PF chapître 3 : évaluation

Jean-François Monin

POLYTECH GRENOBLE

Plan

Évaluation

Évaluer une expression $\underline{\underline{\text{déf}}}$ *déterminer sa valeur*

Mais qu'est-ce qu'une *valeur* ?

Deux points de vue

Point de vue dénotationnel

Les valeurs sont dans un monde (mathématique) extérieur aux expressions.

Exemple : $(15 + 3 * 3)$ et $(15 + 9)$ dénotent la même valeur, qui est l'entier mathématique 24.

Deux points de vue

Point de vue dénotationnel

Les valeurs sont dans un monde (mathématique) extérieur aux expressions.

Exemple : $(15 + 3 * 3)$ et $(15 + 9)$ dénotent la même valeur, qui est l'entier mathématique 24.

Point de vue opérationnel

Les valeurs sont des expressions particulières
= celles qui ne peuvent pas être réduites
et reflètent ainsi un résultat.

Exemple : l'expression 24 est (reflète) une valeur, contrairement à $(15 + 3 * 3)$ et $(15 + 9)$.

Comment évaluer

On réduit progressivement
$\longrightarrow$ relation de *réduction* notée $\triangleright$

Comment évaluer

On réduit progressivement
$\longrightarrow$ relation de *réduction* notée $\triangleright$

Exemple

$$15 + 9 - 4 * 2$$

Comment évaluer

On réduit progressivement
$\rightarrow$ relation de *réduction* notée $\triangleright$

Exemple

$$15 + 9 - 4 * 2$$

$$\triangleright$$

$$24 - 4 * 2$$

Comment évaluer

On réduit progressivement
$\rightarrow$ relation de *réduction* notée $\triangleright$

Exemple

$$15 + 9 - 4 * 2$$
$$\triangleright$$
$$24 - 4 * 2$$
$$\triangleright$$
$$24 - 8$$

Comment évaluer

On réduit progressivement
$\rightarrow$ relation de *réduction* notée $\triangleright$

Exemple

$$15 + 9 - 4 * 2$$
$$\triangleright$$
$$24 - 4 * 2$$
$$\triangleright$$
$$24 - 8$$
$$\triangleright$$
$$16$$

Expressions et valeurs

$5 + 4$, 3×3, 9, $\sqrt{81}$, `if true then 9 else 6`
false, `true && not true`
$157.03 + 89.75 - 4 \times 45.35$, $\frac{(124.63 + 38.82)}{2.5}$ et 65.38
sont des exemples d'*expressions*
Parmi celles-ci, seules 9, false et 65.38 sont des *valeurs*.

Comment les distinguer ?

Expressions et valeurs

$5 + 4$, 3×3, 9, $\sqrt{81}$, `if true then 9 else 6`
false, `true && not true`
$157.03 + 89.75 - 4 \times 45.35$, $\frac{(124.63 + 38.82)}{2.5}$ et 65.38
sont des exemples d'*expressions*
Parmi celles-ci, seules 9, false et 65.38 sont des *valeurs*.

Comment les distinguer ?
Pas d'opération appliquée à des opérandes

Expressions et valeurs

$5 + 4$, 3×3, 9, $\sqrt{81}$, `if true then 9 else 6`
false, `true && not true`
$157.03 + 89.75 - 4 \times 45.35$, $\frac{(124.63 + 38.82)}{2.5}$ et 65.38
sont des exemples d'*expressions*
Parmi celles-ci, seules 9, false et 65.38 sont des *valeurs*.

Comment les distinguer ?
Pas d'opération appliquée à des opérandes
Plus généralement : expression irréductible
(aucune opportunité de réduire)

Comparaison

Deux valeurs sont égales si et seulement si elles sont représentées par des expressions irréductibles littéralement (syntaxiquement) identiques.

Algorithme de comparaison entre les valeurs dénotées par deux expressions E_1 et E_2

- évaluer $E_1 \rightarrow$ on obtient v_1
- évaluer $E_2 \rightarrow$ on obtient v_2
- comparer syntaxiquement v_1 et v_2

Discussion délicate

Cohérence

- La valeur (dénotée) est invariante lors de la réduction.
- Confluence : la valeur obtenue par réduction ne dépend pas de la stratégie de réduction.
- Une valeur (dénotationnelle) est représentée par *au plus* une expression irréductible.

Complétude

- Certaines expressions ne dénotent aucune valeur claire.
  En particulier : problème de la terminaison.
- Une valeur (dénotationnelle) est-elle représentée par *au moins* une expression irréductible ?
  PAS toujours mais moins important que la cohérence.

Bilan (1)

Approche dénotationnelle

- Plus propre, avec une distinction claire entre l'univers syntaxique des expressions et l'univers sémantique des valeurs.
- Simple dans le cas des expressions arithmétiques « usuelles » (simplement composées à partir des opérations arithmétiques telles que l'addition etc.)
- De même pour les expresions booléennes « usuelles ».
- En revanche la notion de valeur se complique avec les fonctions notamment récursives il faut des espaces mathématiques beaucoup plus complexes (treillis, ...)

C'est donc l'approche adoptée au début de LT, par exemple : `eval` qui prend en entrée un AST de type `aexp` et rend une valeur de type `nat`.

Bilan (2)

Approche opérationnelle

- Moins propre.
- Par exemple le problème de la confluence se pose.
- Mais les constructions donnant des calculs pouvant ne pas terminer ne demandent pas de nouveaux concepts mathématiques.

Plan

On adopte le point de vue opérationnel

On va définir une relation de réduction $\triangleright$

en considérant tour à tour les différentes formes d'expression

Convention

Pour alléger, on considère que la relation de réduction est réflexive et transitive : lorsque $E_1 \triangleright E_2 \ldots \triangleright E_n$ on peut écrire $E_1 \triangleright E_n$.

En particulier, pour dire que l'évaluation de E donne V, on écrit simplement $E \triangleright V$,
étant entendu que V est une expression irréductible.

Plan

Évaluation d'une expression constante

Expression =

- valeur $\Rightarrow$ rien à calculer

Évaluation d'une expression constante

Expression =

- valeur $\Rightarrow$ rien à calculer
- fonction appliquée à des arguments (sous-expressions) :
  - évaluation des arguments a_i dans un ordre non spécifié
  - application de la fonction aux valeurs obtenues

Évaluation d'une expression constante

Expression =

- valeur $\Rightarrow$ rien à calculer
- fonction appliquée à des arguments (sous-expressions) :
  - évaluation des arguments a_i dans un ordre non spécifié
  - application de la fonction aux valeurs obtenues

Exemple

mult (plus 2 3) (succ (moins 5 (-3)))
▷ mult (plus 2 3) (succ 8)
▷ mult 5 (succ 8)
▷ mult 5 9
▷ 45

Expression conditionnelle

Expression =

- valeur (exemple : un entier)
- fonction appliquée à des arguments : $f\, a_1 \ldots a_n$
- `if` B `then` E_1 `else` E_2
- *autres possibilités vues plus tard*

Évaluation d'une expression conditionnelle

- évaluation de B, i.e. $B \triangleright V$
- $V =$ `true`, évaluation de E_1, i.e. $E_1 \triangleright V_1$
  (`if` B `then` E_1 `else` E_2) $\triangleright V_1$
- $V =$ `false`, évaluation de E_2, i.e. $E_2 \triangleright V_2$
  (`if` B `then` E_2 `else` E_2) $\triangleright V_2$

Plan

Expressions en **let**

Rappel

Ce n'est pas une affectation

Permet de rendre le code plus lisible et de factoriser

let $gros = 37938573 \times 4869582/8576$ `in`
$(2 + gros - 27 \times 31) \times \text{succ}\,(5 \times 27 - gros)$

Environnement (contexte d'évaluation)

Définition

$$\underset{\text{déf}}{\textit{environnement}} = \text{ensemble d'associations nom} \mapsto \text{valeur}$$

L'environnement est parfois appelé *contexte d'évaluation*

En programmation fonctionnelle

- associations **définitives** dans la portée considérée
- les valeurs sont **immuables**

Plan

Évaluation dans un environnement

Expression =

- valeur (exemple : un entier)
- nom
- fonction appliquée à des arguments : $f\, a_1 \ldots a_n$
- let-expression : **let** nom = *expr1* `in` *expr2*

Évaluation dans un environnement

Expression =

- valeur (exemple : un entier)
- nom
- fonction appliquée à des arguments : $f\, a_1 \ldots a_n$
- let-expression : let nom = *expr1* in *expr2*

Évaluation avec environnement

- valeur $\Rightarrow$ rien à calculer

Évaluation dans un environnement

Expression =

- valeur (exemple : un entier)
- nom
- fonction appliquée à des arguments : $f\, a_1 \ldots a_n$
- let-expression : let nom = *expr1* in *expr2*

Évaluation avec environnement

- valeur $\Rightarrow$ rien à calculer
- nom x $\Rightarrow$ valeur V trouvée dans l'environnement (l'environnement **doit** contenir $x \mapsto$ V)

Évaluation dans un environnement

Expression =

- valeur (exemple : un entier)
- nom
- fonction appliquée à des arguments : $f\, a_1 \ldots a_n$
- let-expression : let nom = *expr1* in *expr2*

Évaluation avec environnement

- valeur $\Rightarrow$ rien à calculer
- nom $x \Rightarrow$ valeur V trouvée dans l'environnement (l'environnement **doit** contenir $x \mapsto V$)
- fonction appliquée à des arguments (sous-expressions) :
  - évaluation des arguments a_i dans un ordre non spécifié
  - *un petit quelque chose, voir plus loin*
  - application de la fonction aux valeurs obtenues

Évaluation dans un environnement

Expression =

- valeur (exemple : un entier)
- nom
- fonction appliquée à des arguments : $f\, a_1 \ldots a_n$
- let-expression : **let** nom = *expr1* `in` *expr2*

Évaluation avec environnement

- valeur $\Rightarrow$ rien à calculer
- nom $x \Rightarrow$ valeur V trouvée dans l'environnement (l'environnement **doit** contenir $x \mapsto V$)
- fonction appliquée à des arguments (sous-expressions) :
  - évaluation des arguments a_i dans un ordre non spécifié
  - *un petit quelque chose, voir plus loin*
  - application de la fonction aux valeurs obtenues
- let-expression : cf. suite

Évaluation d'un **let**... `in`...

let nom = *expr1* `in` *expr2*

- évaluation de *expr1* : $\triangleright$ *val1*
- environnement augmenté = ancien environnement + nom $\mapsto$ *val1*
- évaluation de *expr2* dans l'environnement augmenté

Évaluation d'un **let**... `in`...

Exemple 1

let $y = 2 + 3$ `in` **let** $x = 32 \times 3$ `in` **let** $z = 75/3$ `in` f x y z

Évaluation d'un **let**... in...

Exemple 1

$$\textbf{let } y = 2 + 3 \texttt{ in } \textbf{let } x = 32 \times 3 \texttt{ in } \textbf{let } z = 75/3 \texttt{ in } \mathrm{f}\ x\ y\ z$$

$$[y \mapsto 5] \qquad \textbf{let } x = 32 \times 3 \texttt{ in } \textbf{let } z = 75/3 \texttt{ in } \mathrm{f}\ x\ y\ z$$

Évaluation d'un **let**... in...

Exemple 1

$$\textbf{let } y = 2 + 3 \texttt{ in } \textbf{let } x = 32 \times 3 \texttt{ in } \textbf{let } z = 75/3 \texttt{ in } \mathrm{f}\ x\ y\ z$$

$$[y \mapsto 5] \qquad \textbf{let } x = 32 \times 3 \texttt{ in } \textbf{let } z = 75/3 \texttt{ in } \mathrm{f}\ x\ y\ z$$

$$[y \mapsto 5,\ x \mapsto 96] \qquad \textbf{let } z = 75/3 \texttt{ in } \mathrm{f}\ x\ y\ z$$

Évaluation d'un **let**... in...

Exemple 1

$$\textbf{let } y = 2 + 3 \texttt{ in } \textbf{let } x = 32 \times 3 \texttt{ in } \textbf{let } z = 75/3 \texttt{ in } \mathsf{f}\ x\ y\ z$$
$$[y \mapsto 5] \qquad \textbf{let } x = 32 \times 3 \texttt{ in } \textbf{let } z = 75/3 \texttt{ in } \mathsf{f}\ x\ y\ z$$
$$[y \mapsto 5,\ x \mapsto 96] \qquad \textbf{let } z = 75/3 \texttt{ in } \mathsf{f}\ x\ y\ z$$
$$[y \mapsto 5,\ x \mapsto 96,\ z \mapsto 25] \qquad \mathsf{f}\ x\ y\ z$$

Évaluation d'un **let**... in...

Exemple 1

let $y = 2+3$ in **let** $x = 32 \times 3$ in **let** $z = 75/3$ in f x y z		
$[y \mapsto 5]$	**let** $x = 32 \times 3$ in **let** $z = 75/3$ in f x y z	
$[y \mapsto 5, x \mapsto 96]$	**let** $z = 75/3$ in f x y z	
$[y \mapsto 5, x \mapsto 96, z \mapsto 25]$	f x y z	
$[y \mapsto 5, x \mapsto 96, z \mapsto 25]$	f 96 5 25	
$[y \mapsto 5, x \mapsto 96, z \mapsto 25]$	*...(application de la fonction)*	

Exemple 2

let $y = 2+3$ in **let** $x = 32 \times 3$ in **let** $z = x+y$ in f x y z

Évaluation d'un **let**... in...

Exemple 1

let $y = 2+3$ in **let** $x = 32 \times 3$ in **let** $z = 75/3$ in f x y z		
$[y \mapsto 5]$	**let** $x = 32 \times 3$ in **let** $z = 75/3$ in f x y z	
$[y \mapsto 5, x \mapsto 96]$	**let** $z = 75/3$ in f x y z	
$[y \mapsto 5, x \mapsto 96, z \mapsto 25]$	f x y z	
$[y \mapsto 5, x \mapsto 96, z \mapsto 25]$	f 96 5 25	
$[y \mapsto 5, x \mapsto 96, z \mapsto 25]$	*...(application de la fonction)*	

Exemple 2

let $y = 2+3$ in **let** $x = 32 \times 3$ in **let** $z = x+y$ in f x y z	
$[y \mapsto 5, x \mapsto 96]$	**let** $z = x+y$ in f x y z

Évaluation d'un **let**... in...

Exemple 1

let $y = 2 + 3$ in **let** $x = 32 \times 3$ in **let** $z = 75/3$ in f x y z

$[y \mapsto 5]$ **let** $x = 32 \times 3$ in **let** $z = 75/3$ in f x y z

$[y \mapsto 5, x \mapsto 96]$ **let** $z = 75/3$ in f x y z

$[y \mapsto 5, x \mapsto 96, z \mapsto 25]$ f x y z

$[y \mapsto 5, x \mapsto 96, z \mapsto 25]$ f 96 5 25

$[y \mapsto 5, x \mapsto 96, z \mapsto 25]$ *...(application de la fonction)*

Exemple 2

let $y = 2 + 3$ in **let** $x = 32 \times 3$ in **let** $z = x + y$ in f x y z

$[y \mapsto 5, x \mapsto 96]$ **let** $z = x + y$ in f x y z

$[y \mapsto 5, x \mapsto 96, z \mapsto 101]$ f x y z

Évaluation d'un **let**... in...

Exemple 1

let $y = 2+3$ in **let** $x = 32 \times 3$ in **let** $z = 75/3$ in f x y z		
$[y \mapsto 5]$	**let** $x = 32 \times 3$ in **let** $z = 75/3$ in f x y z	
$[y \mapsto 5, x \mapsto 96]$	**let** $z = 75/3$ in f x y z	
$[y \mapsto 5, x \mapsto 96, z \mapsto 25]$	f x y z	
$[y \mapsto 5, x \mapsto 96, z \mapsto 25]$	f 96 5 25	
$[y \mapsto 5, x \mapsto 96, z \mapsto 25]$	*...(application de la fonction)*	

Exemple 2

let $y = 2+3$ in **let** $x = 32 \times 3$ in **let** $z = x+y$ in f x y z		
$[y \mapsto 5, x \mapsto 96]$	**let** $z = x+y$ in f x y z	
$[y \mapsto 5, x \mapsto 96, z \mapsto 101]$	f x y z	
$[y \mapsto 5, x \mapsto 96, z \mapsto 101]$	f 96 5 101	
$[y \mapsto 5, x \mapsto 96, z \mapsto 101]$	*...(application de la fonction)*	

Contrôle de l'ordre d'évaluation

Remarque

L'ordre d'évaluation dans un **let**... `in`... est bien déterminé

- sans grande importance dans un cadre purement fonctionnel
- important en cas d'*effets de bord*
  - Entrées/sorties
  - Exceptions

(sera approfondi plus tard)

Plan

Encore des valeurs : les fonctions (1)

Exemple : la fonction successeur

Notation ML (Ocaml) : `fun` x $\rightarrow$ $x + 1$
Notation mathématique abrégée : $\lambda x.\ x + 1$

Évaluation

On considère l'application d'une valeur fonctionnelle $\lambda x.$ *corps*
à un argument *arg*
$(\lambda x.$ *corps*$)$ *arg* se réduit comme
let $x =$ *arg* `in` *corps*

- *corps* et *arg* sont des expressions
- en OCaml, *arg* est en fait une valeur
  (stratégie dite « stricte »)

Encore des valeurs : les fonctions (2)

Exemple de réduction

$$\textbf{let}\ \mathit{suc} = \texttt{fun}\, x \to\ x + 1\ \texttt{in}\ \textbf{let}\ a = 12\ \texttt{in}\ \mathit{suc}\ (a + a)$$

Encore des valeurs : les fonctions (2)

Exemple de réduction

$$\textbf{let } suc = \texttt{fun}\, x \to x + 1 \texttt{ in } \textbf{let } a = 12 \texttt{ in } suc\ (a + a)$$
$$[suc \mapsto \lambda x.\ x + 1] \qquad \textbf{let } a = 12 \texttt{ in } suc\ (a + a)$$
$$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12] \qquad suc\ (a + a)$$

Encore des valeurs : les fonctions (2)

Exemple de réduction

let $suc = \texttt{fun}\, x \to x + 1$ `in` **let** $a = 12$ `in` $suc\ (a + a)$

$[suc \mapsto \lambda x.\ x + 1]$ **let** $a = 12$ `in` $suc\ (a + a)$

$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12]$ $suc\ (a + a)$

(evaluation de $a + a$ *:*

$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12]\ \ a + a \triangleright a + 12 \triangleright 12 + 12 \triangleright 24)$

Encore des valeurs : les fonctions (2)

Exemple de réduction

$$\textbf{let } suc = \texttt{fun}\, x \to x + 1 \texttt{ in } \textbf{let } a = 12 \texttt{ in } suc\ (a + a)$$

$$[suc \mapsto \lambda x.\ x + 1] \qquad \textbf{let } a = 12 \texttt{ in } suc\ (a + a)$$

$$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12] \qquad suc\ (a + a)$$

(evaluation de $a + a$ *:*

$$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12]\ \ a + a \triangleright a + 12 \triangleright 12 + 12 \triangleright 24)$$

$$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12] \qquad suc\ 24$$

Encore des valeurs : les fonctions (2)

Exemple de réduction

let $suc = \texttt{fun}\, x \to x + 1$ `in` **let** $a = 12$ `in` $suc\ (a + a)$

$[suc \mapsto \lambda x.\ x + 1]$ **let** $a = 12$ `in` $suc\ (a + a)$

$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12]$ $suc\ (a + a)$

(evaluation de $a + a$:

$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12]$ $a + a \triangleright a + 12 \triangleright 12 + 12 \triangleright 24)$

$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12]$ $suc\ 24$

$[\ldots]$ $(\lambda x.\ x + 1)\ 24$

Encore des valeurs : les fonctions (2)

Exemple de réduction

let $suc = \texttt{fun}\, x \to x + 1$ `in` **let** $a = 12$ `in` $suc\ (a + a)$		
$[suc \mapsto \lambda x.\ x + 1]$	**let** $a = 12$ `in` $suc\ (a + a)$	
$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12]$	$suc\ (a + a)$	
	(evaluation de $a + a$ *:*	
$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12]$	$a + a \triangleright a + 12 \triangleright 12 + 12 \triangleright 24)$	
$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12]$	$suc\ 24$	
$[\ldots]$	$(\lambda x.\ x + 1)\ 24$	
$[\ldots,\ x \mapsto 24]$	$x + 1$	

Encore des valeurs : les fonctions (2)

Exemple de réduction

let $suc = \texttt{fun}\, x \to x + 1$ `in` **let** $a = 12$ `in` $suc\ (a + a)$		
$[suc \mapsto \lambda x.\ x + 1]$	**let** $a = 12$ `in` $suc\ (a + a)$	
$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12]$	$suc\ (a + a)$	
	(evaluation de $a + a$ *:*	
$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12]$	$a + a \triangleright a + 12 \triangleright 12 + 12 \triangleright 24$)	
$[suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12]$	$suc\ 24$	
$[\ldots]$	$(\lambda x.\ x + 1)\ 24$	
$[\ldots,\ x \mapsto 24]$	$x + 1$	
$[\ldots,\ x \mapsto 24]$	$24 + 1$	
$[\ldots,\ x \mapsto 24]$	25	

Encore des valeurs : les fonctions (3)

Que représente « . . . » dans $[\ldots,\ x \mapsto 24]$?

- L'environnement courant $\mathit{suc} \mapsto \lambda x.\ x + 1,\ a \mapsto 12$?
- Autre chose ?

Encore des valeurs : les fonctions (3)

Que représente « . . . » dans $[\ldots,\ x \mapsto 24]$?

- L'environnement courant $suc \mapsto \lambda x.\ x + 1,\ a \mapsto 12$?
- Autre chose ?

Utiliser la portée où la fonction appelée est définie

Dans l'exemple précédent où cette fonction est *suc*, cela n'a pas d'incidence.
Mais ce n'est pas le cas général.

Encore des valeurs : les fonctions (4)

Utiliser la portée où la fonction appelée est définie

let $a = 3$ `in` **let** $pla = \texttt{fun}\, x \rightarrow x + a$ `in` **let** $a = 12$ `in` $pla\ (a + a)$

Encore des valeurs : les fonctions (4)

Utiliser la portée où la fonction appelée est définie

$$\textbf{let } a = 3 \texttt{ in } \textbf{let } pla = \texttt{fun}\, x \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3] \qquad \textbf{let } pla = \texttt{fun}\, x \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a] \qquad \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a,\ a \mapsto 12] \qquad pla\ (a + a)$$

(evaluation de $a + a$:

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a,\ a \mapsto 12] \quad a + a \triangleright a + 12 \triangleright 12 + 12 \triangleright 24)$$

Encore des valeurs : les fonctions (4)

Utiliser la portée où la fonction appelée est définie

$$\textbf{let } a = 3 \texttt{ in } \textbf{let } pla = \texttt{fun x} \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3] \qquad \textbf{let } pla = \texttt{fun x} \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a] \qquad \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a,\ a \mapsto 12] \qquad pla\ (a + a)$$

(evaluation de $a + a$:

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a,\ a \mapsto 12] \quad a + a \triangleright a + 12 \triangleright 12 + 12 \triangleright 24)$$

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a,\ a \mapsto 12] \qquad pla\ 24$$

Encore des valeurs : les fonctions (4)

Utiliser la portée où la fonction appelée est définie

$$\textbf{let } a = 3 \texttt{ in } \textbf{let } pla = \texttt{fun x} \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3] \qquad \textbf{let } pla = \texttt{fun x} \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a] \qquad \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a,\ a \mapsto 12] \qquad pla\ (a + a)$$

(evaluation de $a + a$:

$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a,\ a \mapsto 12]$ $\quad a + a \triangleright a + 12 \triangleright 12 + 12 \triangleright 24)$

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a,\ a \mapsto 12] \qquad pla\ 24$$

$$[a \mapsto 3] \qquad (\lambda x.\ x + a)\ 24$$

Encore des valeurs : les fonctions (4)

Utiliser la portée où la fonction appelée est définie

$$\textbf{let } a = 3 \texttt{ in } \textbf{let } pla = \texttt{fun x} \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3] \qquad \textbf{let } pla = \texttt{fun x} \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a] \qquad \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a,\ a \mapsto 12] \qquad pla\ (a + a)$$

(evaluation de $a + a$:

$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a,\ a \mapsto 12]\ \ a + a \triangleright a + 12 \triangleright 12 + 12 \triangleright 24)$

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a,\ a \mapsto 12] \qquad pla\ 24$$

$$[a \mapsto 3] \qquad (\lambda x.\ x + a)\ 24$$

$$[a \mapsto 3,\ x \mapsto 24] \qquad x + a$$

Encore des valeurs : les fonctions (4)

Utiliser la portée où la fonction appelée est définie

$$\textbf{let } a = 3 \texttt{ in } \textbf{let } pla = \texttt{fun x} \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3] \qquad \textbf{let } pla = \texttt{fun x} \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a] \qquad \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a,\ a \mapsto 12] \qquad pla\ (a + a)$$

(evaluation de $a + a$:

$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a,\ a \mapsto 12]\ \ a + a \triangleright a + 12 \triangleright 12 + 12 \triangleright 24$*)*

$$[a \mapsto 3,\ pla \mapsto \lambda x.\ x + a,\ a \mapsto 12] \qquad pla\ 24$$

$$[a \mapsto 3] \qquad (\lambda x.\ x + a)\ 24$$

$$[a \mapsto 3,\ x \mapsto 24] \qquad x + a$$

$$[a \mapsto 3,\ x \mapsto 24] \qquad 24 + 3$$

$$[a \mapsto 3,\ x \mapsto 24] \qquad 27$$

Encore des valeurs : les fonctions (5)

Conserver la portée où la fonction appelée est définie

let $a = 3$ `in` **let** $pla =$ `fun` $x \rightarrow x + a$ `in` **let** $a = 12$ `in` $pla\ (a + a)$
$[a \mapsto 3]$ **let** $pla =$ `fun` $x \rightarrow x + a$ `in` **let** $a = 12$ `in` $pla\ (a + a)$

Encore des valeurs : les fonctions (5)

Conserver la portée où la fonction appelée est définie

$$\textbf{let } a = 3 \texttt{ in } \textbf{let } pla = \texttt{fun}\, x \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$
$$[a \mapsto 3] \qquad \textbf{let } pla = \texttt{fun}\, x \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$
$$[a \mapsto 3,\ pla \mapsto ([a \mapsto 3],\ \lambda x.\ x + a)] \qquad \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

Encore des valeurs : les fonctions (5)

Conserver la portée où la fonction appelée est définie

$$\textbf{let } a = 3 \texttt{ in } \textbf{let } pla = \texttt{fun}\, x \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$
$$[a \mapsto 3] \qquad \textbf{let } pla = \texttt{fun}\, x \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$
$$[a \mapsto 3,\ pla \mapsto ([a \mapsto 3],\ \lambda x.\ x + a)] \qquad \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$
$$[a \mapsto 3,\ pla \mapsto ([a \mapsto 3],\ \lambda x.\ x + a),\ a \mapsto 12] \qquad pla\ (a + a)$$
$$\textit{(evaluation de } a + a \textit{ :}$$
$$[a \mapsto 3,\ pla \mapsto \ldots,\ a \mapsto 12] \quad a + a \triangleright a + 12 \triangleright 12 + 12 \triangleright 24)$$
$$[a \mapsto 3,\ pla \mapsto ([a \mapsto 3],\ \lambda x.\ x + a),\ a \mapsto 12] \qquad pla\ 24$$

Encore des valeurs : les fonctions (5)

Conserver la portée où la fonction appelée est définie

$$\textbf{let } a = 3 \texttt{ in } \textbf{let } pla = \texttt{fun}\, x \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3] \qquad \textbf{let } pla = \texttt{fun}\, x \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3,\ pla \mapsto ([a \mapsto 3],\ \lambda x.\ x + a)] \qquad \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3,\ pla \mapsto ([a \mapsto 3],\ \lambda x.\ x + a),\ a \mapsto 12] \qquad pla\ (a + a)$$

(evaluation de $a + a$:

$$[a \mapsto 3,\ pla \mapsto \ldots,\ a \mapsto 12] \quad a + a \triangleright a + 12 \triangleright 12 + 12 \triangleright 24)$$

$$[a \mapsto 3,\ pla \mapsto ([a \mapsto 3],\ \lambda x.\ x + a),\ a \mapsto 12] \qquad pla\ 24$$

$$[a \mapsto 3] \qquad (\lambda x.\ x + a)\ 24$$

Encore des valeurs : les fonctions (5)

Conserver la portée où la fonction appelée est définie

$$\textbf{let } a = 3 \texttt{ in } \textbf{let } pla = \texttt{fun}\, x \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3] \qquad \textbf{let } pla = \texttt{fun}\, x \to x + a \texttt{ in } \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3,\ pla \mapsto ([a \mapsto 3],\ \lambda x.\ x + a)] \qquad \textbf{let } a = 12 \texttt{ in } pla\ (a + a)$$

$$[a \mapsto 3,\ pla \mapsto ([a \mapsto 3],\ \lambda x.\ x + a),\ a \mapsto 12] \qquad pla\ (a + a)$$

(evaluation de $a + a$:

$[a \mapsto 3,\ pla \mapsto \ldots,\ a \mapsto 12]$ $a + a \triangleright a + 12 \triangleright 12 + 12 \triangleright 24$)

$$[a \mapsto 3,\ pla \mapsto ([a \mapsto 3],\ \lambda x.\ x + a),\ a \mapsto 12] \qquad pla\ 24$$

$$[a \mapsto 3] \qquad (\lambda x.\ x + a)\ 24$$

$$[a \mapsto 3,\ x \mapsto 24] \qquad x + a$$

$$[a \mapsto 3,\ x \mapsto 24] \qquad 24 + 3$$

$$[a \mapsto 3,\ x \mapsto 24] \qquad 27$$

Retour sur l'évaluation

Expression =

- valeur (exemple : un entier)
- nom
- fonction appliquée à des arguments : $f\, a_1 \ldots a_n$
- let-expression : let nom = *expr1* in *expr2*

Évaluation

- valeur $\Rightarrow$ rien à calculer
- nom x $\Rightarrow$ valeur V trouvée dans l'environnement
- fonction appliquée à des arguments (sous-expressions) :
  - évaluation des arguments a_i dans un ordre non spécifié
  - *évaluation de f*
  - application de la fonction *obtenue* aux valeurs obtenues
- **let** x = *expr1* `in` *expr2* : évaluation de *expr2* avec $x \mapsto V_1$

Plan

Définition récursive de fonction

Exemple

let rec fact = `fun` n → `if` $n = 0$ `then` 1 `else` n * fact $(n - 1)$

Évaluation

- valeur ⇒ rien à calculer (rem : `fun` y → *expr* est une valeur !)
- ...
- **let** x = *expr1* `in` *expr2* : évaluation de *expr1* $\triangleright V_1$,
  puis de *expr2* dans l'environnement augmenté de $x \mapsto V_1$
- **let rec** x = *expr1* `in` *expr2* : évaluation de *expr2*
  dans l'environnement augmenté de $x \mapsto$ *expr1*
  OK si *expr1* est de la forme `fun` y → *expr*

Exemple fact 2

```
let rec fact = fun n -> if n = 0 then 1 else n * fact (n - 1)
    in fact 2;;
```

Exemple fact 2

$$\texttt{let rec } \mathit{fact} = \texttt{fun n} \rightarrow \texttt{ if } n = 0 \texttt{ then } 1 \texttt{ else } n * \mathit{fact}\ (n-1)$$
$$\texttt{in } \mathit{fact}\ 2\ ;;$$
$$\triangleright\ [\mathit{fact} \mapsto ([\mathit{fact} \mapsto (.,.)]\ ,\ \lambda n\ .\ \texttt{if } n = 0 \texttt{ then } 1 \texttt{ else } n * \mathit{fact}\ (n-1))]$$
$$\mathit{fact}\ 2$$

Exemple fact 2

let rec *fact* = `fun` n → `if` *n* = 0 `then` 1 `else` *n* * *fact* (*n* − 1)
`in` *fact* 2 ;;

▷ [*fact* ↦ ([*fact* ↦ (., .)] , λ *n* . `if` *n* = 0 `then` 1 `else` *n* * *fact* (*n* − 1))]
fact 2

▷ [*fact* ↦ (., .), *n* ↦ 2] `if` *n* = 0 `then` 1 `else` *n* * *fact* (*n* − 1)

Exemple fact 2

let rec *fact* = `fun` n → `if` *n* = 0 `then` 1 `else` *n* * *fact* (*n* − 1)
`in` *fact* 2 ;;

▷ [*fact* ↦ ([*fact* ↦ (., .)] , λ *n* . `if` *n* = 0 `then` 1 `else` *n* * *fact* (*n* − 1))]
fact 2

▷ [*fact* ↦ (., .), *n* ↦ 2] `if` *n* = 0 `then` 1 `else` *n* * *fact* (*n* − 1)

▷ [*fact* ↦ (., .), *n* ↦ 2] *n* * *fact* (*n* − 1)

Exemple fact 2

let rec *fact* = `fun` n → `if` $n = 0$ `then` 1 `else` n * *fact* $(n-1)$
`in` *fact* 2 ;;

▷ $[fact \mapsto ([fact \mapsto (.,.)]\ ,\ \lambda n\ .$ `if` $n = 0$ `then` 1 `else` n * *fact* $(n-1))]$
fact 2

▷ $[fact \mapsto (.,.),\ n \mapsto 2]$ `if` $n = 0$ `then` 1 `else` n * *fact* $(n-1)$

▷ $[fact \mapsto (.,.),\ n \mapsto 2]$ n * *fact* $(n-1)$

▷ $[fact \mapsto (.,.),\ n \mapsto 2]$ 2 * *fact* $(2-1)$

Exemple fact 2

let rec *fact* = fun n → if $n = 0$ then 1 else n * *fact* $(n-1)$
in *fact* 2 ;;

▷ $[fact \mapsto ([fact \mapsto (.,.)]\ ,\ \lambda n\ .$ if $n = 0$ then 1 else n * *fact* $(n-1))]$ *fact* 2

▷ $[fact \mapsto (.,.),\ n \mapsto 2]$ if $n = 0$ then 1 else n * *fact* $(n-1)$

▷ $[fact \mapsto (.,.),\ n \mapsto 2]$ n * *fact* $(n-1)$

▷ $[fact \mapsto (.,.),\ n \mapsto 2]$ 2 * *fact* $(2-1)$

▷ $[fact \mapsto (.,.),\ n \mapsto 2]$ 2 * *fact* 1

Exemple fact 2

$$\texttt{let rec } fact = \texttt{fun n} \rightarrow \texttt{if } n = 0 \texttt{ then } 1 \texttt{ else } n * fact\ (n-1) \texttt{ in } fact\ 2\ ;;$$

$$\triangleright\ [fact \mapsto ([fact \mapsto (.,.)]\ ,\ \lambda n\ .\ \texttt{if } n = 0 \texttt{ then } 1 \texttt{ else } n * fact\ (n-1))] \qquad fact\ 2$$

$$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2] \qquad \texttt{if } n = 0 \texttt{ then } 1 \texttt{ else } n * fact\ (n-1)$$

$$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2] \qquad n * fact\ (n-1)$$

$$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2] \qquad 2 * fact\ (2-1)$$

$$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2] \qquad 2 * fact\ 1$$

$$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2,\ n \mapsto 1] \qquad 2 * (\texttt{if } n = 0 \texttt{ then } 1 \texttt{ else } n * fact\ (n-1))$$

$$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2,\ n \mapsto 1] \qquad 2 * (n * fact\ (n-1))$$

Exemple fact 2

let rec *fact* = fun n → if $n = 0$ then 1 else n * *fact* $(n-1)$
in *fact* 2 ;;

▷ $[fact \mapsto ([fact \mapsto (.,.)]\ ,\ \lambda n\ .$ if $n = 0$ then 1 else n * *fact* $(n-1))]$ *fact* 2

▷ $[fact \mapsto (.,.),\ n \mapsto 2]$ if $n = 0$ then 1 else n * *fact* $(n-1)$

▷ $[fact \mapsto (.,.),\ n \mapsto 2]$ n * *fact* $(n-1)$

▷ $[fact \mapsto (.,.),\ n \mapsto 2]$ 2 * *fact* $(2-1)$

▷ $[fact \mapsto (.,.),\ n \mapsto 2]$ 2 * *fact* 1

▷ $[fact \mapsto (.,.),\ n \mapsto 2,\ n \mapsto 1]$ 2 * (if $n = 0$ then 1 else n * *fact* $(n-1)$)

▷ $[fact \mapsto (.,.),\ n \mapsto 2,\ n \mapsto 1]$ 2 * $(n$ * *fact* $(n-1))$

▷ $[fact \mapsto (.,.),\ n \mapsto 2,\ n \mapsto 1]$ 2 * (1 * *fact* 0)

Exemple fact 2

$$\textbf{let rec } fact = \texttt{fun n} \rightarrow \texttt{if } n = 0 \texttt{ then } 1 \texttt{ else } n * fact\ (n-1) \texttt{ in } fact\ 2\ ;;$$

$\triangleright\ [fact \mapsto ([fact \mapsto (.,.)]\ ,\ \lambda n\ .\ \texttt{if } n = 0 \texttt{ then } 1 \texttt{ else } n * fact\ (n-1))]$	$fact\ 2$
$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2]$	$\texttt{if } n = 0 \texttt{ then } 1 \texttt{ else } n * fact\ (n-1)$
$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2]$	$n * fact\ (n-1)$
$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2]$	$2 * fact\ (2-1)$
$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2]$	$2 * fact\ 1$
$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2,\ n \mapsto 1]$	$2 * (\texttt{if } n = 0 \texttt{ then } 1 \texttt{ else } n * fact\ (n-1))$
$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2,\ n \mapsto 1]$	$2 * (n * fact\ (n-1))$
$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2,\ n \mapsto 1]$	$2 * (1 * fact\ 0)$
$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2,\ n \mapsto 1,\ n \mapsto 0]$	$2 * (1 * (\texttt{if } n = 0 \texttt{ then } 1 \texttt{ else } \ldots))$
$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2,\ n \mapsto 1,\ n \mapsto 0]$	$2 * (1 * 1\ )$
$\triangleright\ [fact \mapsto (.,.),\ n \mapsto 2,\ n \mapsto 1,\ n \mapsto 0]$	2

Liste récursive

let rec *x* = *expr1* `in` *expr2* :
évaluation de *expr2*
dans l'environnement augmenté de $x \mapsto$ *expr1*

Comment évaluer :
let rec x = 0 :: x `in` x ;;

Liste récursive

let rec *x* = *expr1* `in` *expr2* :
évaluation de *expr2*
dans l'environnement augmenté de $x \mapsto$ *expr1*

Comment évaluer :

```
# let rec x = 0 :: x in x;;
```

$[x \mapsto 0 :: x]$ 0 :: x

Cela donne une liste cyclique

Que rend OCaml ?

```
# let rec x = 0 :: x;;
```

Cela donne une liste cyclique

Que rend OCaml ?

```
# let rec x = 0 :: x;;
val x : int list = [0; <cycle>]
```

Cela donne une liste cyclique

Que rend OCaml ?

```
# let rec x = 0 :: x;;
val x : int list = [0; <cycle>]
```

Moralement : `0 :: 0 :: 0 :: 0 :: 0 :: 0 :: 0 :: ...`

Cela donne une liste cyclique

Que rend OCaml ?

```
# let rec x = 0 :: x;;
val x : int list = [0; <cycle>]
```

Moralement : `0 :: 0 :: 0 :: 0 :: 0 :: 0 :: 0 :: ...`

```
# let rec n_ieme n u =
      match u with
      | [] -> failwith "derriere la fin de liste"
      | x :: v -> if n = 0 then x else n_ieme (n - 1) v;;
# n_ieme 3 x;;
```

Cela donne une liste cyclique

Que rend OCaml ?

```
# let rec x = 0 :: x;;
val x : int list = [0; <cycle>]
```

Moralement : `0 :: 0 :: 0 :: 0 :: 0 :: 0 :: 0 :: ...`

```
# let rec n_ieme n u =
      match u with
      | [] -> failwith "derriere la fin de liste"
      | x :: v -> if n = 0 then x else n_ieme (n - 1) v;;
# n_ieme 3 x;;
- : int = 0
```

Mais incompatible avec la récursivité et la récurrence

```
# let rec long u =  match u with
                    | [] -> 0
                    | x :: v -> 1 + long v;;
# long x;;
```

Mais incompatible avec la récursivité et la récurrence

```
# let rec long u =  match u with
                    | [] -> 0
                    | x :: v -> 1 + long v;;
# long x;;
Stack overflow during evaluation (looping recursion?)
```

Mais incompatible avec la récursivité et la récurrence

```
# let rec long u =  match u with
                    | [] -> 0
                    | x :: v -> 1 + long v;;
# long x;;
Stack overflow during evaluation (looping recursion?)
```

Un programme récursif structurel sur une liste doit être invoqué sur une liste sans cycle.

De même, un raisonnement par récurrence structurelle sur une liste n'a de sens que pour les listes sans cycle.

Induction et co-induction (*)

Les types somme récursifs se divisent en deux espèces mathématiques distinctes :

- les types inductifs :
  pas de cycles, pas d'utilisation infinie de constructeurs (on finit toujours par un cas de base) ;
- les types coinductifs :
  tous les coups sont permis à la construction.

On peut toujours précéder par cas, mais récursion et récurrence structurelle n'ont de sens que pour les objets inductifs.

Coq distingue ces deux espèces mais pas OCaml.

Une structure de donnée cyclique utile (*)

L'environnement d'une fonction récursive introduit un cycle

- **let rec** $x = expr1$ `in` $expr2$: évaluation de $expr2$ dans l'environnement augmenté de $x \mapsto expr1$

```
let rec fact = fun n -> if n = 0 then 1 else n * fact (n - 1)
        in fact 2 ;;
```

Une structure de donnée cyclique utile (*)

L'environnement d'une fonction récursive introduit un cycle

- **let rec** $x = expr1$ `in` $expr2$: évaluation de $expr2$ dans l'environnement augmenté de $x \mapsto expr1$

```
let rec fact = fun n -> if n = 0 then 1 else n * fact (n - 1)
        in fact 2 ;;
```

▷

$$[\underbrace{fact \mapsto ([fact \mapsto \ldots]}_{\textbf{cycle}}, \lambda\ n.\ \texttt{if}\ n = 0\ \texttt{then}\ 1\ \texttt{else}\ \ldots)] \quad fact\ \ 2$$