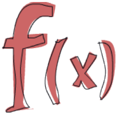
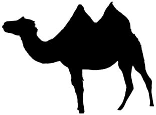
INF201
Algorithmique et Programmation Fonctionnelle
Cours 5 : Fonctions et types récursifs (suite)

Année 2018 - 2019





Plan

Fonctions récursive ?

Fonction dont la valeur du résultat est obtenue en exécutant plusieurs fois cette même fonction sur des données différentes :

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

$\rightarrow$ permet de décrire un nombre variable de pas de calcul . . .

Fonctions récursive ?

Fonction dont la valeur du résultat est obtenue en exécutant plusieurs fois cette même fonction sur des données différentes :

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

$\rightarrow$ permet de décrire un nombre variable de pas de calcul ...

Définition d'une fonction récursive `f`

- spécification =
  - description : ce que fait la fonction
  - profil : les type des paramètres, du résultat
    (`f` : `t1` $\rightarrow$ `t2` $\rightarrow$ ... `tn` $\rightarrow$ `t`)
  - des exemples
  - les équations récursives
  - un argument de terminaison ???
- implémentation (le code en OCaml)
  $\rightarrow$ dérivée des équations récursives ...

Equations récursives

Définition d'une fonction récursive $f : \mathcal{D}_1 \to \mathcal{D}_2$?

$\to$ un ensemble d'équations de 2 formes :

- les **cas de base** : $f(x_0) = e_0$
  - $x_0 \in \mathcal{D}_1$
  - e_0 est une expression (de type $\mathcal{D}_2$) qui **ne dépend pas** de f
- les **cas récursifs** : $f(x) = e$
  - $x \in \mathcal{D}_1$, $x \neq x_0$
  - e est une expression (de type $\mathcal{D}_2$) qui **dépend** de f

Exemple :

Equations récursives de la fonction $f : \mathbb{N} \to \mathbb{N}^+$ t.q. $f(n) = \sum_{i=0}^{n} i^2$.

Equations récursives

Définition d'une fonction récursive $f : \mathcal{D}_1 \to \mathcal{D}_2$?

$\to$ un ensemble d'équations de 2 formes :

- les **cas de base** : $f(x_0) = e_0$
  - $x_0 \in \mathcal{D}_1$
  - e_0 est une expression (de type $\mathcal{D}_2$) qui **ne dépend pas** de f
- les **cas récursifs** : $f(x) = e$
  - $x \in \mathcal{D}_1$, $x \neq x_0$
  - e est une expression (de type $\mathcal{D}_2$) qui **dépend** de f

Exemple :

Equations récursives de la fonction $f : \mathbb{N} \to \mathbb{N}^+$ t.q. $f(n) = \sum_{i=0}^{n} i^2$.

Les équations récursives doivent être **bien fondées** :
$\to$ elle doivent permettre de calculer $f(x)$ pour tout $x \in \mathcal{D}_1$
(le calcul de $f(x)$ doit terminer !)

contre-exemple : $\mathcal{D}_1 = \mathbb{N}$ et $\mathcal{D}_2 = \mathbb{N}$

$$\begin{array}{rcl} f(0) & = & 0 \\ f(x) & = & x^2 + f(x+2) \end{array}$$

Plan

Terminaison

Pensez vous que l'exécution de cette fonction termine ? Fonction 91 de McCarthy (cf. wikipedia)

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

Terminaison

Pensez vous que l'exécution de cette fonction termine ? Fonction 91 de McCarthy (cf. wikipedia)

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

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 ? Fonction 91 de McCarthy (cf. wikipedia)

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

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

Les fonctions suivantes terminent-elles ?

$f_1 : \mathbb{Z} \to \mathbb{N}$

$$\begin{array}{rcl} f_1(3) & = & 4 \\ f_1(n) & = & 1 + f_1(n-1) \text{ si } n \neq 3 \end{array}$$

$f2 : \mathbb{Z} \to \mathbb{N}$

$$\begin{array}{rcl} f_2(0) & = & 0 \\ f_2(n) & = & 1 + f_2(n-1) \text{ si } n > 0 \\ f_2(n) & = & 1 + f_2(n+1) \text{ si } n < 0 \end{array}$$

$f_3 : \mathbb{Z} \to \mathbb{N}$

$$\begin{array}{rcl} f_3(0) & = & 1 \\ f_3(n) & = & 1 + f_3(n-2) \text{ si } n > 0 \\ f_3(n) & = & 1 + f_3(n+2) \text{ si } n < 0 \end{array}$$

$f_4 : \mathbb{N} \to \mathbb{N}$

$$\begin{array}{rcl} f_4(0) & = & 1 \\ f_4(1) & = & 2 \\ f_4(n) & = & 1 + f_4(n-2) \text{ si } n \notin \{0, 1\} \end{array}$$

$f_5 : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$$\begin{array}{rcl} f_5(0, 0) & = & 1 \\ f_5(a, b) & = & f_5(a, b-1) \text{ si } a > b \\ f_5(a, b) & = & f_5(a-1, b) \text{ si } a \leq b \end{array}$$

Comment prouver qu'une fonction termine ?

Toute suite positive strictement décroissante converge . . .

Terminaison de `f` : `t1` $\rightarrow$ `t2` $\rightarrow$ ... $\rightarrow$ `tn` $\rightarrow$ `t` ?

Comment prouver qu'une fonction termine ?

Toute suite positive strictement décroissante converge ...

Terminaison de `f` : `t1` $\rightarrow$ `t2` $\rightarrow$ ... $\rightarrow$ `tn` $\rightarrow$ `t` ?

Trouver une mesure $\mathcal{M}(f(n_1, n_2, \ldots, n_p))$ t.q. :

- $\mathcal{M}(f(n_1, n_2, \ldots, n_p))$ est positive

Comment prouver qu'une fonction termine ?

Toute suite positive strictement décroissante converge . . .

Terminaison de `f : t1` $\rightarrow$ `t2` $\rightarrow$ ... $\rightarrow$ `tn` $\rightarrow$ `t` ?

Trouver une mesure $\mathcal{M}(f(n_1, n_2, \ldots, n_p))$ t.q. :

- $\mathcal{M}(f(n_1, n_2, \ldots, n_p))$ est positive
- $\mathcal{M}(f(n_1, n_2, \ldots, n_p))$ dépend des paramètres n_1, n_2, $\ldots$ n_p de f

Comment prouver qu'une fonction termine ?

Toute suite positive strictement décroissante converge . . .

Terminaison de `f` : `t1` $\rightarrow$ `t2` $\rightarrow$ ... $\rightarrow$ `tn` $\rightarrow$ `t` ?

Trouver une mesure $\mathcal{M}(f(n_1, n_2, \ldots, n_p))$ t.q. :

- $\mathcal{M}(f(n_1, n_2, \ldots, n_p))$ est positive
- $\mathcal{M}(f(n_1, n_2, \ldots, n_p))$ dépend des paramètres n_1, n_2, $\ldots n_p$ de f
- $\mathcal{M}(f(n_1, n_2, \ldots, n_p))$ décroit strictement entre deux appels récursifs : $\forall$ équation de la forme

$$f(n_1, n_2, \ldots n_p) = \ldots f(n'_1, n'_2, \ldots n'_p)) \ldots \text{ (appel récursif)}$$

alors

$$\mathcal{M}(f(n_1, n_2, \ldots, n_p)) > \mathcal{M}(f(n'_1, n'_2, \ldots, n'_p))$$

Comment prouver qu'une fonction termine ?

Toute suite positive strictement décroissante converge ...

Terminaison de `f` : `t1` $\rightarrow$ `t2` $\rightarrow$ ... $\rightarrow$ `tn` $\rightarrow$ `t` ?

Trouver une mesure $\mathcal{M}(f(n_1, n_2, \ldots, n_p))$ t.q. :

- $\mathcal{M}(f(n_1, n_2, \ldots, n_p))$ est positive
- $\mathcal{M}(f(n_1, n_2, \ldots, n_p))$ dépend des paramètres n_1, n_2, $\ldots n_p$ de f
- $\mathcal{M}(f(n_1, n_2, \ldots, n_p))$ décroit strictement entre deux appels récursifs : $\forall$ équation de la forme

  $$f(n_1, n_2, \ldots n_p) = \ldots f(n'_1, n'_2, \ldots n'_p)) \ldots \text{ (appel récursif)}$$

  alors

  $$\mathcal{M}(f(n_1, n_2, \ldots, n_p)) > \mathcal{M}(f(n'_1, n'_2, \ldots, n'_p))$$

- $\forall$ exécution de f, la suite $\mathcal{M}(f(n_1, n_2, \ldots n_p)$ converge vers une valeur m_0 associée à un cas de base :
  - $\exists$ une équation de la forme : $f(n^0_1, n^0_2, \ldots n^0_p) = e_0$ (sans appel récursif)
  - $\mathcal{M}(f(n^0_1, n^0_2, \ldots, n^0_p)) = m_0$

Retour sur les exemples ...

$f_1 : \mathbb{Z} \to \mathbb{N}$

$$\begin{aligned} f_1(3) &= 4 \\ f_1(n) &= 1 + f_1(n-1) \text{ si } n \neq 3 \end{aligned}$$

$f2 : \mathbb{Z} \to \mathbb{N}$

$$\begin{aligned} f_2(0) &= 0 \\ f_2(n) &= 1 + f_2(n-1) \text{ si } n > 0 \\ f_2(n) &= 1 + f_2(n+1) \text{ si } n < 0 \end{aligned}$$

$f_3 : \mathbb{Z} \to \mathbb{N}$

$$\begin{aligned} f_3(0) &= 1 \\ f_3(n) &= 1 + f_3(n-2) \text{ si } n > 0 \\ f_3(n) &= 1 + f_3(n+2) \text{ si } n < 0 \end{aligned}$$

$f_4 : \mathbb{N} \to \mathbb{N}$

$$\begin{aligned} f_4(0) &= 1 \\ f_4(1) &= 2 \\ f_4(n) &= 1 + f_4(n-2) \text{ si } n \notin \{0, 1\} \end{aligned}$$

$f_5 : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$$\begin{aligned} f_5(0,0) &= 1 \\ f_5(a,b) &= f_5(a, b-1) \text{ si } a > b \\ f_5(a,b) &= f_5(a-1, b) \text{ si } a \leq b \end{aligned}$$

Exercice : la fonction puissance (2 versions)

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

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

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

Plan

Fonctions *mutuellement* récursives

Sur un exemple

Récursivité "directe" : appels récursifs à une seule fonction
Qu'en est-il d'une fonction `f` qui appelle `g` qui appelle `f` qui appelle `g` . . .
$\hookrightarrow$ fonctions mutuellement récursives (récursivité croisée)

Fonctions *mutuellement* récursives

Sur un exemple

Récursivité “directe” : appels récursifs à une seule fonction
Qu’en est-il d’une fonction `f` qui appelle `g` qui appelle `f` qui appelle `g` . . .
$\hookrightarrow$ fonctions mutuellement récursives (récursivité croisée)

Exemple :
Comment déterminer si un entier est pair ou impair sans utiliser /, $*$, `mod`
(donc en utilisant uniquement $-$ et =) ?

Fonctions *mutuellement* récursives

Sur un exemple

Récursivité “directe” : appels récursifs à une seule fonction
Qu’en est-il d’une fonction `f` qui appelle `g` qui appelle `f` qui appelle `g` ...
$\hookrightarrow$ fonctions mutuellement récursives (récursivité croisée)

Exemple :
Comment déterminer si un entier est pair ou impair sans utiliser /, $*$, `mod`
(donc en utilisant uniquement $-$ et =) ?

- $n \in \mathbb{N}$ est impair si $n - 1$ est pair
- $n \in \mathbb{N}$ est pair si $n - 1$ est impair
- 0 est pair
- 0 n’est pas impair

Fonctions *mutuellement* récursives

Sur un exemple

Récursivité “directe” : appels récursifs à une seule fonction
Qu’en est-il d’une fonction `f` qui appelle `g` qui appelle `f` qui appelle `g` ...
$\hookrightarrow$ fonctions mutuellement récursives (récursivité croisée)

Exemple :
Comment déterminer si un entier est pair ou impair sans utiliser /, $*$, `mod`
(donc en utilisant uniquement $-$ et =) ?

- $n \in \mathbb{N}$ est impair si $n - 1$ est pair
- $n \in \mathbb{N}$ est pair si $n - 1$ est impair
- 0 est pair
- 0 n’est pas impair

```
let rec
 pair (n:int):bool = if n=0 then true else impair (n-1)
 and
 impair (m:int):bool = if m=0 then false else pair (m-1)
```

DEMO: pair et impair, récursivité croisée

Fonctions mutuellement récursives

Généralisation

```
let rec fct1 [parametres+type resultat] = expr_1
  and fct2 [parametres+ type resultat] = expr_2
     ....
  and fctn [parametres+type resultat] = expr_n
```

où

`expr_1`, `expr_2`, ..., `expr_n` peuvent appeler `fct1`, `fct2`, ..., `fctn`

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 de 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 de 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 de 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 de 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 de 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 de type `t` ?

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

$\rightarrow$ 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**

Résumons-nous sur les types récursifs ...

Un type récursif est défini en “fonction de lui-même” ...

Résumons-nous sur les types récursifs ...

Un type récursif est défini en “fonction de lui-même” ...

Exemples :

- un ensemble est un soit ensemble vide, soit un ensemble auquel on ajoute un élément
- un polynôme est soit un monôme, soit l’addition d’un monôme et d’un polynôme
- etc.

Résumons-nous sur les types récursifs . . .

Un type récursif est défini en “fonction de lui-même” . . .

Exemples :

- un ensemble est un soit ensemble vide, soit un ensemble auquel on ajoute un élément
- un polynôme est soit un monôme, soit l'addition d'un monôme et d'un polynôme
- etc.

En pratique, les types récursifs doivent être “bien fondés”

$\rightarrow$ ils sont définis par un type somme avec au moins un constructeur non récursif (constant ou non)

```
type typeR =
    C1 of t1 |C2 of t2 | ... |Cn of tn
   |R1 of ... typeR ... |R2 of ... typeR ... |... |Rp of ... typeR ...
```

C_i : constructeurs **non récursifs**, R_i : constructeurs **récursifs**

Autres exemples de définition de type récursif

- Un ensemble d'entier est :
  - soit l'ensemble vide : $\emptyset$
  - soit l'**insertion** d'un entier x à un ensemble E : $Ins(x, E)$

$$ens = \emptyset \cup \{Ins(x, E) \mid x \in \mathbb{N}, E \in ens\}$$

En CAML :

Autres exemples de définition de type récursif

- Un ensemble d'entier est :
  - soit l'ensemble vide : $\emptyset$
  - soit l'**insertion** d'un entier x à un ensemble E : $Ins(x, E)$

$$ens = \emptyset \cup \{Ins(x, E) \mid x \in \mathbb{N}, E \in ens\}$$

En CAML : `type ens = Vide | Ins of int * ens`

Autres exemples de définition de type récursif

- Un ensemble d'entier est :
  - soit l'ensemble vide : $\emptyset$
  - soit l'**insertion** d'un entier x à un ensemble E : $Ins(x, E)$

$$ens = \emptyset \cup \{Ins(x, E) \mid x \in \mathbb{N}, E \in ens\}$$

En CAML : `type ens = Vide | Ins of int * ens`

- Un polynôme à 1 variable $\alpha_n X^n + \alpha_{n-1} X^{n-1} + \ldots + \alpha_1 X^1 + \alpha_0$ est
  - soit un monôme de coefficient c et de degré d : $Mn(c, d)$
  - soit la **somme** d'un monôme m et d'un polynôme P : $Plus(m, P)$

$$\begin{array}{rcl} monome & = & \{Mn(c, d) \mid (c, d) \in \mathbb{N}^2\} \\ polynome & = & monme \cup \{Plus(m, P) \mid m \in \mathbb{N}, P \in monome\} \end{array}$$

En CAML :

Autres exemples de définition de type récursif

- Un ensemble d'entier est :
  - soit l'ensemble vide : $\emptyset$
  - soit l'**insertion** d'un entier x à un ensemble E : $Ins(x, E)$

$$ens = \emptyset \cup \{Ins(x, E) \mid x \in \mathbb{N}, E \in ens\}$$

En CAML : `type ens = Vide | Ins of int * ens`

- Un polynôme à 1 variable $\alpha_n X^n + \alpha_{n-1} X^{n-1} + \ldots + \alpha_1 X^1 + \alpha_0$ est
  - soit un monôme de coefficient c et de degré d : $Mn(c, d)$
  - soit la **somme** d'un monôme m et d'un polynôme P : $Plus(m, P)$

$$\begin{aligned} monome &= \{Mn(c, d) \mid (c, d) \in \mathbb{N}^2\} \\ polynome &= monme \cup \{Plus(m, P) \mid m \in \mathbb{N}, P \in monome\} \end{aligned}$$

En CAML :

```
 type coef = int  (* non nul *)
type degre = int
type monome = coef * degre
type polynome = Mn of monome |Plus of monome * polynome
```

Fonction (récursive) définie sur un type récursif

Ecrire une fonction $f : tR \to t$ où tR est un type récursif ?

```
type tR =
    C1 of t1 |C2 of t2 | ... |Cn of tn
   |R1 of ... typeR ... |R2 of ... typeR ... |... |Rp of ... typeR ...
```

Fonction (récursive) définie sur un type récursif

Ecrire une fonction $f : tR \to t$ où tR est un type récursif ?

```
type tR =
    C1 of t1 |C2 of t2 | ... |Cn of tn
   |R1 of ... typeR ... |R2 of ... typeR ... |... |Rp of ... typeR ...
```

- *f* doit (en principe) être définie **pour toute valeur** de *tR*
- les valeurs de *tR* sont obtenues en combinant :
  - un nombre arbitraire de constructeurs **récursifs** de *tR* ;
  - un (et un seul !) constructeur **non récursif** de *tR*

$x \in$ `tR` de la forme :

`C1(x1)`, `C2(x1)`, `R1(... C1(x1) ...)`, `R2 (... R1(C2(x2)) ...)`, etc.

Fonction (récursive) définie sur un type récursif

Ecrire une fonction $f : tR \to t$ où tR est un type récursif ?

```
type tR =
    C1 of t1 |C2 of t2 | ... |Cn of tn
   |R1 of ... typeR ... |R2 of ... typeR ... |... |Rp of ... typeR ...
```

- *f* doit (en principe) être définie **pour toute valeur** de *tR*
- les valeurs de *tR* sont obtenues en combinant :
  - un nombre arbitraire de constructeurs **récursifs** de *tR* ;
  - un (et un seul !) constructeur **non récursif** de *tR*

$x \in$ `tR` de la forme :

`C1(x1)`, `C2(x1)`, `R1(... C1(x1) ...)`, `R2 (... R1(C2(x2)) ...)`, etc.

Définition de *f* par induction sur la structure de *tR*

$\to$ au moins une équation par constructeur :

$$\begin{array}{rcl} \text{f (C1(x1))} & = & \cdots \text{ (cas de base)} \\ \text{f (C2(x2))} & = & \cdots \\ \ldots & = & \ldots \\ \text{f (R1(...,x, ...))} & = & \cdots \text{f(x)} \ldots \text{ (cas récursifs)} \\ \text{f (R2(...,x, ...))} & = & \cdots \text{f(x)} \ldots \\ \ldots & = & \ldots \end{array}$$

Exemples de fonctions sur le type “ensemble”

Définition d’un ensemble d’entiers :

```
type ens = Vide | Ins of int * ens
  (* chaque entier est present au plus une fois *)
```

Nombre d’éléments d’un ensemble

nbElem : *ens* $\rightarrow \mathbb{N}$, nbElem(e) est le cardinal de l’ensemble `e`

Exemples de fonctions sur le type “ensemble”

Définition d’un ensemble d’entiers :

```
type ens = Vide | Ins of int * ens
  (* chaque entier est present au plus une fois *)
```

Nombre d’éléments d’un ensemble

nbElem : *ens* $\rightarrow \mathbb{N}$, nbElem(e) est le cardinal de l’ensemble $\mathtt{e}$

$$\text{nbElem (Vide)} \quad =$$

Exemples de fonctions sur le type “ensemble”

Définition d’un ensemble d’entiers :

```
type ens = Vide | Ins of int * ens
  (* chaque entier est present au plus une fois *)
```

Nombre d’éléments d’un ensemble

nbElem : *ens* $\rightarrow \mathbb{N}$, nbElem(e) est le cardinal de l’ensemble $\texttt{e}$

$$\begin{aligned} \text{nbElem (Vide)} &= 0 \\ \text{nbElem (Ins(x,e))} &= \end{aligned}$$

Exemples de fonctions sur le type “ensemble”

Définition d’un ensemble d’entiers :

```
type ens = Vide | Ins of int * ens
  (* chaque entier est present au plus une fois *)
```

Nombre d’éléments d’un ensemble

nbElem : *ens* $\rightarrow \mathbb{N}$, nbElem(e) est le cardinal de l’ensemble e

$$\begin{array}{rcl} \text{nbElem (Vide)} & = & 0 \\ \text{nbElem (Ins(x,e))} & = & 1 + \text{nbElem(e)} \end{array}$$

Appartenance d’un éléments à un ensemble

app : ens $\times \mathbb{N} \rightarrow \mathbb{B}$, app(e,x) vaut vrai ssi $x \in e$

Exemples de fonctions sur le type “ensemble”

Définition d’un ensemble d’entiers :

```
type ens = Vide | Ins of int * ens
  (* chaque entier est present au plus une fois *)
```

Nombre d’éléments d’un ensemble

nbElem : *ens* $\rightarrow \mathbb{N}$, nbElem(e) est le cardinal de l’ensemble `e`

$$\begin{array}{rcl} \text{nbElem (Vide)} & = & 0 \\ \text{nbElem (Ins(x,e))} & = & 1 + \text{nbElem(e)} \end{array}$$

Appartenance d’un éléments à un ensemble

`app` : `ens` $\times \mathbb{N} \rightarrow \mathbb{B}$, `app(e,x)` vaut vrai ssi `x` $\in$ `e`

$$\text{app (Vide, x)} \quad =$$

Exemples de fonctions sur le type “ensemble”

Définition d’un ensemble d’entiers :

```
type ens = Vide | Ins of int * ens
  (* chaque entier est present au plus une fois *)
```

Nombre d’éléments d’un ensemble

nbElem : *ens* $\rightarrow \mathbb{N}$, nbElem(e) est le cardinal de l’ensemble $\mathtt{e}$

$$\begin{aligned} \text{nbElem (Vide)} &= 0 \\ \text{nbElem (Ins(x,e))} &= 1 + \text{nbElem(e)} \end{aligned}$$

Appartenance d’un éléments à un ensemble

$\mathtt{app}$: $\mathtt{ens} \times \mathbb{N} \rightarrow \mathbb{B}$, $\mathtt{app}(\mathtt{e},\mathtt{x})$ vaut vrai ssi $\mathtt{x} \in \mathtt{e}$

$$\begin{aligned} \text{app (Vide, x)} &= \text{false} \\ \text{app (Ins(y,e), x)} &= \end{aligned}$$

Exemples de fonctions sur le type “ensemble”

Définition d’un ensemble d’entiers :

```
type ens = Vide | Ins of int * ens
  (* chaque entier est present au plus une fois *)
```

Nombre d’éléments d’un ensemble

nbElem : *ens* $\rightarrow \mathbb{N}$, nbElem(e) est le cardinal de l’ensemble `e`

$$\begin{aligned} \text{nbElem (Vide)} &= 0 \\ \text{nbElem (Ins(x,e))} &= 1 + \text{nbElem(e)} \end{aligned}$$

Appartenance d’un éléments à un ensemble

`app` : `ens` $\times \mathbb{N} \rightarrow \mathbb{B}$, `app(e,x)` vaut vrai ssi `x` $\in$ `e`

$$\begin{aligned} \text{app (Vide, x)} &= \text{false} \\ \text{app (Ins(y,e), x)} &= \text{x=y or app (e, x)} \end{aligned}$$

DEMO: code CAML de `nbElem` et `app`

Exercice : somme des éléments et maximum d’un ensemble ?

Exercice: conversion entiers de Peano $\leftrightarrow$ `int`

- Définir une fonction qui convertit un entier de Peano en une valeur équivalente de type `int`
- Définir la fonction réciproque
- Prouver que ces fonctions terminent

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

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