Lambda calcul = un mécanisme de calcul.
l'appel de fonction
β-réduction

↳ Evaluation d'expressions.

ex: affectation
$v \leftarrow$ expr $2*x + 5*y$
(fonction appliquée à arguments).

Définir une fonction : $\lambda x.$ corps.

Chapitre 1 : - définitions
- variables libres, liées, portées.

Chapitre 2 : - Etude de la puissance de calcul.

→ On peut tout faire : - structures de données
- structures de contrôle : alternatives
boucles (For ≠ While)

• Boucle "for"
Pour i de 1 à n
faire
|
⟺
Entiers de Church
$C_n = \lambda f x. f(\dots f(x))$

• Boucle "While"
Tant que cond.
faire
|
⟺
Fonctions "récursives"
fact n =
if n = 0 then 1
else n * fact (n-1)

Chapitre 3 : - typage

→ tout terme typé (son évaluation) termine

- conséquence 1 : impossible d'écrire fact' en λ calcul typé

impossible ↓

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

- conséquence 2 : fact (et toutes fct° "récursives" qui terminent) doit s'écrire autrement.

$$\frac{u : T_1 \to T_2 \qquad V : T_1}{uV : T_2}$$

$$\frac{\Gamma, x : T_1 \vdash u : T_2}{\Gamma \vdash \lambda x^{T_1}.u : T_1 \to T_2}$$

bool : $T \to T \to T$	(testO Cn) N_1 N_2
true : $\lambda x^T y^T . x$	⇓
false : $\lambda x^T y^T . y$	On veut paramétrer le type dans
(true u V) : T	bool, nat ...

⇒ <u>lambda Calcul Polymorphe</u> : système F.

TERMES TYPES :

- <u>variables typées</u> : x^T (Noms typés)
- <u>application</u> : uV $\quad \dfrac{u : T_1 \to T_2 \qquad V : T_1}{uV : T_2}$

- <u>abstraction</u> : $\lambda x^{T_1}.u : T_1 \to T_2$ si $u : T_2$.
- <u>variable de Type</u> : X, T, ... (Nom de type)

  En Coq : X : Set, T : Set

- abstraction de type : $\Lambda T.\ (\dots)$ ← expr normale $\lambda x^T.x$
  - Λ : lambda majuscule

$\Lambda T.\ \lambda x^T y^T.x$

- application de type :

$(\Lambda T.\ \lambda x^T y^T x)\ \underbrace{cbool}_{\text{un type}}$

TYPES EUX-MEME :

- variables (ou nom) de types : $X, T, \dots$
- types prédéfinis : nat, bool...
- types flèches : $T_1 \to T_2$
- types polymorphes : $\forall T.$ (expr de type)
  ex : $\forall T\ (T \to T \to T)$

→ Υ est un type pouvant utiliser T.
$\Lambda T.\ E : \forall T, \Upsilon$

Booléens polymorphes :

$pbool = \forall T, T \to T \to T$

$ptrue = \Lambda T.\ \lambda x^T y^T.x$

$pfalse = \Lambda T.\ \lambda x^T y^T.y$

$\underbrace{pNeg}_{pbool \to pbool} = \lambda b^{pbool}.\ \underbrace{\Lambda T.\ \lambda x^T y^T.\ bT\,y\,x}_{\substack{pbool \\ \forall T, T \to T \to T}}$

$\underbrace{pNeg'}_{pbool \to pbool} = \lambda b^{pbool}.\ b\ pbool\ pfalse\ ptrue$

$\underbrace{pNeg''}_{pbool \to pbool} = \lambda b^{pbool}.\ \Lambda T.\ b\ (T \to T \to T)\ \underbrace{(pfalse\ T)}_{T \to T \to T}\ (ptrue\ T)$

06/03/2018

λ-CALCUL

1 CM

Codage

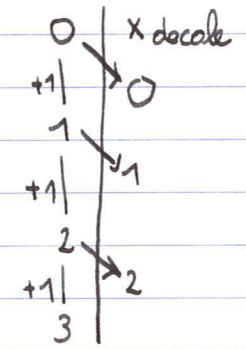
- booléen : $[true] = \lambda x y . x$
- entier : $[n] = \lambda f x . f(\dots f(x))$ (f n fois)
- 0 : itérateur
- successeur : $[0] = \lambda f x . x$
- plus : $[s] = \lambda n . \lambda f x . f(n f x)$
- multiplier : $[plus] = \lambda n m . \lambda f x . m f (n f x)$
- exp
- test à 0

Soustraction

- soustraire m : itérer m fois : soustraire 1

prédécesseur : - itérateur n fois l'opération

intuition



(a, b)
(a+1, b)

puis extraire le 2e elt
couple initial = (0, 0)

Nécessité d'avoir des couples

- fabrication
- extraction des composants

En Ocaml

```
let succ (a,b) = (Sa, a)
let pred n =
    let (x,y) = iter n succ (0,0)
    in y
```

Méthode générale : la fabrication est déterminée par l'utilisation.
Pour les couples utilisés = décomposer (destruction)
en particulier : projeter

1 bis : utiliser x = appliquer x à quelque chose : $x\ u$ ...
En effet, x est une fonction qui doit être appliquée à quelque chose

Pour les couples, soit c est un couple, c [?] doit donner qqchose d'interessant

Donc match c with $(x, y) \to R$: $\lambda x\, y.\, R$ (R represente un resultat, exe : $x+y$)

codé par $c\ (\lambda x\, y.\, R)$ -> continuation

Donc c de la forme $\lambda h.\ \square$

Au final : soient A et B deux termes, $[\![(A,B)]\!] \overset{def}{=} \lambda h.\ h\ A\ B$

exemple : $[\![(0,0)]\!] = \lambda h.\ h\ (\lambda f x.\ x)\ (\lambda f x.\ x)$

↳ de fabrication

exemple : proj 1 (= recuperer le 1er element d'un couple)

↳ utilisation : $[\![\text{proj 1}]\!] = \lambda c.\ c\ (\lambda x\, y.\, x)$

Verification :

$$\begin{aligned}[\text{proj 1}\ (2,1)] &= (\lambda c.\ c\ (\lambda x\, y.\, x))\ [(2,1)] \\ &= [(2,1)]\ (\lambda x\, y.\, x) \\ &= \lambda h.\ h\ ([2])\ ([1])\ (\lambda x\, y.\, x) \\ &= (\lambda x\, y.\, x)\ ([2])\ ([1]) \\ &= \dots \\ &= [2]\end{aligned}$$

$[\text{proj 2}] = \lambda c.\ c\ (\lambda x\, y.\, y)$

$[\text{plus cpl}] = \lambda c.\ c\ [\text{plus}]$

$[\text{open}] = \lambda c.\ c\ (\lambda a\, b.\, \lambda h.\ h\ (\lambda f x.\ f\ (a\ f\ x))\ a$

$[\text{pred}] = \lambda n.\ n\ [\text{open}]\ (\lambda h.\ h\ (\lambda f x.\ x)\ (\lambda f x.\ x))\ (\lambda x\, y.\, y)$

typage de couples

Exemple : couple d'entier

$(2,1) = \lambda h.\ h\ 2\ 1$

en typage simple h : nat -> nat -> nat

marche pour

- Les deux projections : $\text{proj 1} = \lambda c^{(nat \to nat \to nat) \to nat}.\ c\ (\lambda x^{nat}\, y^{nat}.\, x)$

  $\text{proj 2} = \lambda c^{(nat \to nat \to nat) \to nat}.\ c\ (\lambda x\, y.\, y)$

PB1- $[\text{test du 1er element}] = \lambda c.\ c\ (\lambda x\, y.\ \underbrace{[\text{test 0}]\ x}_{nat \to nat \to bool})$

$c.\ (nat \to nat \to bool) \to bool$