \leq 100 \end{cases}$$

Et celles-ci ?

La fonction puissance

$$\begin{cases} x^0 & = 1 \\ x^n & = x * x^{n-1} \quad \text{si } 0 < n \end{cases}$$

La fonction factorielle

$$\begin{cases} fact(0) & 1 \\ fact(1) & = 1 \\ fact(n) & = \frac{fact(n+1)}{n+1} \end{cases}$$

Terminaison

Pensez vous que l'exécution de cette fonction termine ?

$$mac(n) = \begin{cases} n - 10 & \text{si } n > 100 \\ mac(mac(n+11)) & \text{si } n \leq 100 \end{cases}$$

Et celles-ci ?

La fonction puissance

$$\begin{cases} x^0 & = 1 \\ x^n & = x * x^{n-1} \quad \text{si } 0 < n \end{cases}$$

La fonction factorielle

$$\begin{cases} fact(0) & 1 \\ fact(1) & = 1 \\ fact(n) & = \frac{fact(n+1)}{n+1} \end{cases}$$

Il est **fondamental** de savoir décider si une fonction termine ou non

Peut-on caractériser la terminaison sur l'arbre des appels ?

Comment prouver qu'une fonction récursive termine ?

En utilisant une mesure . . .

Theorem

Toute suite positive <u>strictement</u> décroissante converge

Comment prouver qu'une fonction récursive termine ?

En utilisant une mesure . . .

Theorem

Toute suite positive <u>strictement</u> décroissante converge

Méthode générale pour prouver qu'une fonction f termine ?

Dériver une mesure $\mathcal{M}(f\ n)$ t.q. :

- $\mathcal{M}(f\ n)$ est positive
- $\mathcal{M}(f\ n)$ dépend des paramètres de la fonction
- $\mathcal{M}(f\ n)$ décroit strictement entre deux appels récursifs
- la suite $\mathcal{M}(f\ n)$ converge vers une valeur m_0 associée à un cas de base

Comment prouver qu'une fonction récursive termine ?

En utilisant une mesure . . .

Theorem

Toute suite positive strictement décroissante converge

Méthode générale pour prouver qu'une fonction f termine ?

Dériver une mesure $\mathcal{M}(f\ n)$ t.q. :

- $\mathcal{M}(f\ n)$ est positive
- $\mathcal{M}(f\ n)$ dépend des paramètres de la fonction
- $\mathcal{M}(f\ n)$ décroit strictement entre deux appels récursifs
- la suite $\mathcal{M}(f\ n)$ converge vers une valeur m_0 associée à un cas de base

Exemple : Terminaison de la fonction `sum`

```
let rec sum (x : int) : int =
  match x with
   | 0 → 0
   | x → x + sum (x − 1)
```

Comment prouver qu'une fonction récursive termine ?

En utilisant une mesure ...

Theorem

Toute suite positive strictement décroissante converge

Méthode générale pour prouver qu'une fonction f termine ?

Dériver une mesure $\mathcal{M}(f\ n)$ t.q. :

- $\mathcal{M}(f\ n)$ est positive
- $\mathcal{M}(f\ n)$ dépend des paramètres de la fonction
- $\mathcal{M}(f\ n)$ décroit strictement entre deux appels récursifs
- la suite $\mathcal{M}(f\ n)$ converge vers une valeur m_0 associée à un cas de base

Exemple : Terminaison de la fonction `sum`

```
let rec sum (x : int) : int =
  match x with
   | 0 → 0
   | x → x + sum (x − 1)
```

Mesure :

- $\mathcal{M}(sum\ n) = n$, $\mathcal{M}(sum\ n) \in \mathbb{N}$
- $\mathcal{M}(sum\ n) > \mathcal{M}(sum\ (n-1))$ puisque $n > n-1$
- $\mathcal{M}(sum\ n)$ converge vers 0.

Terminaison de quelques fonctions

Exercice: trouver les *mesures*

Prouvez que les fonctions `factorielle`, `puissance`, `quotient`, `reste` terminent . . .

Terminaison de quelques fonctions

factorielle et puissance

Terminaison de `fact`:

```
let rec fact (n:int):int =
  match n with
    0 -> 1
    | n -> n * fact(n-1)
```

Terminaison de `fact`:

```
let rec fact (n:int):int =
  match n with
    0 → 1
    | n → n ∗ fact(n−1)
```

- Définissons $\mathcal{M}(\textit{fact}\ n) = n \in \mathbb{N}$
- $\mathcal{M}(\textit{fact}\ n) > \mathcal{M}(\textit{fact}\ (n-1))$ puisque $n > n-1$
- $\mathcal{M}(\textit{fact}\ n)$ converge vers 0.

Terminaison de quelques fonctions

factorielle et puissance

Terminaison de `fact`:

```
let rec fact (n:int):int =
  match n with
   0 → 1
   | n → n * fact(n−1)
```

- Définissons $\mathcal{M}(\textit{fact}\ n) = n \in \mathbb{N}$
- $\mathcal{M}(\textit{fact}\ n) > \mathcal{M}(\textit{fact}\ (n-1))$ puisque $n > n - 1$
- $\mathcal{M}(\textit{fact}\ n)$ converge vers 0.

Terminaison de `power`:

```
let rec power (a:float) (n:int):float =
  if (n=0) then 1.
 else (if n>0 then a *. power  a (n−1)
 else 1. /. (power a (−n))
  )
```

Terminaison de quelques fonctions

factorielle et puissance

Terminaison de fact:

```
let rec fact (n:int):int =
  match n with
    0 -> 1
    | n -> n * fact(n-1)
```

- Définissons $\mathcal{M}(\textit{fact}\ n) = n \in \mathbb{N}$
- $\mathcal{M}(\textit{fact}\ n) > \mathcal{M}(\textit{fact}\ (n-1))$ puisque $n > n - 1$
- $\mathcal{M}(\textit{fact}\ n)$ converge vers 0.

Terminaison de power:

```
let rec power (a:float) (n:int):float =
  if (n=0) then 1.
 else (if n>0 then a *. power  a (n-1)
 else 1. /. (power a (-n))
  )
```

- Définissons $\mathcal{M}(\textit{power}\ a\ n) = n$
- pour tout appel récursif $\mathcal{M}(\textit{power}\ a\ n) \in \mathbb{N}$
- $\mathcal{M}(\textit{power}\ a\ n) > \mathcal{M}(\textit{power}\ a\ (n-1))$
- $\mathcal{M}(\textit{power}\ a\ n)$ converge vers 0.

Terminaison de quelques fonctions

```
let rec quotient (a:int) (b:int):int =
 if (a<b) then 0
 else 1 + quotient (a-b) b
```

```
let rec reste (a:int) (b:int):int =
  if (a<b) then a
  else reste (a-b) b
```

Terminaison de quelques fonctions

```
let rec quotient (a:int) (b:int):int =
 if (a<b) then 0
 else 1 + quotient (a-b) b
```

```
let rec reste (a:int) (b:int):int =
  if (a<b) then a
  else reste (a-b) b
```

Terminaison de `quotient` et `reste`:

- Définissons $\mathcal{M}(X\,a\,b) = a$
- $\mathcal{M}(X\,a\,b) \in \mathbb{N}$ (d'après la spec)
- $\mathcal{M}(X\,a\,b) > \mathcal{M}(X\,(a-b)\,b)$ puisque $b > 0$
- $\mathcal{M}(X\,a\,b)$ converge vers une valeur a_0 telle que $a_0 < b$.

où $X \in \{\text{quotient}, \text{reste}\}$

Plan

Types récursifs : pour faire quoi ?

fonction récursive

- définie *en "fonction d'elle-même"* (cas de base, cas récursifs)
- permet de décrire des **suites de calcul** de longueur arbitraire
  ex : (`fact` 5), (`fact` 10), etc.
- problème de **terminaison**

Type récursif

- défini *en "fonction de lui-même" . . .* (cas de base, cas récursifs)
- permet de décrire des **données** de taille arbitraire
- problème de **terminaison** : type "bien fondés"

Exemples d'application :

définir des ensembles, des séquences, des arborescences . . .

Types récursifs : définition et exemple

Exemple :

```
type t =
     C of char (* constructeur non recursif *)
    |S of int * t   (* constructeur recursif *)
```

Exemple de valeurs dy type `t` ?

Types récursifs : définition et exemple

Exemple :

```
type t =
    C of char (* constructeur non recursif *)
   |S of int * t   (* constructeur recursif *)
```

Exemple de valeurs dy type `t` ?

`C('x')`

Types récursifs : définition et exemple

Exemple :

```
type t =
     C of char (* constructeur non recursif *)
    |S of int * t   (* constructeur recursif *)
```

Exemple de valeurs dy type `t` ?

`C('x')`

Types récursifs : définition et exemple

Exemple :

```
type t =
    C of char (* constructeur non recursif *)
   |S of int * t   (* constructeur recursif *)
```

Exemple de valeurs dy type `t` ?

`C('x')`   `S(5, C('x'))`

Types récursifs : définition et exemple

Exemple :

```
type t =
     C of char (* constructeur non recursif *)
    |S of int * t   (* constructeur recursif *)
```

Exemple de valeurs dy type `t` ?

`C('x')`   `S(5, C('x'))`   `S(12, S(5, C('x')))`   etc.

Types récursifs : définition et exemple

Exemple :

```
type t =
    C of char (* constructeur non recursif *)
   |S of int * t   (* constructeur recursif *)
```

Exemple de valeurs dy type `t` ?

`C('x')`   `S(5, C('x'))`   `S(12, S(5, C('x')))`   etc.

→ séquence d'entiers terminée par un caractère ...

Définition générale

`type nouveau_type = ... nouveau_type ...`

Pour être “bien fondé”, `nouveau_type` doit être :

- un type **somme**
- avec au moins un constructeur non récursif

DEMO: exemples de définition de types réursifs (bien fondés ou non)

Un type récursif : les entiers de Peano

le point de vue mathématique et le point de vue OCaml

Les entiers de Peano ($\mathrm{NatPeano}$) : une manière de définir $\mathbb{N}$

Définition récursive de $\mathrm{NatPeano}$:

- une base : le constructeur “non récursif” Zero
- un *constructeur* “récursif”:
  Suc: le successeur d’un élément de $\mathrm{NatPeano}$
- Zero est le successeur d’aucun élément de $\mathrm{NatPeano}$
- deux élément de $\mathrm{NatPeano}$ qui ont même successeur sont égaux

$\hookrightarrow$ $\mathbb{N}$ est le plus petit ensemble contenant Zero et le successeur de tout élément de $\mathbb{N}$

Un type récursif : les entiers de Peano

le point de vue mathématique et le point de vue OCaml

Les entiers de Peano $(\mathrm{NatPeano})$: une manière de définir $\mathbb{N}$

Définition récursive de $\mathrm{NatPeano}$:

- une base : le constructeur “non récursif” Zero
- un *constructeur* “récursif”:
  Suc: le successeur d'un élément de $\mathrm{NatPeano}$
- Zero est le successeur d'aucun élément de $\mathrm{NatPeano}$
- deux élément de $\mathrm{NatPeano}$ qui ont même successeur sont égaux

$\hookrightarrow$ $\mathbb{N}$ est le plus petit ensemble contenant Zero et le successeur de tout élément de $\mathbb{N}$

$$\text{Zero} \xrightarrow{\text{Suc}} \text{Suc(Zero)} \xrightarrow{\text{Suc}} \text{Suc(Suc(Zero))} \xrightarrow{\text{Suc}} \cdots$$

Définition de $\mathrm{NatPeano}$ en OCaml:

```
type natPeano = Zero | Suc of natPeano
```

$\hookrightarrow$ `natPeano` est **un type récursif**

Entiers de Peano

Conversion vers/depuis les entiers

Convertir une entier de Peano en entier

- Description: `natPeano2int` traduit un entier de Peano en son équivalent entier
- Profil/Signature: `natPeano2int`: `natPeano` $\rightarrow$ `int`
- Ex.: `natPeano2int Zero` = 0,
  `natPeano2int Suc(Suc(Suc Zero))`=3

Convertir une entier de Peano en entier

- Description: `natPeano2int` traduit un entier de Peano en son équivalent entier
- Profil/Signature: `natPeano2int`: `natPeano` → `int`
- Ex.: `natPeano2int Zero` = 0, `natPeano2int Suc(Suc(Suc Zero))`=3

```
let rec natPeano2int (n:natPeano):int =
 match n with
    Zero → 0
   | Suc (nprime) → 1+ natPeano2int nprime
```

Convertir une entier de Peano en entier

- Description: `natPeano2int` traduit un entier de Peano en son équivalent entier
- Profil/Signature: `natPeano2int`: `natPeano` → `int`
- Ex.: `natPeano2int Zero` = 0, `natPeano2int Suc(Suc(Suc Zero))`=3

```
let rec natPeano2int (n:natPeano):int =
 match n with
   Zero → 0
  | Suc (nprime) → 1+ natPeano2int nprime
```

Convertir un entier en entier de Peano
comme ci-dessus, mais dans le sens inverse . . . !

Convertir une entier de Peano en entier

- Description: `natPeano2int` traduit un entier de Peano en son équivalent entier
- Profil/Signature: `natPeano2int`: `natPeano` → `int`
- Ex.: `natPeano2int Zero` = 0,
  `natPeano2int Suc(Suc(Suc Zero))`=3

```
let rec natPeano2int (n:natPeano):int =
 match n with
    Zero → 0
   | Suc (nprime) → 1+ natPeano2int nprime
```

Convertir un entier en entier de Peano
comme ci-dessus, mais dans le sens inverse . . . !

```
let rec int2natPeano (n:int):natPeano=
  match n with
   0 → Zero
   | nprime → Suc (int2natPeano (n−1))
```

DEMO: Fonctions natPeano2int et int2natPeano

Exercice: somme de deux entiers de Peano

- Définir une fonction qui effectue la *somme de deux entiers de Peano* sans utiliser les fonction de conversion depuis/vers les entiers
- Prouver que votre fonction termine

Exercice: produit de deux entiers de Peano

- Définir une fonction qui *multiplie deux entiers de Peano*
- Prouver que votre fonction termine

Exercice: factorielle d'un entier de Peano

- Définir une fonction qui *calcule la factorielle d'un entier de Peano*
- Prouver que votre fonction termine

Conclusion

La récursivité : une notion fondamentale . . .

On a vu deux formes de récursivité :

- les fonctions récursives
  - équations récursives
  - terminaison
  - définition = spécification (description, profil, équations récursives, exemples)
    + implémentation
    + arguments de terminaison

- les types/valeurs/objets récursifs
  - définition (“bien fondée”)
  - fonctions récursives portant sur des types récursifs :
    $\rightarrow$ construites selon la définition du type récursif