

# Typage et Analyse Statique

## Cours 1

Emmanuel Chailloux

Spécialité Science et Technologie du Logiciel  
Master mention Informatique  
Sorbonne Université

année 2021-2022

# Plan du cours 1

- ▶ termes, substitution,  $\alpha$ -conversion,  $\beta$ -réduction
- ▶ codage de données en  $\lambda$ -calcul
  - ▶ booléens,
  - ▶ couples
  - ▶ entiers (Church, Barendregt)
- ▶ codage de contrôle : la récursion
- ▶ forme normale
- ▶ stratégies de réduction

Le  $\lambda$ -calcul a été inventé par Alonzo Church en 1932.

Le but de Church était de définir la notion de calculabilité effective au moyen de la  $\lambda$ -définissabilité.

Cette notion est équivalente aux notions de calculabilité au sens de Turing (machine de Turing) et de Gödel-Herbrand (fonctions récursives).

Cette coïncidence incite à penser qu'il existe une notion de calculabilité universelle, indépendante des formalismes particuliers : c'est la thèse de Church.

Le but de ce cours est de présenter brièvement le  $\lambda$ -calcul, qui sert de base théorique à tout langage fonctionnel bien conçu. Plus qu'un cours formel, il s'agit d'une introduction qu'on espère motivante et qui pourra inciter le lecteur intéressé à consulter la littérature pour plus de détails (voir bibliographie). C'est pourquoi on n'y trouvera guère de démonstrations.

## **bibliographie**

- ▶ Hindley and Seldin. Introduction to Lambda-Calculus and combinators Cambridge University Press, 1986.
- ▶ Chantal Berline. une introduction au lambda-calcul, cours au dea de logique 2001-2003 (cf <https://www.irif.fr/~berline/Cours.html>).

## Termes (1)

On se donne un ensemble  $V$  infini dénombrable de variables. On définit les termes inductivement comme suit :

- ▶ Si  $x \in V$  alors  $x$  est un terme
- ▶ Si  $x \in V$  et  $M$  est un terme, alors  $\lambda x.M$  est un terme ; c'est la fonction qui, à  $x$ , associe  $M$  (qui, en général, dépend de  $x$ ). On dit que la variable  $x$  est abstraite dans  $M$ .
- ▶ Si  $M$  et  $N$  sont des termes, alors  $MN$  est un terme. C'est l'application de  $M$  (à considérer comme une fonction) à  $N$  (à considérer comme son argument).

## Termes (2)

Tout de suite quelques exemples :

- ▶ L'identité :  $\lambda x.x$ , ou bien  $\lambda y.y$  ou bien...
- ▶ On peut l'appliquer à elle-même :  $(\lambda x.x)(\lambda x.x)$ .
- ▶  $K = \lambda x.\lambda y.x$

Pour alléger les notations, on écrira souvent

- ▶  $\lambda x_1 x_2 \dots x_n.M$  au lieu de  $\lambda x_1.\lambda x_2.\dots.\lambda x_n.M$ ,
- ▶ et  $M_1 M_2 \dots M_k$  au lieu de  $(\dots((M_1 M_2) M_3) \dots) M_k$ .

On remarquera que ce sont les conventions habituelles d'OCaml.

## Substitution

La substitution aux occurrences libres (non liées à un  $\lambda$ ) d'une variable  $x$  d'un terme  $M$  par un terme  $N$  sera notée  $M[N/x]$  (où  $N$  écrase  $x$  dans  $M$ ).

### Définition

Pour tout terme  $N, M$  et pour n'importe quelle variable  $x$ , le résultat ( $M[N/x]$ ) de la substitution de toutes les occurrences libres de  $x$  dans  $M$  par  $N$  est définie de la manière suivante :

1.  $x[N/x] \equiv N$
2.  $y[N/x] \equiv y$  pour tout  $y \neq x$
3.  $(M_1 M_2)[N/x] \equiv (M_1[N/x])(M_2[N/x])$
4.  $(\lambda x. Y)[N/x] \equiv \lambda x. Y$
5.  $(\lambda y. Y)[N/x] \equiv (\lambda y. Y[N/x])$  si  $y \neq x$ , et  $y \notin N$  ou  $x \notin Y$
6.  $(\lambda y. Y)[N/x] \equiv (\lambda z. Y[z/y][N/x])$  si  $y \neq x$  et  $y \in N$  et  $x \in Y$  où  $z$  est une nouvelle variable.

## $\alpha$ -conversion

On remarque dans ces exemples que le nom des variables liées dans un terme n'a aucune importance. Cela se traduit par la règle d' $\alpha$ -conversion :

$$\lambda x.M \equiv \lambda y.M[y/x]$$

si  $x$  n'est pas liée dans  $M$  et  $y$  ne figure pas libre ou liée dans  $M$ . L'importance de cette règle apparaîtra avec la  $\beta$ -conversion.

Exemples :

- ▶  $\lambda x.xy \equiv \lambda z.zy$  mais  $\lambda x.xy \not\equiv \lambda y.yy$ .
- ▶  $\lambda x.yx \equiv \lambda z.yz$  mais  $\lambda x.yx \not\equiv \lambda y.yy$

## $\beta$ -réduction (1)

C'est la règle fondamentale du  $\lambda$ -calcul. Un terme de la forme  $(\lambda x.M)N$  est appelé *redex*. Voici la réduction d'un redex :

$$(\lambda x.M)N \rightarrow_0 M[N/x]$$

à condition que  $x$  n'apparaisse pas libre dans  $N$  (si c'est le cas, faire une  $\alpha$  conversion sur  $\lambda x.M$ ) et qu'aucune variable libre de  $N$  ne soit capturée dans  $M$  (là aussi faire une  $\alpha$  conversion sur  $M$ ). Le terme de droite est appelé *réduit*.



## $\beta$ -réduction (2)

Ce qu'on appellera  $\beta$ -réduction, c'est la réduction d'un redex à l'intérieur d'un terme. Plus précisément, c'est la relation de réduction  $\rightarrow$  définie par :

- ▶ Si  $M \rightarrow_0 M'$  alors  $M \rightarrow M'$ .
- ▶ Si  $M \rightarrow M'$  alors  $\lambda x.M \rightarrow \lambda x.M'$ .
- ▶ Si  $M \rightarrow M'$  alors  $MN \rightarrow M'N$ .
- ▶ Si  $N \rightarrow N'$  alors  $MN \rightarrow MN'$ .

C'est grâce à cette règle que le  $\lambda$ -calcul est véritablement un calcul.

## Booléens (1)

En  $\lambda$ -calcul pur nous pouvons représenter les booléens par des  $\lambda$ -termes. Une représentation possible est la suivante :

- ▶  $T = \lambda xy.x$
- ▶  $F = \lambda xy.y$

**opérateur conditionnel** : fonction qui prend comme arguments une condition  $c$  et deux expressions  $e_1$  et  $e_2$ . Cette fonction doit retourner  $e_1$ , si  $c$  évalue à  $T$ , et  $e_2$  si  $c$  évalue à  $F$ , ce qui conduit à la définition suivante :

- ▶ **cond** =  $\lambda ce_1e_2.((c e_1) e_2)$

## Booléens (2)

Essayons. Soient  $E1$  et  $E2$  deux  $\lambda$ -termes quelconques.

$$\begin{aligned} \text{cond } T \ E1 \ E2 &= (\lambda c e_1 e_2. ((c \ e_1) \ e_2)) \ T \ E1 \ E2 \\ &\rightarrow ((T \ E1) \ E2) \\ &= (((\lambda xy. x) \ E1) \ E2) \\ &\rightarrow (\lambda y. E1) \ E2 \\ &\rightarrow E1 \end{aligned}$$

Naturellement on peut, pour se faciliter la vie, se définir des abbréviations comme la suivante :

► **if c then e1 else e2 = cond c e1 e2**

## Booléens (3)

Comment définir les connecteurs logiques à partir de *cond* ?

- ▶ **not e1** = *if e1 then F else T*  
*cond e1 F T*
- ▶ **or e1 e2** = *if e1 then T else e2*  
*cond e1 T e2*
- ▶ **and e1 e2** = *if e1 then e2 else F*  
*cond e1 e2 F*

## Couples (1)

On a besoin d'un constructeur, appelé ici **cons**, qui prend deux éléments et retourne un couple et de deux accesseurs **fst** et **snd** qui prenant un couple  $(a, b)$  retourne respectivement  $a$  et  $b$ , c'est-à-dire vérifient les équations suivantes :

$$fst (cons a b) = a$$

$$snd (cons a b) = b$$

Pour la fonction **cons** nous pouvons donner la définition :

▶ **cons** =  $\lambda xyf.f \times y$

Appliquée à A et B elle donne :

$$\begin{aligned} cons A B &= (\lambda xyf.f \times y) A B \\ &\rightarrow \lambda f.f A B \end{aligned}$$

## Couples (2)

Or le terme  $\lambda f. f A B$  est capable de capturer des termes qui remplaceront  $f$ . En particulier nous pouvons remplacer  $f$  par des fonctions `first` et `second` capables de sélectionner le premier ou le deuxième parmi les arguments auxquelles elles sont appliquées :

- ▶ **first** =  $\lambda xy.x$

- ▶ **second** =  $\lambda xy.y$

`fst` et `snd` seront ainsi définies :

- ▶ **fst** =  $\lambda x.(x \text{ first})$

- ▶ **snd** =  $\lambda x.(x \text{ second})$

## Entiers de Church (1)

Un nombre est représenté comme un terme qui représente  $n$  applications successives d'une fonction à un argument (si le nombre en question est  $n$ ). Le nombre  $n$  sera représenté par  $\lambda f x. f^n x$

▶  $\bar{0} = \lambda f x. x$

▶  $\bar{1} = \lambda f x. f x$

▶  $\bar{2} = \lambda f x. f(f x)$

▶  $\bar{n} = \lambda f x. f^n x$

ce qui équivaut à  $\lambda f x. (f(f \dots (f x) \dots))$

## Entiers de Church (2)

La fonction successeur,  $\sigma$ , doit prendre un numéral  $n$  de la forme  $\lambda fx.f^n x$  et rendre  $\lambda fx.f(f^n x)$ . On peut « ouvrir » le numéral  $\bar{n}$  en l'appliquant à deux arguments quelconques, par exemple à  $f$  et à  $x$  :

$$(\lambda fx.f^n x) f x \rightarrow f^n x$$

Pour obtenir le successeur de  $\bar{n}$  il faut donc « capturer »  $n$ , l'ouvrir, lui ajouter un  $f$  en tête, et « laisser », à la tête de tout ça un nouveau  $\lambda fx$ .

Ceci est fait par la fonction :

▶  $\sigma = \lambda nfx.f(n f x)$



## Entiers de Church (3)

Nous pouvons à l'aide de  $\sigma$  définir des simples fonctions comme la somme

▶  $\text{add} = \lambda mn.m\sigma n$

.

et pour :

▶  $\text{mul} = \lambda mn.$

▶  $\text{exp} = \lambda mn.$

et prédécesseur ???

## Entiers de Barendregt (1)

Cette représentation des entiers, qui vient de Barendregt, nous permettra d'utiliser le calcul sur les booléens que nous avons déjà défini.

Soient :

$$\blacktriangleright \bar{0} = \lambda x.x$$

$$\blacktriangleright \sigma = \lambda n.\lambda f. ((f \ F) \ n)$$

Si nous appliquons un numéral  $\bar{n}$  ( $\bar{n} = \sigma^n \bar{0}$ ) à  $T$  (i.e.  $\lambda xy.x$ ) nous obtenons de façon triviale  $T$  ou  $F$ , selon que  $\bar{n}$  est  $\bar{0}$  ou un autre numéral :

$$\begin{aligned}\bar{0} \ T &= (\lambda x.x) (\lambda xy.x) \\ &\rightarrow \lambda xy.x \\ &= T\end{aligned}$$

## Entiers de Barendregt (2)

Par contre

$$\begin{aligned}\bar{n} T &= (\lambda f.((f F) \overline{n-1}) (\lambda xy.x)) \\ &\rightarrow ((\lambda xy.x F) \overline{n-1}) \\ &\rightarrow (\lambda y.F \overline{n-1}) \\ &= (\lambda y.(\lambda ab.b) \overline{n-1}) \\ &\rightarrow (\lambda y.\lambda b.b) \\ &= F\end{aligned}$$

Définissons en outre :

- ▶ **iszero** =  $\lambda n.(n \text{ first})$
- ▶ **pred1** =  $\lambda n.(n \text{ second})$
- ▶ **pred** =  $\lambda n.(\text{if } (\text{iszero } n) \text{ then } \bar{0} \text{ else } (\text{pred1 } n))$

## Récursion (1)

En essayant de définir l'addition, on serait tenté de la définir de la façon suivante :

▶  $\text{add} = \lambda mn. \text{if } (\text{iszero } n) \text{ then } m \text{ else } \sigma(\text{add } m \text{ (pred } n))$

Or ceci ne marche pas, car « add » est purement une abbréviation, si bien qu'il faudrait remplacer son occurrence dans le corps de la définition par la définition elle même, qui contient une occurrence de « add » qui doit être remplacée . . .et ainsi de suite.

La solution est donnée par l'opérateur de *point fixe*  $Y$ , qui a la propriété suivante : pour toute fonction  $F$ ,

$$(Y F) \cong F(Y F)$$

où  $\cong$  est l'équivalence engendrée par  $\rightarrow$ , appelée  $\beta$ -conversion. Plusieurs opérateurs, munis de cette propriété, ont été définis.

## Récursion (2)

- ▶  $Y = (\lambda f.(\lambda s.f (s s)) (\lambda s.f (s s)))$

La somme sera alors définie en faisant d'abord une abstraction sur le symbole *add* dans la définition précédente, puis en appliquant *Y* :

- ▶  $add1 = \lambda fmn.if (iszero n) then m else succ(f m (pred n))$

- ▶  $add = Y add1$

## Récursion (3)

Exemple :

$$\begin{aligned} \text{add } 9 \ 1 &\cong (Y \ \text{add1}) \ 9 \ 1 \\ &= (\lambda f.(\lambda s.f \ (s \ s)) \ (\lambda s.f \ (s \ s))) \ \text{add1} \ 9 \ 1 \\ &\rightarrow (\lambda s.\text{add1} \ (s \ s)) \ (\lambda s.\text{add1} \ (s \ s)) \ 9 \ 1 \\ &\rightarrow \text{add1} \ ((\lambda s.\text{add1} \ (s \ s))(\lambda s.\text{add1} \ (s \ s))) \ 9 \ 1 \\ &= \text{add1} \ (Y \ \text{add1}) \ 9 \ 1 \end{aligned}$$

...

$$\begin{aligned} &= (\lambda fmn.\text{if} \ (\text{iszero} \ n) \ \text{then} \ m \ \text{else} \ \text{succ}(f \ m \ (\text{pred} \ n))) \ (Y \ \text{add1}) \ 9 \ 1 \\ &\rightarrow \text{if}(\text{iszero} \ 1) \ \text{then} \ 9 \ \text{else} \ \text{succ} \ ((Y \ \text{add1}) \ 9 \ (\text{pred} \ 1)) \\ &\rightarrow \text{if}(\text{iszero} \ 1) \ \text{then} \ 9 \ \text{else} \ \text{succ} \ ((Y \ \text{add1}) \ 9 \ 0) \\ &\rightarrow \text{succ} \ (\text{add1} \ (Y \ \text{add1}) \ 9 \ 0) \\ &\rightarrow \text{succ} \ (\text{if} \ (\text{iszero} \ 0) \ \text{then} \ 9 \ \text{else} \ \text{succ}((Y \ \text{add1}) \ 9 \ (\text{pred} \ 0))) \\ &\rightarrow \text{succ} \ 9 \\ &\rightarrow 10 \end{aligned}$$

## Définitions et théorèmes (1)

On note  $\rightarrow^*$  la fermeture réflexive-transitive de la relation de conversion  $\rightarrow$  définie précédemment. (Donc  $M \rightarrow^* N$  si et seulement si il existe une suite finie  $(M_1, \dots, M_n)$  avec  $n \geq 1$  telle que  $M = M_1$ ,  $N = M_n$  et  $M_1 \rightarrow M_2 \cdots M_n$ .)

### Théorème

*(Church-Rosser) Soit  $M$  un  $\lambda$ -terme. Soient  $M_1$  et  $M_2$  des termes tels que  $M \rightarrow^* M_1$  et  $M \rightarrow^* M_2$ . Il existe un terme  $N$  tel que  $M_1 \rightarrow^* N$  et  $M_2 \rightarrow^* N$ .*

Ce théorème a une conséquence intéressante.

## Définitions et théorèmes (2)

### Définition

- ▶ On dit qu'un terme est en *forme normale* s'il ne possède aucun redex.
- ▶ On dit qu'un terme a une forme normale, ou qu'il est normalisable, s'il existe une réduction de ce terme (pour  $\rightarrow^*$ ) qui mène à une forme normale.
- ▶ On dit qu'il est fortement normalisable si toute réduction pour  $\rightarrow^*$  mène à une forme normale.

Alors, bien sûr

### Théorème

*Si un terme est normalisable, il a une unique forme normale.*

C'est une conséquence immédiate du théorème de Church-Rosser.  
(Pourquoi?)



## Définitions et théorèmes (3)

Remarquons qu'il existe des termes non normalisables : soit  $\Omega = (\lambda x.xx)(\lambda x.xx)$ . On a

$$\Omega \rightarrow_0 \Omega \rightarrow_0 \Omega \rightarrow_0 \dots$$

Il existe aussi des termes normalisables qui ne sont pas fortement normalisables.

C'est le cas le  $(\lambda xy.y)\Omega$ . Si l'on choisit de réduire d'abord le redex qui se trouve à l'intérieur de  $\Omega$ , et de continuer ensuite dans cette voie, on est perdu. Sinon, c'est-à-dire si l'on réduit le « redex de tête » (qui est le terme lui-même), on a gagné, car  $\Omega$  disparaît. Cet exemple laisse entrevoir que le choix de la stratégie de réduction est crucial en  $\lambda$ -calcul.

## réduction standard (1)

On remarque facilement que tout  $\lambda$ -terme  $M$  est de la forme

$$\lambda x_1 x_2 \cdots x_n. M_1 M_2 \cdots M_k$$

où  $M_1$  est soit une variable, soit une abstraction.

### Définition

Si  $M_1$  est une variable  $x$ , cette variable est appelée variable de tête de  $M$ . On dit alors que  $M$  est en forme normale de tête.

Dans le cas contraire, on a  $M_1 = \lambda y. N_1$ , et le redex  $(\lambda y. N_1) M_2$  est appelé « redex de tête » de  $M$ . La stratégie standard consiste

- ▶ dans le cas où il y a un redex de tête, à le réduire,
- ▶ et si, par contre,  $M$  est en forme normale de tête, à appliquer la réduction standard à chacun des  $M_2, \dots, M_k$  (ce sont des réductions indépendantes; on pourrait les mener en parallèle).

Voici (sans démonstration) le résultat annoncé :

### Théorème

*Appliquée à un terme normalisable, la réduction standard termine.*

## réduction standard (2)

Deux situations sont possibles, quand on applique la réduction standard à un terme :

- ▶ on aboutit à une forme normale de tête. Le terme est alors dit solvable.
- ▶ Cela ne termine pas, et il n'y a jamais de forme normale de tête. Terme non solvable. C'est le cas de  $\Omega$ .

Un terme solvable n'est pas forcément normalisable ; il se peut que la réduction standard produise une suite infinie de formes normales de tête. On peut alors voir chaque variable de tête successive comme un « bout d'information » sur une donnée infinie qu'on n'atteindra jamais dans sa totalité. Exemple : soit  $A = \lambda xy.y(xx)$  alors le terme  $AA$  est de cette sorte, puisqu'en effet :

$$AA \rightarrow \lambda y.y(AA) \rightarrow \lambda y.y(\lambda y.y(AA)) \rightarrow \dots$$

et on produit ainsi la liste des variable de tête  $y, y \dots$

## réduction standard (3)

Pour finir, voici un exemple qui montre que la réduction standard n'est pas optimale en général. Soit  $I = \lambda x.x$  l'identité. On considère  $B = (\lambda y.yy)(II)$ .

Si on applique la stratégie standard, on obtient :

$$B \rightarrow II(II) \rightarrow I(II) \rightarrow II \rightarrow I$$

mais on aurait pu réduire d'abord à droite (le redex  $II$ ) dans  $B$ , ce qui aurait donné par exemple :

$$B \rightarrow (\lambda y.yy)I \rightarrow II \rightarrow I$$

soit trois étapes au lieu de quatre.