Cours Programmation Fonctionnelle - Programmation Logique

Pr. Mouhamadou Thiam

April 26, 2017

Domaine du XXIe siecle

- Machine Learning.
- Semantic Web.
- Annotation.

Parties du cours

- Lamda Calcul.
- Programmation Fonctionnelle : *Haskell*.
- Logique du 1^{er} ordre.
- Programmation Logique : *Prolog*.

Abstractions, réductions

Le λ-calcul fournit d'abord :

- une notation pour transformer une expression
- par exemple $2x + 1$.
- en une fonction : $F = \lambda x.2x + 1$.
- traditionnellement notée : $x \longrightarrow 2x + 1$
- cette construction est appelée *abstraction*
- Cette λ-expression comporte deux parties, séparées par un point:
  1. à gauche une variable, dite liée
  2. à droite le corps de F

Abstractions, réductions

- Le nom de la variable liée est sans importance
- l'expression $\lambda t.2t + 1$ est équivalente
- en $\lambda - jargon$, le changement de nom d'une variable liée $\equiv \alpha - conversion$.
- Pour appliquer F à un argument (par exemple $a^2 + b^2$)
- substitue, dans le corps de F, l'argument à la variable liée
- on obtient : $2(a^2 + b^2) + 1$;
- cette opération de substitution est appelée $\beta - réduction$

Abstractions, réductions

- On note, par simple juxtaposition, $F\ M$ l'application de F à l'argument M
- cette convention diffère en deux points de la convention mathématique usuelle :
- on ne place pas l'argument entre parenthèses
- les lettres minuscules sont réservées aux variables, une majuscule désigne donc une expression quelconque
- cette convention n'est pas universelle (autrement dit elle n'est pas utilisée dans tous les traités de $\lambda - calcul$)
- Les parenthèses sont utilisées comme d'habitude pour lever les ambiguïtés

Abstractions, réductions

- Une fonction de 2 variables est notée : $G = \lambda x.(\lambda y.x^2 + y^2)$
- Cette représentation d'une fonction de plusieurs variables par des fonctions d'une variable emboîtées est la *curryfication*
- du nom du mathématicien *Haskell Brooks Curry*. On abrège l'expression ci-dessus en :
- $G = \lambda xy.x^2 + y^2$
- on note par simple juxtaposition l'application de G à deux arguments M et N
- par convention : $G\ M\ N$ est une abréviation pour :$(G\ M\ )N$

par allègement des parenthèses : le corps d'une

λ-calcul pur

- Notations du $\lambda - calcul$, nous avons utilisé des expressions arithmétiques, comme $2x + 1$ ou $x^2 + y^2$
- Le $\lambda - calcul$ pur ne suppose définis ni les entiers, ni a fortiori les opérations arithmétiques
- une $\lambda - expression$ est construite récursivement à partir de :
  1. variables;
  2. opérations d'abstraction
  3. opérations d'application
- Autrement dit :
- une variable x est une $\lambda - expression$;
- **abstraction** : si x est une variable, et M une $\lambda - expression$, $\lambda x\ .\ M$ est une $\lambda - expression$;

λ-calcul pur

- Pas d'autres règles
- Sauf les parenthèses pour lever les ambiguités.
- nous appellerons *terme* une $\lambda - expressions$ quelconque
- *variable* est un *terme*, un *terme* n'est pas, en général, une simple *variable*
- Un *terme* est représenté par son arbre
- feuilles étiquetées par des variables
- **abstraction** : nœuds internes sont étiquetés par λ
- **application** : nœuds non étiquetés

λ-calcul pur

- En $\lambda - \textit{calcul terme}$
  $\equiv \lambda - \textit{terme} \equiv \textit{expression} \equiv \lambda - \textit{expression}$
- Minuscules $\longrightarrow$ *variables* (f, g)
- Majuscules $\longrightarrow$ *termes* (F, G)
- λx y . M $\equiv \lambda$ x . (λ y . M)
- λx . M N $\equiv \lambda$ x . (M N) et non (λ x . M) N
- $F\ U\ V \equiv (F\ U)\ V$ et non $F\ (U\ V)$
- Un terme est un programme, qu'on "exécute" par (β-)réductions successives
- (λ x . M) N $\longrightarrow$ M[N/x]

λ-calcul pur

- Substitution dans le terme M de x par N
- Une transformation de termes
- Le terme gauche est un *redex*
- Changement de variables liés peut être nécessaire
- Ce modèle de calcul, par récritures $\equiv$ à celui des machines de Turing
- Le λ-calcul est à la base de nombreux langages de programmation, dits fonctionnels, tels que *Lisp* (*Scheme, ML*)
- (λ-calcul pur, ces langages) $\equiv$ (machine de Turing, langage C)

entiers de Church

- *entiers de Church* $\equiv$ *opérateurs* sur les fonctions
- **3** $\equiv$ l'opérateur $\lambda\ f\ .\ f^3$
- $f^3 \equiv \lambda x\ .\ f\ (f\ (f\ x\ )\ )$
- **0** $\equiv \lambda\ \ f\ x\ .\ x$ ($f^0 =$ identité)
- **1** $\equiv \lambda\ \ f\ x\ .\ f\ x$
- **2** $\equiv \lambda\ \ f\ x\ .\ f\ (f\ x)$
- **3** $\equiv \lambda\ \ f\ x\ .\ f\ (f\ (f\ x)\ )$
- etc.

entiers de Church

- Tout entier de Church **n** est donc un opérateur qui transforme f en f^n
- $f^n = f \circ f \circ \ldots \circ f \equiv \mathbf{n}\ f$.
- $\mathbf{succ} = \lambda\ n\ .\ (\lambda\ f\ .\ f^{n+1})$
- $= \lambda\ n\ f\ x\ .\ f^n\ (\ f\ x\ ) = \lambda\ n\ f\ x\ .\ n\ f\ (\ f\ x\ ) = \lambda\ n\ f\ x\ .\ f\ (\ n\ f\ x\ )$
- $\mathbf{add} = \lambda\ m\ n\ .\ (\lambda\ f\ .\ f^{m+n}) = \lambda\ m\ n\ f\ x\ .\ f^m\ (f^n\ x\ ) = \lambda\ m\ n\ f\ x\ .\ m\ f\ (\ n\ f\ x\ )$
- $\mathbf{mul} = \lambda\ m\ n\ .\ (\lambda\ f\ .\ (\ f^m)^n) = \lambda\ m\ n\ f\ .\ n\ (\ m\ f\ )$
- $\mathbf{exp} = \lambda\ m\ n\ .\ n\ m$ calcule m^n

entiers de Church

- **mul** commutative $\longrightarrow$ ordre de m et n sans importance
- par contre il est essentiel pour le calcul de m^n
- Exemple : $n\ f\ (\ f\ x\ )$ interprété comme $(\ n\ f\ )\ (\ f\ x\ )$
- sans les parenthèses $\rightarrow$ faute car $n\ f\ f\ x\ equiv\ (\ n\ f\ f\ )\ x$
- le λ-calcul pur n'est pas typé
- rien ne dit que m et n sont entiers, ce sont de simples variables
- opérateurs appliqués à des termes $\neg$ *entiers de Church*, le résultat est *"imprévisible"*. On ne peut rien dire d'intéressant sur le cas général.

entiers de Church

1. Exemple 1 : Réduire **succ 2** en **3**
2. $(\lambda\ n\ f\ x\ .\ n\ f\ (\ f\ x\ )\ )\ \mathbf{2} \rightarrow$
3. $\lambda\ f\ x\ .\ \mathbf{2}\ f\ (\ f\ x\ ) \rightarrow$
4. $\lambda\ f\ x\ .\ (\lambda\ g\ t\ .\ g\ (\ g\ t\ )\ )\ f\ (\ f\ x\ ) \rightarrow$
5. $\lambda\ f\ x\ .\ f\ (\ f\ (\ f\ x\ )\ )\ ) = \mathbf{3}$
6. De *l2* à *l3* : substitution n à **2**
7. De *l3* à *l4* : réduction de (**2** f), on utilise t pour éviter les confusions.
8. De *l4* à *l5* : substitution de $(\ f\ x\ )$ à t

entiers de Church

- Exemple 2 : Addition de **2** et **3**
- $(\lambda\ m\ n\ f\ x\ .\ m\ f\ (\ n\ f\ x\ )\ )\ \mathbf{2}\ \mathbf{3} \rightarrow$
- $\lambda\ \ f\ x\ .\ 2\ f\ (\ 3\ f\ x\ ) \rightarrow$
- $\lambda\ \ f\ x\ .\ 2\ f\ (\ f\ (\ f\ (\ f\ x\ )\ )\ ) \rightarrow$
- $\lambda\ \ f\ x\ .\ f\ (\ f\ (\ f\ (\ f\ (\ f\ x\ )\ )\ )\ ) = \mathbf{5}$
- De *I2* à *I3* : 2 substitutions en parallèle
- Une pour chacun des arguments.

entiers de Church

- Exemple 3 : Multiplication de **2** et **3**
- $(\lambda\ m\ n\ f\ .\ n\ (\ m\ f\ )\ )\ \mathbf{2}\ \mathbf{3} \rightarrow$
- $\lambda\ f\ .\ \mathbf{3}\ (\ \mathbf{2}\ f\ ) \rightarrow$
- $\lambda\ f\ .\ \mathbf{3}\ (\ \lambda\ t\ .\ f\ (\ f\ t\ )\ ) \rightarrow$
- Soit $M = \lambda\ \mathrm{t}\ .\ \mathrm{f}\ (\ \mathrm{f}\ \mathrm{t}\ )$
- $\mathbf{3}\ M \rightarrow \lambda\ x\ .\ M\ (\ M\ (\ M\ x\ )\ )$
- On a 3 réductions possibles correspondants aux 3 occurrences de M
- Choisissons la plus interne.

entiers de Church

- $\lambda\, f\, x\, .\; M\, (\, M\, (\, M\, x\,)\,) \rightarrow$
- $\lambda\, f\, x\, .\; M\, (\, M\, (\, f\, (\, f\, x\,)\,)\,) \rightarrow$
- $\lambda\, f\, x\, .\; M\, (\, M\, (\, f^2\, x\,)\,) \rightarrow$
- $\lambda\, f\, x\, .\; M\, (\, f\, (\, f\, (\, f^2\, x\,)\,) \rightarrow$
- $\lambda\, f\, x\, .\; M\, (\, (\, f^4\, x\,)\,) \rightarrow$
- $\lambda\, f\, x\, .\; M\, (\, f\, (\, f\, (\, f^4\, x\,)\,) \rightarrow$
- $\lambda\, f\, x\, .\; f\, (\, f\, (\, f^4\, x\,)\,) \rightarrow$
- $\lambda\, f\, x\, .\; (\, f^6\, x\,) \rightarrow \mathbf{6}$

entiers de Church

- On a choisi une stratégie interne de réduction, y compris lorsqu'on a réduit **3** *(* **2** *f)*
- On pourra vérifier que le résultat ne dépend pas de la stratégie choisie
- C'est le théorème de **Church-Rosse**

entiers de Church

- Exemple 4 : **exp** de **2** et **3**
- $\lambda\ m\ n\ .\ n\ m\ \mathbf{2}\ \mathbf{3} \rightarrow \mathbf{3}\ \mathbf{2} \rightarrow \lambda\ x\ .\ \mathbf{2}\ (\ \mathbf{2}\ (\ \mathbf{2}\ x\ )\ )$
- AY dans la jungle !!! I konw **2** f and not **2** x
- Ouf : nom de variables sans importance, λ-calcul non typé
- aucune variables ne désigne plutôt une fonction ou plutôt un objet
- changeons x en f : α-conversion
- $\mathbf{3}\ \mathbf{2} \rightarrow \lambda\ f\ .\ \mathbf{2}\ (\ \mathbf{2}\ (\ \mathbf{2}\ f\ )\ )$
- $\mathbf{3}\ \mathbf{2} \rightarrow \lambda\ f\ .\ \mathbf{2}\ (\ \mathbf{2}\ (\ \lambda\ t\ .\ f\ (\ f\ t)\ )\ )$

entiers de Church

- Posons $M = \lambda\ t\ .\ f\ (\ f\ t)$
- $\mathbf{3}\ \mathbf{2} \rightarrow \lambda\ f\ .\ \mathbf{2}\ (\ \mathbf{2}\ (\ M\ )\ )$
- $\rightarrow \lambda\ f\ .\ \mathbf{2}\ (\ \lambda\mu\ M\ (\ M\ \mu)\ )$
- $\rightarrow \lambda\ f\ .\ \mathbf{2}\ (\ \lambda\mu\ M\ (\ M\ \mu)\ )$
- Posons $N = \lambda\ \mu\ M\ (\ M\ \mu)$
- $\rightarrow \lambda\ f\ .\ \mathbf{2}\ N$
- $\rightarrow \lambda\ f\ x\ .\ N\ (\ N\ x\ )$
- $\rightarrow \lambda\ f\ x\ .\ N\ (\ M\ (\ M\ x\ )\ )$

entiers de Church

- $\rightarrow \lambda\ f\, x\ .\ M\ (\ M\ (\ M\ (\ M\, x\ )\ )\ )$
- Utiliser **4** fois la définition de M pour avoir **8**
- Méditer sur la puissance du λ-calcul
- Il permet de définir aussi simplement l'opérateur d'exponentiation,
- appliquer un entier à un entier est possible
- puisqu'un entier a été défini comme un opérateur !

Récursion et points fixes

- Le λ-calcul définir très simplement **false**, **true** et **if**.
- Le test **if** est une fonction de trois variables telle que
  1. **if true** $M\ N \longrightarrow M$
  2. **if false** $M\ N \longrightarrow N$
- Comme les entiers booléens **opérateurs** à 2 arguments
- **true** sélectionne son $1^e r$ argument
- **false** fait l'inverse
- **if** inutile mais conservé pour **lisibilité**

Récursion et points fixes

- **true** $= \lambda\ x\ y\ .\ x$
- **false** $= \lambda\ x\ y\ .\ y$
- **if** $= \lambda\ f\ x\ y\ .\ f\ x\ y$
- Remarque : terme **false** = codage de **0**
- termes représentant **true** et **1** distincts.
- Possibilité de définir **couples**
- De même que les **listes**

Récursion et points fixes

int fact (int n) {
if (n == 0) return 1;
else return n * fact (n - 1);
}

fact = $\lambda\ n\ .$ **if** (**iszero** n) **1** (**mul** n (**fact** (**pred** n))))

- **if**, **iszero** (test de nullité) et **pred** définis avant
- L'égalité ci dessus n'est pas un terme mais *équation* qui doit être satisfaite
- $F = \lambda\ f\ n\ .$ **if** (**iszero** n) **1** (**mul** n (f (**pred** n)))
- L*équation* devient : **fact** = F **fact**
- **fact** doit être un *point fixe* de F

Récursion et points fixes : **Miracle**

- $\exists$ Y tel que $\forall$ M terme
- $Y\ M \longrightarrow M\ Y\ M$
- $Y =$ *opérateur de point fixe*
- Il fabrique un *point fixe* $\forall$ M
- $\exists$ +sieurs termes ayant cette faculté.
- Le plus simple est le *combinateur paradoxal de Curry*
- $Y = \lambda\ f\ .\ (\lambda\ x\ .\ f\ (x\ x)\ )\ (\lambda\ x\ .\ f\ (x\ x)\ )$

Récursion et points fixes : Vérification

- $Y\ M \longrightarrow \lambda\ (\lambda\ x\ .\ M\ (x\ x)\ )\ (\lambda\ x\ .\ M\ (x\ x)\ ) = N\ N$
- en posant $N = \lambda\ x\ .\ M\ (x\ x)$
- cette définition de $N \Longrightarrow N\ N \longrightarrow M\ (\ N\ N\ )$
- donc $Y\ M \longrightarrow N\ N$, d'où : $M\ (Y\ M) \longrightarrow M\ (N\ N)$
- par réduction de $N\ N$: $Y\ M \longrightarrow M\ (\ N\ N\ )$
- en résumé : $Y\ M \longrightarrow M\ (\ N\ N\ ) \longleftarrow M\ (\ Y\ N\ )$
- terme milieu $\rightarrow$ terme droite est β-expansion : inverse β-réduction
- β-équivalence souvent denotée simplement par le signe d'égalité,

Récursion et points fixes

- Définition exacte de Y pas importante!!! opérateur $Y \exists!$ OK
- **fact** $= Y\ F$
- d'où : **fact** $= Y\ F \rightarrow F\ (\ Y\ F\ )\ =$ F **fact**
- **fact 3** $\longrightarrow$ F **fact 3** $\longrightarrow$
- $\rightarrow$ **if (iszero 3)** 1 (**mul 3** (fact (pred 3))) $\rightarrow$
- $\rightarrow$ **mul 3** (*fact (pred 3))*
- last reduction : **iszero 3** $\rightarrow$ **false** obtenu **if false** M $N \rightarrow N$

Récursion et points fixes

- **mul 3** (fact (pred 3)) $\rightarrow$ **mul 3** (**fact** *2*)
- **fact 2** $\rightarrow$ **mul 2** (**fact** *1*)
- **fact 1** $\rightarrow$ **mul 1** (**fact** *0*)
- **fact 0** $\rightarrow$ *F* **fact 0** $\rightarrow$
- **if (iszero 0) 1 (mul 0 (fact (pred 0)))** $\rightarrow$ **1**
- **iszero 0** $\longrightarrow$ **true**
- **if true** *M N* $\longrightarrow$ *M*
- **fact 3** $\longrightarrow$ **mul 3 (mul 2 (mul 1 1))** $\longrightarrow$ **6**

Théorème de **Church-Rosser**

- Syntaxe λ-calcul très simple
- Une seule règle : β-réduction
- redex = terme de la forme $(\lambda x\ .\ M\ )\ N$
- Un calcul termine lorsqu'on obtient un terme irréductible
- Aucune β-réduction possible, par absence de *redex*
- Les objets de base : **entiers**, **opérateurs sur les entiers** (**addition**, **produit**, etc.), **booléens** et **opérateurs logiques** sont des termes irréductibles
- A l'exception de **fact** qui définit la factorielle

Théorème de **Church-Rosser**

- Un calcul est essentiellement non déterministe (un terme possède en général plusieurs *redex*), et il faut donc examiner si différentes stratégies de réduction d'un terme (c'est-à-dire diffeérents choix du prochain redex à reéduire) influent sur le résultat final obtenu. Le théorème fondamental est le suivant :
- **Théoreème de Church-Rosser**. Si le terme M peut être réduit en M_1 d'une part, et en M_2 d'autre part, alors il existe un terme N tel qu'on puisse réduire à la fois M_1 et M_2 en N.

Théorème de **Church-Rosser**

Supposons maintenant que M puisse être réduit en M_1 et M_2 irréductibles ; d'après le théorème de **Church-Rosser**, les termes M_1 et M_2 peuvent être réduits en un même terme N ; comme par ailleursM_1 et M_2 sont irréductibles, on en déduit que $M_1 = M_2 = N$. Autrement dit :

Si le calcul de M termine, le terme final N (irréductible par définition)
ne dépend pas de la stratégie choisie ; on dit que N est la *forme normale* de M.

Terminaison

- Calcul ne terminant pas : inévitables
- La terminaison dépend de la stratégie choisie
- Exemple classique de terme sans forme normale : $\Omega = (\ \lambda\ x\ .\ x\ x\ )\ (\ \lambda\ x\ .\ x\ x\ )$
- Appliquer à λ la seule β-réduction possible possible on a ... Ω indéfiniment
- **fact** $\rightarrow$ F **fact** $\rightarrow$ $\lambda\ n\ .$ **if (iszero** n) **1** (**mul** n (**fact** (**pred** n)))

Terminaison

- Cette réduction peut être répété à l'infini car **fact** n'a pas de forme normale.
- Cependant **fact 3** $\rightarrow$ **6** qui est irréductible
- $\exists$ expressions avec formes normales, bien que des termes qui la composent n'en ont pas
- En traitant **fact 3** par réduction indéfinie de **fact** en F **fact** (sans jamais se préoccuper de l'argument) ne termine pas.

Stratégies externes

- $\forall \lambda$-*terme* on peut chercher à le réduire
- cas intéressant : applique un terme A, 1 fonction, à un argument U
- Si $A \equiv \lambda x.B$ (**add**, **mul**, etc.),
- une stratégie *externe* consiste à réduire immédiatement le redex :
- $AU = (\lambda x.B)U \longrightarrow B\,[U/x]$
- remplacer toute occurrence libre de x dans B par U
- U peut être un terme complexe, à dupliquer autant de fois que x apparaît dans B
- En informatique cette stratégie est appelée *appel par nom*

Stratégies externes

- Si A n'est pas de la forme $\lambda x.B$ tel **fact**
- La stratégie externe *gauche* $\rightarrow$ récursivement A jusqu'à se ramener au cas précédent
- Pour réduire **fact** U, commencer par : **fact** $\rightarrow F$ **fact**

  fact $U \rightarrow F$ **fact** $U \rightarrow$ **if**(**iszero** U) **1** (**mul** U (**fact** (**pred** U)))
- $F = \lambda\ f\ n\ .$ **if** (**iszero** n) **1** (**mul** n (f (**pred** n)))

Stratégies externes

- Facilement on voit que les conventions sur les fonctions à plusieurs variables entraînent que la stratégie externe revient à substituer simultanément les arguments (ici **fact** et U) aux variables (ici f et n).
- On voit sur cet exemple que l'argument U a ete duplique trois fois, ce qui n'est certainement pas une stratégie efficace si U est un terme complexe, qui après de longs calculs, se réduit par exemple à **4**
- Répéter ces longs calculs (???? **3** fois).
- Poursuivre calcul - choisir tjrs redex le plus externe (non sous-terme d'aucun autre redex) et le plus à gauche

Stratégies externes

Théoreème Une stratégie externe gauche est sûre : si on l'applique à un terme M qui possède une forme normale N, le calcul termine.

- **Exercice** : Continuer la réduction de **fact** U, en appliquant rigoureusement une stratégie externe gauche, et en supposant que la forme normale de U est **4**. Démontrer ce théorème.

Stratégies internes

- Soit le terme à réduire $A\ U$: A est fonction et U argument
- Une stratégie interne consiste à commencer par réduire autant que possible A et U
- Choisir toujours un redex interne $c - a - d$ ne contenant aucun autre redex.
- Une stratégie interne *gauche* réduit A avant de réduire U
- C'est l'inverse pour une stratégie *droite*.
- Evaluer l'argument avant d'exécuter un appel de fonction est la stratégie du langage C
- Usuellement la plus efficace : évite des évaluations répétées du même argument

Stratégies internes

- En prog. impérative, évaluer $A \Longrightarrow$ trouver l'adresse de la procédure à exécuter
- En λ-calcul, évaluer A est souvent nécessaire et longue que évaluer U
- C'est le cas de **fact** et des **booléens**
- Opérateurs à réduire à **true** ou **false** avant de les appliquer aux arguments
- Idem pour les entiers, qui ne sont pas des arguments inertes, mais des opérateurs à evaluer avant de les faire agir.
- L'appel par valeur fonctionne avec une convention restrictive en programmation impérative

Stratégies internes

- Une **stratégie interne stricte** consiste à évaluer à la fois P, U et V avant d'évaluer l'expression conditionnelle, ce qui n'est pas très malin.
- Pour **fact** elle conduit à une suite infinie de réductions de **fact**
- **fact 0** $\rightarrow$ **if** (**iszero 0**) **1** (**mul 0** (**fact** (**pred 0**)))
- $\rightarrow$ **if true 1** (**mul 0** (**fact** (**pred 0**)))
- Une stratégie interne se poursuit stupidement par :
- **pred 0** $\longrightarrow$ **0**
- **fact 0** $\cdots \rightarrow$ **if true 1** (**mul 0** (**fact 0**))
- la suite de réductions internes boucle

Stratégies internes

- Situation embarrassante pour le λ-calcul ????
- Stratégie externe sûre, mais inefficace,
- Stratégie interne stricte conduit fréquemment à des calculs sans fin
- Alors que le terme à réduire possède une forme normale.
- Compromis : en particulier l'*appel par nécessité*, ou *évaluation paresseuse* (*lazy evaluation*)
- Travaux intéressants, mais aucune des stratégies proposeées ne s'est révélée être une panaceée.

Stratégies internes

- Entiers de Church : un seul intérêt théorique
- Un langage fonctionnel utilise la représentation binaire des entiers, et les opérations du processeur sur ces entiers
- ce qui implique, d'une façon ou d'une autre, l'apparition de types et de règles spéciales d'évaluation selon ces types.
- Enfin certaines des difficultés présentées dans cette section apparaissent aussi en programmation impérative
- une instruction : $while\ (\ i >= 0\ \&\&\ x! = t[i]\ ) \cdots$
- $i < 0$ non évaluer $t[i]$ cpdt optimiseur $\subset$ compilateur pour la phase de production de code

- Le λ-calcul, un modele d'une puissance équivalente à celle de Von Neumann
- Ce dernier décrit l'architecture des processeurs usuels
- Ou à celui des machines de Turing
- Une fonction f : **N** $\longrightarrow$ **N** est λ-définissable s'il existe un λ-terme **F** tel que, $\forall$ x :
  1. si $y = f(x)$ est défini, alors **F x** $\longrightarrow$ y,
  2. sinon **F x** n'a pas de forme normale
  3. **x** et **y** sont entiers de Church associés aux entiers x et y

- **y** est irréductible, c'est FN de **F x** dans le 1^{er} cas ci-dessus.
- On peut alors montrer qu'une fonction est λ-définissable sssi elle est calculable
- Le résultat précédent devien
- si $y = f(x)$ est défini, alors **F x** $\longrightarrow$ y,
- sinon **F x** est *irrésoluble*

Objectifs de cette partie cours

P.1 Découvrir un **autre** type de programmation : la programmation fonctionnelle

1. fondé sur la notion de fonction calculable (au sens mathématique),
2. le typage (des données, des fonctions),
3. la récursivité.

P.2

1. Listes : fonctions primitives
2. Fonctions : **gardes**
3. Fonctions : appel par filtrage
4. Récursivité sur les listes