

## Programmation fonctionnelle

Xavier Urbain

ensiié

2011/2012

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 1

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 2

## Programmation fonctionnelle

Aspects principaux *ici* :

- Tout est valeur (expressions, fonctions de première classe)
- **Persistence** (JAMAIS de modification)
  - ↪ Simplification du code et donc correction
  - ↪ Efficacité (undo gratuit)

Programmation similaire à raisonnement mathématique (modulo scories)  
↪ **Abstraction**, programmation de haut niveau (*pourquoi ≠ comment*)

**Bonus** modèle formel :  $\lambda$ -calcul  $x \quad (M_1 \ M_2) \quad \lambda x.M \quad \xrightarrow{\beta}$   
capture tout programme  
↪ Preuve de correction aisée et naturelle induction/récurrence

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 3

## Programmation...

Troll : impératif ! fonctionnel !

Futé : **Problème** ↪ bonne structure ↪ langage adapté

Débroussaillage (1 ou 2 cours)

Chaque partie :

- **Structure** de donnée
- **Outils** (itérateurs, constructions, etc.)

Machines : manipulation, mise en œuvre

Travaux dirigés : aspects théoriques, intérêt

## Programmation fonctionnelle

## Troll bashing

Efficace ?

Utilisé ?

Pénible ?

100% pur jus ?

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 4

## Programmation fonctionnelle langages

Haskell, Scheme, Lisp, ML, Erlang, Gallina...

OCAML développé à l'INRIA <http://caml.inria.fr>

- Fonctionnel + traits impératifs ici : noyau fonctionnel et exceptions
- Fortement typé les types sont nos amis !
- Évaluation stricte arguments toujours évalués en premier
- Compilé  $\rightsquigarrow$  vers natif, efficace
- Interprété  $\rightsquigarrow$  vers bytecode, générique
- Boucle d'interaction  $\rightsquigarrow$  pratique

Persistence  $\implies$  gestion mémoire efficace, très complexe, cachée

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 5

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 6

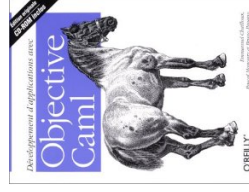
mais on va quand-même apprendre la syntaxe

# CE COURS

# N'EST PAS

# UN COURS D'OCAML

## Biblio



- Développement d'applications avec OCaml
- Chailloux, Manoury, Pagano
- O'Reilly
- <http://www.pps.jussieu.fr/Livres/ora/DA-OCAML/>



- Purely Functional Data Structures
- Okasaki
- Cambridge University Press

**SML !**

## Biblio



- Approche fonctionnelle de la programmation
  - Cousineau, Mauny
  - Éditions/Dunod
- Camlight !**



- Le langage Caml
- Leroy, Weis
- Dunod

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 7

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 8

## Noyau fonctionnel

## exploration

Programme = suite finie

- Expressions → valeurs
- Déclarations → noms

Séparateur (expr) ; ;

Valeurs basiques...

Comparaisons, opérations logiques...

Arithmétique...

## Mémoire...

## intuition

En gros :

```
...
110101010001011110010101011011110011010100001110
00101010110111110001010100010111100011010100101
100001010100010101010101000010111000010101000101
0000000000000000000010110111101000101010000111
...
```

## Mémoire...

## intuition

En gros :

```
...
11010101000101110010101011011110011010100001110
0010101011011110001010100010111100011010100101
10000101010001010101010100001011100010101000101
0000000000000000000010110111101000101010000111
...
```

```
001010101101111001101010000 ?
```

Segmentation ?

```
001010101101111001101010000 ?
```

## Noyau fonctionnel

## exploration

Programme = suite finie

- **Expressions** → valeurs
- **Déclarations** → noms

Séparateur (expr) **;;**

Valeurs basiques...

typées

Comparaisons, opérations logiques...

sur même type

Arithmétique...

sur même type, sans implicite

**Fortement typé** : **jamais** de problèmes de confusion

(ni segfault)

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 13

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 14

## Noyau fonctionnel

## expressions

Branchement

```
if c then value1 else value2
```

Type de **c** **bool** et **même type** pour **value1** et **value2**

Égal à **value1** si **c** vaut **true**

Égal à **value2** si **c** vaut **false**

```
if (2.718 > 3.141) then 666 else 42 ;; (* vaut 42 *)
```

Remarque : **branchement** ≠ fonction logique

```
if condition then true else false ;;
```

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 13

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 14

## Noyau fonctionnel

## expressions

Branchement

```
if c then value1 else value2
```

Type de **c** **bool** et **même type** pour **value1** et **value2**

Égal à **value1** si **c** vaut **true**

Égal à **value2** si **c** vaut **false**

```
if (2.718 > 3.141) then 666 else 42 ;; (* vaut 42 *)
```

Remarque : **branchement** ≠ fonction logique

```
condition ;;
```

Encore des **expressions** !

```
(0x2ff_df_aa = 50_323_370) or (3 < 666) ;;
```

```
3 + (if 42 <= 666 then 1 else -272) ;;
```

- Manipulation d'expressions (tout au même niveau)
- Bon typage nécessaire :
  - Mêmes types des deux côtés dans branchement
  - Mêmes types pour comparaisons, etc.
- Pas de conversion implicite

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 15

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 16

## Noyau fonctionnel

## déclarations

Donner un nom à une expression globalement ou localement

```
let x = expr      (* x : nom de la valeur d'expr *)
```

```
let toto = 40 + 2
```

**Jamais** modifié car **persistance**

Identificateur : **expression**

```
let titi=toto
```

```
let toto = 666
```

```
titi ? ↔ 42
```

```
{{(toto,42),...}}
```

```
{{(titi,42),(toto,42),...}}
```

```
{{(toto,666),(titi,42),(toto,42),...}}
```

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 17

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 18

## Noyau fonctionnel

## expressions avec déclarations

Donner un nom à une expression globalement ou localement

```
let x = e1 in e2
```

```
let titi = 111 * 6 in
```

```
  titi * 666 * titi
```

```
{{(toto,42),...}}
```

```
{{(titi,666),(toto,42),...}}
```

```
{{(titi,666),(toto,42),...}}
```

```
{{(toto,42),...}}
```

Identificateur viable **dans** expression interne

**portée**

```
let x = e1 in e2
```

```
let titi = 111 * 6 in
```

```
  titi * 666 * toto
```

```
{{(toto,42),...}}
```

```
{{(titi,666),(toto,42),...}}
```

```
{{(titi,666),(toto,42),...}}
```

```
{{(toto,42),...}}
```

```
let x = e1 in e2
```

```
let toto = 111 * 6 in
```

```
  toto * 666 * toto
```

```
{{(toto,42),...}}
```

```
{{(toto,666),(toto,42),...}}
```

```
{{(toto,666),(toto,42),...}}
```

```
{{(toto,42),...}}
```

Identificateur viable **dans** expression interne

**portée**

ici : **capture**

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 19

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 20

## Noyau fonctionnel

Enchaîner les déclarations

```

let x = e1 in
  let y = e2 in
    ...

```

Déclarations simultanées

```

let x = e1
and y = e2 in
  ...

```

Donner un nom à une expression ou pas

```

let _ = e1 in e2

```

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 21

## déclarations

## Noyau fonctionnel

Objet comme les autres

```

fun x -> M

```

Un paramètre formel, ici  $x$ , un corps  $M$  (expression)

Type : type paramètre  $\rightarrow$  type de la valeur du corps obtenue

Évaluation de l'application :

1. Remplacement dans le corps du paramètre par une valeur
2. Évaluation du corps

Syntaxe application :

```

(my_function my_param)

```

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 22

## fonctions

```

 $\lambda x.M$ 

```

## Noyau fonctionnel

Objet comme les autres

```

fun x -> M

```

Un paramètre formel, ici  $x$ , un corps  $M$  (expression)

Type : si  $M : \beta$  quand  $x : \alpha$  alors type de fonction :  $\alpha \rightarrow \beta$

Évaluation de l'application :

1. Remplacement dans le corps du paramètre par une valeur
2. Évaluation du corps

Syntaxe application :

```

(my_function my_param)

```

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 23

## fonctions

```

 $\lambda x.M$ 

```

## Noyau fonctionnel

Objet comme les autres

Nommer...

```

let f = fun x -> M

```

Prendre en paramètre...

```

let f = fun x -> (x something)

```

Retourner comme valeur...

```

let f = fun x -> (fun y -> M1)

```

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 24

## Noyau fonctionnel

Prendre en paramètre...

$\text{let } f = \text{fun } x \rightarrow \overbrace{(\dots (x \text{ something}) \dots)}^M$

Type :  $(\alpha \rightarrow \beta) \rightarrow \gamma$  lorsque :

something :  $\alpha, x : \alpha \rightarrow \beta$  et  $M : \gamma$  sachant ça

Retourner comme valeur...

$\text{let } f = \text{fun } x \rightarrow (\text{fun } y \rightarrow M1)$

Type :  $\alpha \rightarrow (\beta \rightarrow \gamma)$  lorsque  $M1 : \gamma$  sachant  $x : \alpha$  et  $y : \beta$

$$(\alpha \rightarrow \beta) \rightarrow \gamma \neq \alpha \rightarrow (\beta \rightarrow \gamma)$$

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 25

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 26

## fonctions

(ordre supérieur)

## Noyau fonctionnel

$\text{let } f = \text{fun } x_1 \rightarrow \dots \rightarrow \text{fun } x_n \rightarrow M$

Donc de type (au pire) :  $\alpha_1 \rightarrow \dots \rightarrow \alpha_n \rightarrow \beta$

**Remarque.** — application partielle

( $f \ e_1 \ \dots \ e_k$ ) avec  $k < n$

(tant que  $\rightarrow$  dans type)

$\rightsquigarrow$  Nouvelles fonctions dépendant de l'environnement à la création

Procédé : clôture

**Remarque.** — Notation équivalente :

$\text{let } f \ x_1 \ \dots \ x_n = M$

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 25

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 26

## Noyau fonctionnel

**Remarque.** — Pas d'évaluation « sous les  $\lambda$  »

$\text{let } f_1 = \text{fun } x \rightarrow \text{fun } y \rightarrow x * x + y * y$

$\text{let } f_2 = \text{fun } x \rightarrow$

$\text{let } x_2 = x * x \text{ in fun } y \rightarrow x_2 + y * y$

**Pas équivalentes** pour application partielle

$f_1 \ 42 \rightsquigarrow \text{fun } y \rightarrow 42 * 42 + y * y$

$f_2 \ 42 \rightsquigarrow \text{fun } y \rightarrow 1764 + y * y$

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 27

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 28

## fonctions

## Noyau fonctionnel

Pas que des types de base...

$t_1$  et  $t_2$  types :  $t_1 \times t_2$  type des couples  $(x_1, x_2)$  avec  $x_1 : t_1$  et  $x_2 : t_2$

$(, , , )$

$\alpha \times (\beta \times \gamma) \rightarrow \delta \neq \alpha \times \beta \times \gamma \rightarrow \delta \neq \alpha \rightarrow \beta \rightarrow \gamma \rightarrow \delta$

Types différents, taille fixe

$\text{let triplet} = (42, \text{true}, \text{"Cooper"}) ; ;$

**Déstructuration**

$\text{let } (t1, t2, t3) = \text{triplet} ; ;$

$\rightsquigarrow$  **Nommages respectifs de 42, true et "Cooper"**

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 27

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 28

## Noyau fonctionnel

Cas particulier des [paires](#)

Fonctions d'accès :

- $fst : \alpha \times \beta \rightarrow \alpha$   
Élément de gauche
- $snd : \alpha \times \beta \rightarrow \beta$   
Élément de droite

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 29

## types produits

## Noyau fonctionnel

Créer ses types...

Déclaration : **type** `t = type`

Affirmation : `( ident : type )`

**type** `quadruplet = int * int * int * int ;;`

**let** `t1 = 1,2,3,4;;`

**let** `t2 : quadruplet = 4,3,2,1;;`

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 30

## déclarations de types

## Noyau fonctionnel

Créer ses types...

Types produits avec noms de champs : [enregistrements](#)

**type** `cplx = { real : float ; im : float };;`

**let** `i = { real = 0.0 ; im = 1.0 }`

Accès : `ident.field_name`

`i.real;;`

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 31

## Noyau fonctionnel

Définir la [structure](#) d'une valeur.

**type** `couleur = | Pique | Coeur (* MAJUSCULE *)`

`| Carreau | Trèfle`

**type** `rang = | Roi | Dame | Valet`

`| Ordinaire of int (* Étiquette int *)`

Différencier par **structure** : [filtrage](#)

**match** `valeur with (* Tout ceci : expression *)`

`| motif1 -> e1`

`| motif2 -> e2`

`| ...`

`(* Vus DANS L'ORDRE *)`

Motif : constructeurs + **variables**

Filtrer :  $\exists?$  remplacement idoine

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 32



## Noyau fonctionnel

## filtrage

Expression donc utilisable partout

```
let f = fun x -> match x with ...
```

```
let f = fun motif -> ... (* Déstructuration *)
```

```
let f = function motifs (* Filtrage *)
```

Structure profonde :

```
type carte = C of (couleur * rang)
```

```
match ident with
| C (Coeur, Dame) -> print_string "Qu'on_lui_coupe_la_tête_!"
| _ -> ...
```

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 37

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 38

## Noyau fonctionnel

## inductifs

Le constructeur construit...

Type : description d'un plus petit ensemble clôt par construction

En général :

- Cas de base
- Construction du suivant

```
type tige = Base | Seg of tige
```

```
match ident with
| Base -> ... (* UN CAS PAR *)
| Seg x -> ... (* CONSTRUCTEUR *)
```

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 37

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 38

## Noyau fonctionnel

## réursion

Fonction récursive : cas de base, hypothèse de récurrence

```
let rec f = ... f ...
```

Exemple : factorielle

- Cas de base :  $0! = 1$

- Hyp.  $n! = v$

```
 $\rightsquigarrow (n+1)! = (n+1) \times v$ 
```

```
let rec fact = fun n ->
```

```
  if n = 0 then 1 else
    n * (fact (n-1))
```

Réursion non terminale : empilement non borné de \*  $\rightsquigarrow$  risques

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 39

## Noyau fonctionnel

## réursion

Exemple : factorielle

Propriété plus générale :  $\forall a, \forall n, (f a n) = a * n!$

- Cas de base :  $\forall a, (f a 0) = a$
  - Hyp.  $\forall a, (f a n) = a \times n!$
- $\rightsquigarrow (f a (n+1)) = (f a \times (n+1)) n = a \times (n+1) \times n! = a \times (n+1)!$
- ```
let rec fact_gen = fun acc -> fun n ->
```

```
  if n = 0 then acc else
    fact_gen (acc * n) (n - 1)
```

Cas particulier

```
let rec fact = fact_gen 1;
```

Réursion terminale : jamais d'empilement trop gros

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 39

XU - ensieie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 40

## Noyau fonctionnel

- Cas de base = constructeurs de base
- Suivant = Constructeur sur hyp. rec.

Hypothèse : on sait faire sur un constituant **structurel**

Distinction des cas : filtrage

## induction

## Noyau fonctionnel

Exemple : factorielle

```
fact 4
  4 * fact 3
  4 * 3 * fact 2
  4 * 3 * 2 * fact 1
  4 * 3 * 2 * 1 * fact 0
  4 * 3 * 2 * 1 * 1
24
```

## terminal/non terminal

## Noyau fonctionnel

Exemple : factorielle

```
fact 4
fact_gen 1 4
fact_gen 4 3
fact_gen 12 2
fact_gen 24 1
fact_gen 24 0
24
```

## terminal/non terminal

## Noyau fonctionnel

Situation de départ



Récursion terminale : **jamais** d'empilement trop gros

## Noyau fonctionnel

Situation d'arrivée, un disque à la fois



XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 45

illustration

## Noyau fonctionnel

Jamais de gros sur un petit : OK



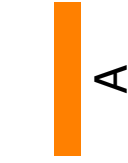
XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 46

illustration

## Noyau fonctionnel

Jamais de gros sur un petit : interdit



XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 47

illustration

## Noyau fonctionnel

Solution pour  $n$  disques ?



XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 48

illustration

## Noyau fonctionnel

## principe de l'inférence

```
let rec f = fun g -> fun n -> fun x ->  
  if n = 0 then x  
  else f g (n-1) (g x);
```

Type de  $f$  ?

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 49

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 50

## Listes

Structure **dynamique**

Mémoire

Soit **vide**, soit élément et liste qui **suit**  $\rightsquigarrow$  **passage au suivant**

Structure

On veut **persistante**

- Vide
- Construction  $e$  en tête de  $l$

Fonctions

- Ajout (en tête ou pas)    Retrait
- Appartenance
- Concaténation

Le mal

## Listes

Structure **dynamique**

- + Ajout **en tête** en temps **constant**
- + **Ordre d'entrée** conservé (FILO)
- Appartenance **linéaire**
- Concaténation **linéaire**

## premiers pas

Mémoire

Bien !

Mal...

## Listes

Type 'a list natif dans Ocaml :

- Base []
- Constructeur ::

Utilisation directe dans filtrage

Notation alternative de la valeur :

[a;b;...;k]

Dans ce cours a::b::...::k::[]

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 51

XU - ensie - Prog. fonctionnelle 2011/2012

UN JEU DE SLIDES N'EST PAS UN POLY DE RÉFÉRENCE 52

## Listes

## itérateurs

Parcours de structure, construction de résultat portant sur structure

`map` liste d'applications de fonction par élément

`map f a1::a2::...::an::[] ~> (f a1)::(f a2)::...::(f an)::[]`

Type :  $(\alpha \rightarrow \beta) \rightarrow \alpha\text{list} \rightarrow \beta\text{list}$

## Listes

## itérateurs

Parcours de structure, construction de résultat portant sur structure

`fold` composition d'applications de fonction par élément

`fold_left f acc a1::a2::...::an::[] ~>`

`f (...(f (f acc a1) a2) ...) an`

Type :  $(\alpha \rightarrow \beta \rightarrow \alpha) \rightarrow \alpha \rightarrow \beta\text{list} \rightarrow \alpha$

`fold_right f a1::a2::...::an::[] acc ~>`

`f a1 (f a2 (...(f an acc) ...))`

Type :  $(\alpha \rightarrow \beta \rightarrow \beta) \rightarrow \alpha\text{list} \rightarrow \beta \rightarrow \beta$

**Non terminal**

## Listes

## itérateurs

Parcours de structure, construction de résultat portant sur structure

`fold` composition d'applications de fonction par élément

`fold_left f acc a1::a2::...::an::[] ~>`

`f (...(f (f acc a1) a2) ...) an`

Type :  $(\alpha \rightarrow \beta \rightarrow \alpha) \rightarrow \alpha \rightarrow \beta\text{list} \rightarrow \alpha$

**Terminal**

## Noyau encore

## exceptions

Opérations partielles : valeur exceptionnelle en cas d'erreur

`# 1/0 ;`

Exception: `Division_by_zero`

... en cas de domaine de définition dépassé

Exception: `Invalid_argument`

... pour interrompre une évaluation

éventuellement la reprendre sur une autre expression !

## Noyau encore

Définition :

```
exception Toto
```

```
exception Terrible_erreur of string
```

(mais pas polymorphe)

Levée d'une exception :

```
raise Toto
```

```
raise (Terrible_erreur "subjonctif_après_«_après_que_»")
```

## Noyau encore

Rattrapage: construction `try ... with`

```
try e1 with Motif_exc -> e2
```

1. Évaluation de  $e_1$ , sans exception : calcul terminé

2. Levée dans  $e_1$  :

- Filtrée par `Motif_exc` alors calcul poursuivi
- Non filtrée : exception propagée

valeur  $e_1$

valeur  $e_2$

ni  $e_1$  ni  $e_2$  évalués

## exceptions