Lambda calculus

L'origine vient d'un questionnement sur l'origine des preuves et l'envie de faire de la preuve automatisé.

On ne peut pas automatisé les preuves. Le theoreme d'incompletdude de Gödel.

> Des choses vrais ne pourront pas être prouver avec un système automatisé.

Apres la guerre les notion de calculabilités ont été démontré équivalente. On parle donc d'une notion de calculabilité.

Il y a plusieur facons de voir le lambda calcul.

- Théorie mathématique
- Langage de programation
- Semantique operationelle (Approche informatique)
- Denotationnelle semantique (Approche mathématique)

Il y a des :

- Variables
- Functions (Abstraction)
- Applications

Il n'y a pas de types !

Un lambda-term:

$M ::= x|(\lambda x.M)|(MM)$

Conventions:

- On peut enlever les parentheses
- L'abstraction est associative a droite
- L'application est associative a gauche
- Plusieurs arguments (sucre synxique)

Il y a plein de notations :

- $x \to 2x + 1$
- $\lambda x.2x + 1$

Variables

- Variables libres : Genre de variables global qui font référence a un contexte.

- Variables lié : Une variables non libre.
- Un terme clos : Un terme sans variable libre.
- Un combinateur : Synonyme de terme clos.

```
int x = 42;
[&x](){x++;}(); // x est une variable libre
[x](){x++;}(); // x n'est pas une variable libre (elle passe par copie)
```

Transformation de lambda terme

L'idee est de considérer équivalent:

- $\lambda x.x$
- $\lambda y.y$
- $\lambda z.z$

On peut renomer les variables. Cependant attention:

- $x\lambda x.x \not\equiv y\lambda y.y$
- $\lambda x.\lambda y.xy \not\equiv \lambda x.\lambda x.xx$

Pour éviter les problemes de renommage. On va les réecrire avec des nom de variables différentes lorsqu'il ne s'agit pas de la meme.

Substitution

> La substitution de x par M dans N est noté $[M/x]N$

Attention c'est une notation pas une operation

Intuitivement toutes les occurrence libre de x sont remplacé par M.

Par exemple :

$[\lambda z.zz/x]\lambda y.xy = \lambda y.(\lambda z.zz)y$

Beta conversion

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

Beta redex

Un β-redex est un terme sous la forme: $(\lambda x.M)N$

L'idée est de réduire petit pas par petit pas.

Example:

- $(\lambda x.x)y \to y$
- $(\lambda x.xx)y \to yy$
- $(\lambda x.xx)(\lambda x.xx) \to (\lambda x.xx)(\lambda x.xx)$

Le troisieme exemple nous montre que la beta reduction ne simplifie pas toujours.

Omega combinators

- $\omega \equiv \lambda x.xx$
- $\Omega \equiv ww$
- $\widetilde{\Omega} \equiv \lambda x.x(xx)$

Plus d'exemple (qui se passe mieux):

- $(\lambda x.xyx)\lambda z.z \to (\lambda z.z)y(\lambda z.z)$
- $(\lambda x.x)((\lambda y.y)x) \to (\lambda x.x)(x)$
- $(\lambda x.x)((\lambda y.y)x) \to ((\lambda x.x)x)$
- $(\lambda x.x)((\lambda y.y)x) \overset{*}{\to} x$

Forme normal

- Identite : $\lambda x.x$ est en forme beta normal.
- II a une forme beta normal. On peut effectuer un Beta réduction.
- II est donc fortement normalizable.
- Ω est faiblement normalizable.
  $\Omega = (\lambda x.xx)(\lambda x.xx) \to (\lambda x.xx)(\lambda x.xx) = \Omega$
- $KI\Omega$ est faiblement normalizable. $(K = \lambda x.(\lambda y.x))$
  $KI\Omega \to I$

Propriete de Church-Rosser

- Si R a la propriete de Church-Rosser alors tous les programmes sont Beta convertible.

- Si R est Church-Rosser alors il a une unique forme normale.

Propriété :

> **B-Reduction est Church-Rosser**

Donc tous les termes a au moins une unique forme normale.

Types

Le lambda calcul est un vrai langage fonctionnel. On peut donc représenté les types classiques voici comment.

Booleen

On aimerai un booleen qui soit une fonction qui prend deux arguments.

MNL

Si le booleen est vrai il renvoi le premier argument N sinon L.

- $T := \lambda ab.a$
- $F := \lambda ab.b$

Integer

$n := \lambda f.\lambda x.f^n x = \lambda f.\lambda x.(f...(fx)...)$

Donc:

- $0 = \lambda fx.x$
- $1 = \lambda fx.fx$
- $2 = \lambda fx.f(fx)$
- $3 = \lambda fx.f(f(fx))$

Succ

$succ := \lambda n.\lambda f.\lambda x.f(nfx)$

Plus

$plus := \lambda m.\lambda n.\lambda f.\lambda x.mf(nfx)$

Pair

- $pair := \lambda xy.\lambda f.fxy$ (Constructeur)
- $first := \lambda p.pT$ (Accesseur 1er element)
- $second := \lambda p.pF$ (Accesseut 2nd element)

Recursion

On utilise les combinateurs de points fixes. Il en existe plusieurs par exemple:

Curry's Y Combinator:

> $Y := \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$

Turing; s Θ Combinator:

> $\Theta := (\lambda xy.y(xxy))(\lambda xy.y(xxy))$

Logique combinatoire

L'idée est quue l'on a plus besoin de λ.

On a besoin de:

- I fonction identité
- K fonction boolean (True)
- S fonction distributeur

Tq:

- $Ix = x$
- $Kxy = x$
- $Sxyz = xz(yz)$

La logique combinatoire à la même expressivité que le lambda calcul.

C'est très simple mais peut lisible du coup on a garder le lambda calcul.

Lambda calcul typé

La notion de type sont des objets syntaxique associé a un terme.

Cela a pour but d'interdir des lambdas termes qui ne peuvent pas se normaliser et qui ont par ailleur pas beaucoup de sens.

Exemple:

$\omega := \lambda x.xx$

On a un symbol $\rightarrow$ et un ensemble de types ($\alpha, \beta, ...$)

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

$\lambda^{\rightarrow}$ est fortement normalisable. C'est a dire que tout lambda expression (typé) est normalisable.

Sequent

Un sequent est une suite d'hypothèses suivie d'une suite de conclusions, les deux suites etant usuellement séparées par le symbole $\vdash$.

System LK

Logistischer Klassischer.

Logique constructiviste

Contrairement a la logique classique qui ne fait que découvrir la vérité, la logique constructiviste créer de nouvelle preuve.

On considére que les choses ne sont pas forcément vraie ou fausse. On supprime le **tier exclus**.

Qu'est ce que ca veut dire de réfléchir par l'absurde ?

> On l'utilise pour prouver l'inexistance d'un objet (Ex: $\sqrt{2} \notin \mathbb{Q}$).

Mais on peut aussi l'utiliser pour prouver l'existance d'un objet sans pouvoir le construire. C'est génant, on appelle ca le probleme de constructivité.

On aimerai être capable d'exiber un témoins des preuves d'existance.

Ex:

> Il existe des réels qui ne peuvent pas etre ecrit par un programme (meme s'il tourne à l'infini). Car il y a une bijection entre les programme et $\mathbb{N}$. Alors que $\mathbb{N} < \mathbb{R}$

Une frontiere entre math et informatique est la constructibilité:

- L'informaticien s'en moque de savoir si un objet existe il veut pouvoir le construire.
- Le matématicien s'en moque de la constructibilité il veut savoir si un ensemble est plus grand qu'un autre ou si un objet a une propriété.

Elimination du tier exclus

Plein de chose vienne du tier exclu et qui seront donc éliminer en logique intuitionniste.

Exemple:

- La double négation
- La suppression de la négation
- La contradiction

On va donc s'interdir la négation. On définit alors:

$\neg A$ revient à écrire $A \implies \bot$

On peut toujours prouver: $\neg\neg A \vdash A$ et $\neg\neg\neg A \vdash \neg A$

On a super donc super envi de supprimer la double négation mais on ne peut pas car ca ne marche pas dans tous les cas.

Calcul des séquent en intuitionniste

la régle du calcul intuitionniste:

- A droite du séquent on a une seule formule a droite du séquent. (Alors qu'en classique on peut avoir un groupe de formule)

On peut montrer ca en introduisant les polarités.

En résumé, souvent quand on fait des preuves par l'absurde on perd quelque chose qui est la constructibilité.