Alexandre DIKOVSKY
Université de Nantes

Lambda-calcul et Systèmes de Réécriture

Cours 1. Calculabilité et Lambda définissabilité des fonctions

1 Introduction

Comment calcule-t-on les fonctions ? Deux idées magistrales :
Nouvelle [Turing, Fon Neuman] : Par transformations locales du contenu des cellules addressées de la mémoire selon des systèmes finis des instructions (programmes).
Ancienne : Par substitution des sous expressions par ses valeurs.

Exemple 1. *Calculs numériques :*

$$\begin{array}{rcl}(7+4)*(8+5*3) & \rightarrow & 11*(8+5*3)\\ & \rightarrow & 11*(8+15)\\ & \rightarrow & 11*23\\ & \rightarrow & 253.\end{array}$$

Exemple 2. *Calculs symboliques :*

$$\begin{array}{l}\dot{p}n\quad(t\quad(h\quad((\text{"}\dot{s}\quad\text{"},\text{"}h\quad\text{"})t\quad((\text{"}h\quad\text{"},\text{"}h\quad\text{"}))))) \rightarrow\\ \dot{p}n\quad(t\quad(h\quad((\text{"}\dot{s}\quad\text{"},\text{"}h\quad\text{"})(\text{"}h\quad\text{"},\text{"}h\quad\text{"})))) \rightarrow\\ \dot{p}n\quad(t\quad((\text{"}\dot{s}\quad\text{"},\text{"}h\quad\text{"},\text{"}h\quad\text{"},\text{"}h\quad\text{"}))) \rightarrow\\ \dot{p}n\quad((\text{"}h\quad\text{"},\text{"}h\quad\text{"},\text{"}h\quad\text{"},\text{"}\dot{s}\quad\text{"})) \rightarrow\\ \text{"}h\quad\text{"}.\end{array}$$

SCHEMA : $\mathbf{Val}(f(\not{A}_1 \ldots \not{A}_n)) \rightarrow \mathbf{Rule}_f(\mathbf{Val}(\not{A}_1) \ldots \mathbf{Val}(\not{A}_n))$.

Comment définit-on les règles $\mathbf{Rule}_f$:
- par programmes integrés (ex. 1),
- par récurrence sur les structures des arguments (ex. 2),
- directement par des systèmes de substitution :

Exemple 3. *Calculs textuels :*
Rules :

$$\begin{array}{lcl}\mathbf{r_1} & \not{B} & \rightarrow N\\ \mathbf{r_2} & \not{B} & \rightarrow N\\ \mathbf{r_3} & \not{N} & \rightarrow B\\ \mathbf{r_4} & B & \rightarrow B\end{array}$$

Calcul :

$$\begin{array}{l}\not{B}\_\not{B} \rightarrow\\ \not{N}\_B \rightarrow\\ \not{B}\_ \rightarrow\\ \underline{\not{B}\quad\not{B}} \rightarrow\\ \not{B}\_ \rightarrow\\ \underline{\not{N}\_} \rightarrow\\ B\end{array}$$

PROBLEMES :

PROBLEME TECHNIQUE : Si les expressions *avec les variables* représentent les fonctions (des certaines variables) et les arguments, alors comment savoir quel argument correspond à quelle variable :

$$((Z\ (X\quad 2))\ Y\quad 5,3)\ \rightarrow\ X*Y*2+5+3\ ?$$

PROBLEMES FONDEMENTAUX : Dans le schéma des calculs fonctionnels
(*i*) le résultat, dépende-t-il de l'ordre d'application des règles $\mathbf{Rule}_f$ dans les occurrences différentes ?
(*i*) ce résultat, existe-t-il toujours ?
(*i*) peut-on calculer tout ce qui est calculable par des règles pareilles de calcul fonctionnel ?

Le Lambda calcul et les systèmes de réécriture donnent un cadre théorique formel pour résoudre ces problèmes. Les deux moyens sont utilisé pour réaliser les langages de programmation fonctionnelle : LISP, ML, MIRANDA, CAML, etc.

2 Fonctions récursives

Ce sont les fonctions arithmétiques sur N définies par composition et récurrence à partir des fonctions primitives.

Définition 1. [*Gödel*]
0. $\mathbf{0}$ *est une seule constante.*
1. **Fonctions primitives**
(1) $s(x) = x+1$ *% successeur*
(2) $\mathbf{Z}(x) = 0$ *% zéro*
(3) $I_n^k(x_1, x_k, x_n) = x_k$ *% projection*
2. **Composition**
$f(\overline{x}) = h(g_1(\overline{x}), \ldots, g_m(\overline{x}))$
3. **Récursion primitive**
$f(\overline{x}, 0) = g(\overline{x})$
$f(\overline{x}, s(y)) = h(\overline{x}, y, f(\overline{x}, y))$
% f est définie par récurrence sur y à partir de g, h
4. **Minimisation**
$f(\overline{x}) = \mu y\ (t(y, \overline{x}) = \mathbf{0})$
% μy est la solution minimale de l'équation $t(y, \overline{x}) = 0$ (si elle existe).
La classe des fonctions récursives primitives $\mathcal{FR}^{prim}$ *(*récursives partielles $\mathcal{FRP}$*) est l'ensemble minimal des fonctions contenant les fonctions primitives et fermé par la composition et la récursion primitive (et aussi par la minimisation).*

Exemple 4. *(1)* **Prédécesseur** [1]

$$\begin{cases} pr(\mathbf{0}) &= \mathbf{0} \\ pr(s(y)) &= y \end{cases}$$

[1] Dans la deuxième équation, $h(x,y) = I_2^2(x,y)$. C'est-à-dire, $pre(s(y)) = h(pre(y), y) = y$.

(2) **Addition à partir de successeur et projection**

$$\begin{cases} x + \mathbf{0} &= x \\ x + s(y) &= s(x + y) \end{cases}$$

(3) **Soustraction réduite à partir de prédécesseur**

$$\begin{cases} x \dot{-} \mathbf{0} &= x \\ x \dot{-} s(y) &= p(x \dot{-} y) \end{cases}$$

(4) **Multiplication à partir d'addition et zéro**

$$\begin{cases} x * \mathbf{0} &= \mathbf{0} \\ x * s(y) &= x * y + x \end{cases}$$

(5) **Exponentiation à partir de multiplication**

$$\begin{cases} x^{\mathbf{0}} &= s(\mathbf{0}) \\ x^{s(y)} &= x^y * x \end{cases}$$

(6) **Carrée par excés à partir de multiplication et nonégalité**
Supposons que $x \geq y$ est exprimé. Alors, la fonction

$$\phi_{\geq}(x) = \mu y\ (y * y \geq x)$$

est définie pour tous x

Définition 2. [*Gödel*]
Une fonction récursive partielle est récursive *si elle est totale. La classe des fonctions récursives est notée $\mathcal{FR}$.*

Corollaire 1. *Toute fonction récursive primitive est totale, par conséquant récursive.*

La majorité des fonctions calculables utilisées dans les applications réélles sont récursives primitives. Toutes les fonctions (1)-(6) le sont. En voici encore plusieurs exemples :

Exemple 5. *(7)* $sg(x) =_{df}$ **si** $x = \mathbf{0}$ **alors** $\mathbf{0}$ **sinon** $s(\mathbf{0})$.
(8) $x = y$ *ou* $x \geq y$
(9) les fonctions booléennes : $\phi \wedge \psi, \neg\phi, \phi \vee \psi, \phi \rightarrow \psi$.
(10) $IE(\phi, x, y) =$ **si** ϕ **alors** x **sinon** y

Il est clair que toute fonction récursive est calculée par un algorithme. Décrivez par exemple un algorithme qui calcule μy si un y existe. Par ailleurs, toute fonction récursive n'est pas récursive primitive.

Exemple 6. *Fonction d'Akkerman :*

$$\begin{cases} A(\mathbf{0}, y) &= s(y) \\ A(s(x), \mathbf{0}) &= A(x, s(\mathbf{0})) \\ A(s(x), s(y)) &= A(x, A(s(x), y)). \end{cases}$$

Théorème 1. *[Akkerman] La fonction $A(x, y)$ est récursive et pour toute fonction récursive primitive $f(n)$, il existe un entier c_f tel que $\forall n\ A(n, n) > f(n)$.*

Ainsi, $\mathcal{FR}^{prim} \subsetneq \mathcal{FR} \subsetneq \mathcal{FRP}$. Gödel et Turing ont prouvé le théorème fondamental suivant.

Théorème 2. *[Gödel, Turing] La classe des fonctions arithmétiques calculées par les MTD coïncide avec la classe $\mathcal{FRP}$.*

3 Expressibilité des fonctions récursives dans l'arithmétique

Théorie de l'égalité

On choisit une signature Σ ($= /2 \in \Sigma_p$).

$\mathcal{EQ}_\Sigma$: (e_1) $\forall x\ (x = x)$

(e_2) $\overline{\forall}\ (x_1 = y_1 \wedge \ldots \wedge x_n = y_n \rightarrow f(x_1,\ldots,x_n) = f(y_1,\ldots,y_n))$ (pour tout $f^n \in \Sigma_f$)

(e_3) $\overline{\forall}\ (x_1 = y_1 \wedge \ldots \wedge x_n = y_n \rightarrow (p(x_1,\ldots,x_n) \rightarrow p(y_1,\ldots,y_n)))$ (pour tout $p^n \in \Sigma$, $= /2$ inclu).

Par exemple, on peut prouver que les formules

$$\phi_{sme} \hat{=} \forall x,y\ (x = y \rightarrow y = x),$$
$$\phi_{tre} \hat{=} \forall x,y,z\ (x = y \wedge y = z \rightarrow x = z)$$

sont des conséquences logiques de $\mathcal{EQ}_\Sigma$: $\mathcal{EQ}_\Sigma \models \phi_{sme}$, $\mathcal{EQ}_\Sigma \models \phi_{tre}$.

Arithmétique de Robinson.

R : $(r1)$ $x = y \rightarrow x' = y'$

$(r2)$ $x = y \rightarrow (x + z = y + z \wedge z + x = z + y)$

$(r3)$ $x = y \rightarrow (x * z = y * z \wedge z * x = z * y)$

$(r4)$ $x' = y' \rightarrow x = y$

$(r5)$ $\neg(\mathbf{0} = x')$

$(r6)$ $\neg(x = \mathbf{0}) \rightarrow \exists y\ (x = y')$

$(r7)$ $x + \mathbf{0} = x$

$(r8)$ $x + y' = (x + y)'$

$(r9)$ $x * \mathbf{0} = \mathbf{0}$

$(r10)$ $x * y' = (x * y + x)$

On peut montrer que $\mathcal{EQ}_{ar} \subseteq \mathcal{T}h^1(\mathbf{R}) \subseteq \mathcal{T}h^1(\mathbf{Ar})$.

Arithmétique de Peano.

S : $(s1)$ $\neg(\mathbf{0} = x')$

$(s2)$ $x' = y' \rightarrow x = y$

$(s3)$ $x + \mathbf{0} = x$

$(s4)$ $x + y' = (x + y)'$

$(s5)$ $x * \mathbf{0} = \mathbf{0}$

$(s6)$ $x * y' = (x * y + x)$

(s_ϕ) $\phi(\mathbf{0}) \rightarrow (\forall x\ (\phi(x) \rightarrow \phi(x')) \rightarrow \forall x \phi\ (x))$

(ind) est l'axiome d'induction. ϕ est une métavariable qui porte sur les formules dans la signature Σ_{ar}. Alors, ce système est infini et dénombrable.

Il s'avère que toute fonction (ou relation) calculable est exprimée en **S** et même en **R** . Soit $\lceil n \rceil$ un code des entiers par les termes arithmétiques. Par exemple, 0 est codé par **0** et $n > 0$ est codé par $\mathbf{0}'^{\cdots'}$ (n fois).

Définition 3. *Une relation arithmétique* R *exprimée en* **S** *par une formule* $\phi(x_1, \ldots, x_k)$ *:*
(1) *si* $R(n_1, \ldots, n_k) = 1$, *alors* $\phi(\lceil n_1 \rceil, \ldots, \lceil n_k \rceil) \in Th_1(\mathbf{S})$,
(2) *si* $R(n_1, \ldots, n_k) = 0$, *alors* $\neg\phi(\lceil n_1 \rceil, \ldots, \lceil n_k \rceil) \in Th_1(\mathbf{S})$,
Une fonction arithmétique f *exprimée en* **S** *par une formule* $\psi(x_1, \ldots, x_k, y)$ *:*
(3) *si* $f(n_1, \ldots, n_k) = n$, *alors* $\psi(\lceil n_1 \rceil, \ldots, \lceil n_k \rceil, \lceil n \rceil) \in Th_1(\mathbf{S})$,
(4) $\forall x, y \ (\psi(\lceil n_1 \rceil, \ldots, \lceil n_k \rceil, x) \wedge \psi(\lceil n_1 \rceil, \ldots, \lceil n_k \rceil, y) \rightarrow x = y) \in Th_1(\mathbf{S})$ *pour tous* $n_1, \ldots, n_k$.

Théorème 3. *Les fonctions et les relations récursives partielles sont exprimées en* **R** *(et ainsi, en* **S***).*

4 Lambda notation, Lambda termes

C'est un langage fonctionnel symbolique proposé par Church, dont la sémantique est définie en termes d'application d'une fonction à une valeur.

λ-termes. Soit un ensemble dénombrable V de variables.
(i) $M \equiv x$ $x \in V$ est un λ-terme ; l'occurrence de x est *libre* dans M
(ii) Soient λ-termes M, N Alors, $T \equiv (MN)$ l'est aussi. Toute occurrence d'une variable x libre dans M ou N est libre dans T
(iii) Soient un λ-terme M et une variable $x \in V$ Alors, $T \equiv \lambda x.M$ l'est aussi. x est liée dans T et M est la *portée* de λx dans T Toutes les variables libres dans M apart x restent libres dans T

Convention :

$$\lambda x\ (\lambda y.M) =_{df} \lambda xy.M \quad ; \quad ((MN)T) =_{df} MNT$$

Exemple 7. *Dans* $\mathbf{I} \equiv \lambda x.x$ *est liée. Dans* $(\mathbf{I}z)$, x *est liée et* z *est libre. Dans* $\mathbf{K} \equiv \lambda xy.x$ *et dans* $\mathbf{S} \equiv \lambda xyz.xz(yz)$ *toutes les occurrences des variables sont liées.*

Notation. $T \equiv M[x := N]$ veut dire que T est le résultat de remplacement dans M de toute occurrence libre de x par N Par exemple,

$$x\ (yz)[z := K] \equiv K\ (yK)\ ; \quad \lambda xyz.x(yz)[z := K] \equiv \lambda xyz.x(yz).$$

Simulation des calculs par conversion des Lambda termes
Conversion naive de Lambda termes. Soient les relations binaires suivantes :
(α) $\lambda x.M \rightarrow_\alpha \lambda y.M[x := y]$.
(β) $(\lambda x.M)N \rightarrow_\beta M[x := N]$ (le terme $(\lambda x.M)N$ s'appelle *rédex*).

La relation $\twoheadrightarrow$ de *conversion de Lambda termes* est la fermeture reflexive-transitive de $\rightarrow_{\alpha} \cup \rightarrow_{\beta}$. Nous allons utiliser cette relation pour simuler les processus de calcul :

$$T \twoheadrightarrow V$$

doit signifier que V est une *valeur* du terme T

Exemple 8. $SKK \equiv (\lambda xyz\ (xz(yz)))(\lambda xy)(\lambda xy)x \twoheadrightarrow (\lambda x_1 yz\ _1z(yz))(\lambda x_2y_1x\ _2)(\lambda x_3y_2x\ _3)x$
$\twoheadrightarrow (\lambda yz\ (\lambda x_2y_1x\ _2)z(yz))(\lambda x_3y_2x\ _3)x \twoheadrightarrow (\lambda z\ (\lambda x_2y_1x\ _2)z((\lambda x_3y_2x\ _3)z))x \twoheadrightarrow (\lambda z\ (\lambda x_2y_1x\ _2)z(\lambda y_2z\ ))x$
$\twoheadrightarrow (\lambda x_2y_1x\ _2)x(\lambda y_2x\ ) \twoheadrightarrow (\lambda y_1x\ )(\lambda y_2x\ ) \twoheadrightarrow x$

Calculs naifs des fonctions
On peut simuler les calculs des foctions booléennes, arithmétiques et autres en utilisant cette relation de conversion même sur les Lambda termes sans constantes.

Booléens. On peut simuler ≪vrai≫ et ≪faux≫ par les termes $T =_{df} K \equiv \lambda xy$ et respectivement $F =_{df} \lambda xy$ et définir les opérateurs standards sur ces valeurs par les termes suivants :
$and \equiv \lambda xyF$
$or \equiv ?$
$not \equiv \lambda xy$
$et \equiv ?$
Tests : $andTF \equiv (\lambda xyF)TF \twoheadrightarrow TTF \equiv (\lambda xy)TF \twoheadrightarrow F$
$orTT \equiv ?$
$notF \equiv (\lambda xy)F \twoheadrightarrow \lambda xyF \equiv \lambda xy\ (\lambda x_1y_1y\ _1)y \twoheadrightarrow \lambda xy \equiv T$
$etNT \equiv ?$

Structures : paires.
$a \equiv \lambda xyz$ % une paire
$\pi_1 \equiv \lambda pT$ % le premier membre
$\pi_2 \equiv \lambda pF$ % le deuxième membre
Test : $\pi_2\ (a\ b) \equiv ?$

Entiers. Barendregt a proposé de simuler 0 par $\lceil 0 \rceil =_{df} I \equiv \lambda x$ et l'entier $n+1, n \geq 0$ par le λ-terme $\lceil n+1 \rceil =_{df} aF\ \lceil n \rceil$ (en particulier, $\lceil 1 \rceil \equiv aF\ \lceil 0 \rceil$). Voici les λ-termes qui simulent plusieurs fonctions arithmetiques simples :
$zero \equiv \lambda nT$ % test $n = 0$
$I_n^k \equiv \lambda x_1 x\ _n x\ _k$ % projection
$suc \equiv \lambda n aF$ % successeur
$pred \equiv \lambda n\ (zero\ n)I(\pi_2\ n)$ % prédécesseur

Combinateur de point fixe (de récursion). On dit qu'un λ-terme fermé Y est **combinateur de point fixe (PF)** si :

$$\forall M\ (YM \twoheadrightarrow M(YM))$$

Théorème 4. *[Kleene]* $Y =_{df} \lambda f\ (\lambda x f\ (x\ x))(\lambda x f\ (x\ x)))$ *est un combinateur de PF.*
Preuve : Soit $W \equiv \lambda x M\ (x\ x)$. Alors, $YM \equiv WW \twoheadrightarrow M(WW) \equiv M(YM)$.

Ainsi on peut simuler les calculs des fonctions récursives.

Soit par exemple, une fonction $f(\overline{x})$ définie par la récursion primitive :

$$\begin{cases} f(\overline{x}, 0) & = & g(\overline{x}) \\ f(\overline{x}, y+1) & = & h(\overline{x}, y, f(\overline{x}, y)) \end{cases}$$

Alors, $M_f =_{df} YT_f$, où :

$$T_f \equiv \lambda f \overline{x} y.\, (zero\ y)\ (M_g\ \overline{x})\ (M_h\ \overline{x}\ (pred\ y)\ (f\ \overline{x}\ (pred\ y)))$$

et M_g, M_h sont des λ-termes qui définissent respectivement g et h

Exemple 9. *Définition de $m+n$: $M_+ =_{df} YT_+$, où :*

$$T_+ \equiv \lambda f m n.\, (zero\ n)\ m\ (succ\ (f m\ (pred\ n))).$$

Comparons cette définition de $'+'$ avec celle en Caml :

```
# let rec sum(m, n) = if n = 0 then m else (sum(m, (n - 1))) + 1;;
sum : int × int → int = <fun>
```

Hélas, nos définitions demandent une fondation théorique : pour les utiliser pour calculer les fonctions il faut absolument résoudre les problèmes fondamentaux cités ci-dessus.