

Analyse des Programmes et Sémantique

Mathieu Jaume

Renaud Rioboo

APS - 2006-2007

Calculabilité

Que peut-on implanter à l'aide de « programmes » ?

- Un modèle fonctionnel : le λ -calcul.
- un modèle abstrait d'ordinateur : les URM.

Les URM (Unlimited Register Machine)

Une URM $\mathcal{R} = \{R_i\}_{i \geq 1}$ est un ensemble de registres pouvant contenir des éléments de \mathbb{N} .

Un programme $P = \{I_i\}_{i=1}^{i=n}$ est une suite finie d'instructions.

Une instruction est soit :

- une remise à zéro du i -ème registre : $Z(i)$,
- l'incréméntation du i -ème registre : $S(i)$,
- le transfert du contenu du i -ème registre dans le j -ème registre : $T(i, j)$,
- un saut conditionnel à la k -ème instruction lorsque les i -ème et j -ème registres sont égaux : $J(i, j, k)$

Fonctionnement

Une URM exécute les instructions d'un programme en séquence sauf lorsqu'elle rencontre une instruction de saut.

La configuration ou l'état d'un programme est l'ensemble des valeurs $\{a_i\}_{i \geq 1}$ contenues dans les registres.

La configuration initiale d'un programme est celle où les premiers registres contiennent les données $\{d_i\}_{i=1}^{i=d}$ et où tous les autres registres sont à zéro :

$$\left\{ \begin{array}{l} R_i \text{ contient } d_i \text{ si } i \leq d \\ R_i \text{ contient } 0 \text{ si } i > d \end{array} \right. \mathcal{R} = \{d_1, \dots, d_d, \vec{0}\}$$

Arret

Un programme s'arrête lorsque le contrôle passe à une instruction au delà de la dernière.

Par convention les résultat d'un programme est le contenu du premier registre lorsque le programme s'arrête.

Exemple :

$$\mathcal{R}_0 = \{0, \vec{0}\}$$
$$P : I_1 = J(1, 1, 1)$$

est un programme qui ne s'arrête pas !

Un exemple plus compliqué

$$\mathcal{R}_0 = \{9, 7, 0, \vec{0}\}$$

$$P : I_1 = J(1, 2, 6)$$

$$I_2 = S(2)$$

$$I_3 = S(3)$$

$$I_4 = J(1, 2, 6)$$

$$I_5 = J(1, 1, 2)$$

$$I_6 = T(3, 1)$$

Idem avec

$$\mathcal{R}_0 = \{8, 4, 2, \vec{0}\}$$

$$\mathcal{R}_0 = \{2, 3, 0, \vec{0}\}$$

Comment montrer la correction du résultat ?

Notations

Soit $\{d_i\}_{i \geq 1}$ une configuration initiale et P un programme.

- $P(\{d_i\})$ l'exécution de P avec la configuration initiale $\{d_i\}_{i \geq 1}$.
- $P(\{d_i\}) \downarrow$ si $P(\{d_i\})$ s'arrête. Le calcul converge.
- $P(\{d_i\}) \uparrow$ si $P(\{d_i\})$ ne s'arrête pas. Le calcul diverge.
- $P(\{d_i\}) \downarrow r$ si $P(\{d_i\}) \downarrow$ avec r dans le premier registre. On ne s'intéresse pas aux autres registres.
- Quelles sont les fonction de \mathbb{N}^d dans \mathbb{N} que l'on peut implanter à l'aide d'une URM ?
- Peut-on implanter avec d'autres machines des fonctions non implantables avec des URM ?

fonctions URM-calculables

Soit f une fonction (partielle) de \mathbb{N}^d dans \mathbb{N} ,
 P un programme URM. P calcule f si

$$\forall d_1 \dots d_d r \in \mathbb{N}, P(\{d_i\}) \downarrow r \iff f(d_1, \dots, d_d) = r$$

- $P(\{d_i\}) \downarrow$ pour toutes les valeurs $d_1 \dots d_d$ où f est définie,
- $P(\{d_i\}) \uparrow$ pour toutes les valeurs $d_1 \dots d_d$ où f n'est pas définie.

Une fonction est URM-calculable s'il existe un programme (URM) qui la calcule.

Soit P un programme exécuté dans une configuration initiale $\{d_1 \dots d_d, \vec{0}\}$ la fonction de \mathbb{N}^d dans \mathbb{N} calculée par P est :

$$f_P(d_1, \dots, d_d) = \begin{cases} r & \text{si } P(\{d_i\}) \downarrow r \\ \perp & \text{sinon} \end{cases}$$

Calculer la somme de deux entiers :

$$\mathcal{R} = \{x, y, \vec{0}\}$$

$$I_1 = J(3, 2, 25)$$

$$I_2 = S(1)$$

$$I_3 = S(3)$$

$$I_4 = J(1, 1, 1)$$

Écrire un programme calculant les fonctions de x :

$$x - 1 = \begin{cases} x - 1 & \text{si } x > 0 \\ 0 & \text{sinon} \end{cases}$$

$$x/2 = \begin{cases} x/2 & \text{si } x \text{ est pair} \\ \perp & \text{sinon} \end{cases}$$

$$1_{\mathbb{N}^*}(x) = \begin{cases} 0 & \text{si } x = 0 \\ 1 & \text{sinon} \end{cases}$$

$$1_{=}(x, y) = \begin{cases} 1 & \text{si } x = y \\ 0 & \text{sinon} \end{cases}$$

$$1_{>}(x, y) = \begin{cases} 0 & \text{si } x \leq y \\ 1 & \text{si } x > y \end{cases}$$

- plusieurs programmes pour calculer une fonction ?
- deux programmes calculent la même fonction ?

Prédicats et décidabilité

Un prédicat sur \mathbb{N}^k est une relation C sur \mathbb{N}^k . On lui associe sa fonction caractéristique :

$$f_C(x_1, \dots, x_k) = \begin{cases} 1 & \text{si } C(x_1, \dots, x_k) \\ 0 & \text{sinon} \end{cases}$$

C est dit décidable si f_C est calculable comme fonction de \mathbb{N}^k dans \mathbb{N}

Énumération

Soit E un ensemble, E est dit dénombrable s'il existe une injection i de E dans \mathbb{N} . E est dit fini si $i(E)$ l'image de E par i est finie.

\mathbb{N}^d est dénombrable :

- soient p_1, \dots, p_d des nombres premiers distincts
- On pose :

$$f(n_1, \dots, n_d) = \left(\prod_{i=1}^d p_i^{n_i} \right)$$

est injective mais pas surjective.

Une énumération ou un codage de E établit une bijection effective (calculable) entre E et \mathbb{N} .

Énumération de \mathbb{N}^2 :

On pose :

$$f(n_1, n_2) = 2^{n_1} (2n_2 + 1) - 1$$

Soit $n \in \mathbb{N}$, $f^{-1}(n)$ se calcule en :

- calculant n_1 en extrayant la plus grande puissance de 2 dans $n + 1$
- il reste $2n_2 + 1$ et on calcule n_2 en enlevant 1 et en divisant par 2.

On peut étendre la notion de calculabilité aux ensembles énumérables :

- E_1 et E_2 deux ensembles énumérés par c_1 et c_2
- f de E_1 dans E_2 est calculable si $c_2 \circ f \circ c_1^{-1}$ est calculable de \mathbb{N} dans \mathbb{N} .

Donner un codage \mathbb{Z} , montrer que la fonction f de \mathbb{Z} dans \mathbb{Z} définie par $f(p) = p - 1$ est calculable.

Caractérisation des fonctions calculables

- La fonction constante : $0_c(x) = 0$,

$$I_1 = J(1, 1, 2)$$

- la fonction successeur : $1_+(x) = x + 1$

$$I_1 = S(1)$$

- les projections, $U_i^{(n)}$ la i -ème projection de n variables :

$$U_i^{(n)}(x_1, \dots, x_n) = x_i$$

$$I_1 = T(i, 1)$$

sont des fonctions calculables.

Normalisation

Un programme (URM) $P = \{I_1, \dots, I_p\}$ est en forme normale s'il ne contient aucune instruction de saut à vers une instruction I_n avec $n > p + 1$.

Tout programme P est équivalent à un programme P^* en forme normale. Ils calculent la même fonction :

$$\forall d_1, \dots, d_n, r, P(\{d_i\}) \downarrow r \iff P^*(\{d_i\}) \downarrow r$$

On ne considère que des programmes normalisés.

Concaténation de programmes

$P = \{I_1, \dots, I_p\}$, $Q = \{I_1, \dots, I_q\}$ de longueurs respective p et q .

On construit

$$I_1, \dots, I_p, I_{p+1}, \dots, I_{p+q}$$

où I_{p+i} est la i -ème instruction de Q dans lequel on a remplacé les instructions $J(r_1, r_2, n)$ de saut à la n -ème instruction par $J(r_1, r_2, p + n)$.

Composition

Pour un programme P on note $\rho(P)$ l'indice d'un registre tel que aucun des registres $R_{\rho(P)+n}$ pour $n > 0$ n'est modifié par P .

Pour appeler P pendant un calcul il faut :

- préparer les données pour P ,
- calculer le résultat de P

sans que l'exécution de P n'interfère avec le reste.

Soit P calculant une fonction $f(x_1, \dots, x_k)$ on veut placer le résultat de $f(x_1, \dots, x_k)$ dans le registre R_t sachant que les valeurs x_i se trouvent dans les registres R_{t_i} .

On construit le translaté de P

$$P [t_1, \dots, t_k, \rightarrow t]$$

obtenu en :

- transférant le contenu de R_{t_1}, \dots, R_{t_k} dans R_1, \dots, R_k ,
 k instructions $T(t_i, i)$ ($i = 1 \dots k$).
- initialisant à 0 les registres $R_{k+1}, \dots, R_{\rho(P)}$,
 $\rho(P) - k$ instructions $Z(i)$ ($i = k+1 \dots \rho(P)$).
- exécutant P ,
les instructions de P .
- transférant R_1 dans R_t ,
1 instruction $T(1, t)$.

Substitution

Soit f de \mathbb{N}^n dans \mathbb{N} et $(g_i)_{i=1}^n$ n fonctions de \mathbb{N}^k dans \mathbb{N} , la composée (générale) h de f par les g_i est

$$h(x_1, \dots, x_k) = f(g_1((x_1, \dots, x_k)), \dots, g_n((x_1, \dots, x_k)))$$

si les g_i sont totales et si f est définie aux points de $(g_1(\mathbb{N}^k), \dots, g_n(\mathbb{N}^k))$ alors h est totale.

Proposition : Si f et les g_i sont calculables, h est calculable.

On construit le programme H à partir des programmes F et G_i associés respectivement à f et g_i . On calcule successivement les $g_i(x)$ puis on utilise F pour calculer $f(g_1(x), \dots, g_k(x))$:

- soit $N = \max(n, k, \rho(F), \rho(G_i))$
- recopier la donnée x dans R_{N+1}, \dots, R_{N+k}
- stocker le résultat de l'exécution de $G_i(x)$ dans R_{N+k+i} :

$$G_i [N + 1, \dots, N + k, \rightarrow N + k + i]$$

- exécuter F avec les arguments en positions $N + k + i$:

$$F [N + k + 1, \dots, N + k + n, \rightarrow 1]$$

Exercice

Soit f une fonction calculable de k variables, soient $(x_{i_1}, \dots, x_{i_k})$ des variables de (x_1, \dots, x_n) .

Montrer que la fonction $g(x_1, \dots, x_n)$ donnée par $f(x_{i_1}, \dots, x_{i_k})$ est calculable.

Alors les fonctions h_i données par :

$$- h_1(x_1, x_2) = f(x_1, x_2)$$

$$- h_2(x) = f(x, x)$$

$$- h_3(x_1, x_2, x_3) = f(x_2, x_3)$$

sont calculables si f est calculable.

Réursion Primitive

Soit $x = (x_1, \dots, x_n) \in \mathbb{N}^n$, et $y \in \mathbb{N}$. Soient f une fonction de n variables et g une fonction de $n + 2$ variables. La fonction h de \mathbb{N}^{n+1} dans \mathbb{N} définie par récursion primitive s'écrit :

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

La fonction h est totale si f et g le sont.

Les fonctions primitives récursives sont stables par composition généralisée et par récursion primitive.

Si f et g sont calculables alors h est calculable . Soient F et G calculant f et g . On donne H calculant h .

Posons $m = \max(n + 2, \rho(F), \rho(G))$

Construction du programme

- la configuration initiale contient les arguments en R_1, R_n et R_{n+1} .
- on sauvegarde les registres R_1, \dots, R_n, R_{n+1} dans $R_{m+1}, \dots, R_{m+n+1}$.
- on utilise un indice k qui vaut 0 initialement que l'on place dans R_{m+n+2}
- on stocke $h(x, k)$ dans R_{n+m+3}
- initialement on calcule $f(x_1, \dots, x_n)$ par $F [1 \dots n, \rightarrow m + n + 3]$
- on teste si $k = y$,
- on incrémente k après avoir exécuté $G [m + 1, \dots, m + n, m + n + 1, m + n + 2, \rightarrow m + n + 3]$
- on revient au test $k = y$

Fonctions usuelles

les fonctions données par :

$x + y$, xy , x^y , $\text{sg}(x)$, $|x|$, $x!$, $\text{quo}(x, y)$, $\text{mod}(x, y)$, $\text{min}(x, y)$...

Sont calculables : elles s'expriment à l'aide du schéma de récursion primitive.

Si P et Q sont des prédicats décidables alors les combinaisons booléennes usuelles de P et Q sont décidables.

Somme et produit bornés

Soit $f(x, y)$ une fonction totale de $n + 1$ variables. La somme et le produit bornés notés respectivement $\sum_{z < y} f(x, z)$ et $\prod_{z < y} f(x, z)$ sont des fonctions des variables x et y définies par :

$$\sum_{z < y} f(x, z) = \begin{cases} 0 & \text{si } y = 0 \\ \sum_{z < (y-1)} f(x, z) + f(x, y-1) & \text{si } y > 0 \end{cases}$$

$$\prod_{z < y} f(x, z) = \begin{cases} 1 & \text{si } y = 0 \\ \prod_{z < (y-1)} f(x, z) \cdot f(x, y-1) & \text{si } y > 0 \end{cases}$$

Si f est totale et calculable alors $\sum_{z < y} f(x, z)$ et $\prod_{z < y} f(x, z)$ sont calculables

Démonstration

Notons $S(x, y)$ la somme bornée.

On calcule $S(x, y)$ en initialisant un registre pour le résultat r et un registre pour un compteur z à zéro :

- on incrémente le compteur
- s'il vaut y on a fini.
- on calcule $F(x, z)$ que l'on ajoute au résultat.
- on revient à incrémenter le compteur.

De même pour le produit borné $\prod_{z < (y-1)} f(x, z)$.

Minimisation bornée

Si $P(x, z)$ est un prédicat et y un entier, on note $\mu(z < y).P(x, z)$ le plus petit entier z plus petit que y vérifiant $P(x, z)$ s'il existe.

Soit $f(x, y)$ une fonction de $n + 1$ variables on définit par minimisation bornée la fonction g de $n + 1$ variables par :

$$g(x, y) = \begin{cases} \mu(z < y).f(x, z) & \text{si } \{z < y \mid f(x, z)\} \neq \emptyset \\ y & \text{sinon} \end{cases}$$

Si f est totale calculable alors $g(x, y)$ est calculable.

Démonstration

On calcule le produit borné

$$h(x, v) = \prod_{u \leq v} \text{sg}(f(x, u))$$

(où $\text{sg}(x)$ la fonction indicatrice de $x \neq 0$) pour u variant de 0 à v .

On s'arrête dès que ce produit est nul ou que v vaut u .

Le résultat est v

Conséquences

si $f(x, z)$ et $k(x, w)$ sont totales calculables de $n + 1$ variables, alors

$$\mu(z < k(x, w)).f(x, z) = 0$$

de $n + 1$ variable est calculable.

Si $R(x, z)$ est un prédicat de $n + 1$ variables décidable, alors les prédicats de n variables

- $\forall z < y.R(x, z)$
- $\exists z < y.R(x, z)$

sont décidables.

Exemples

- $D(x)$ le nombre de diviseurs de x (par convention $D(0) = 1$)
- $P_r(x)$ qui vaut 1 si et seulement si x est premier,,
- $p(x)$ le x -ème nombre premier ($p(0) = 0$)

Le fonction $d(x, y)$ qui donne l'exposant de $p(y)$ dans x est calculable si x et y sont non nuls. On complète en $d(x, 0) = 0$ et $d(0, y) = 0$.

Minimisation arbitraire

Étant donnée $f(x, y)$ une fonction (non nécessairement totale) de $n + 1$ variables. On veut définir

$$g(x) = \text{le plus petit } y \text{ tel que } f(x, y) = 0$$

On définit $\mu y.f(x, y) = 0$ par

- le plus petit y tel que
 - $f(x, z)$ est définie en tout point $z \leq y$ et
 - $f(x, y) = 0$
- s'il existe un y vérifiant ces conditions.
- indéfini sinon.

Calculabilité

Si $f(x, y)$ est calculable alors $\mu y. f(x, y) = 0$ est aussi calculable.

On consruit le programme qui calcule successivement $f(x, i)$ pour i partant de 0; si on trouve $f(x, i) = 0$ alors $y = i$.

On obtient bien un programme qui calcule $\mu y. f(x, y) = 0$ mais on n'a pas de moyen de savoir si on doit s'arrêter!

Si on peut calculer une borne pour y par une fonction calculable de x la fonction peut être définie par minimisation bornée.

Exemple : la fonction d'Ackerman est calculable sans être définissable par minimisation bornée.

Fonctions récursives

La classe des fonctions récursives est la plus petite famille de fonctions (partielles) contenant :

- la fonction constante 0
- la fonction successeur
- les projections

stable par les opérations de récursion et de minimisation.

On a montré que toutes les fonctions récursives sont calculables!

Thèse de Church

Les différents systèmes formels définissant une notion de « calculabilité » définissent la même classe de fonctions.

- Machines de Turing,
 - Fonction récursives,
 - Le lambda calcul
- sont équivalents

Énumération des fonctions calculables

Les instructions sont énumérables.

On définit β :

$$\beta(Z(n)) = 4(n - 1)$$

$$\beta(S(n)) = 4(n - 1) + 1$$

$$\beta(T(n, m)) = 4\phi(n - 1, m - 1) + 2$$

$$\beta(J(m, n, p)) = 4\psi(n - 1, m - 1, p - 1) + 3$$

$$\text{où } \phi(n, m) = 2^n(2m + 1) - 1$$

$$\text{et } \psi(n, m, p) = \phi(\phi(n, m), p)$$

Numérotation de Gödel!

Énumération des programmes

Si E est dénombrable, l'ensemble E^* des suites finies d'éléments de E est dénombrable.

$$E^* = \bigcup_{n \geq 1} E_n$$

Supposons que f_E soit une énumération de E . On construit une énumération de E^* .

Soit e_1, \dots, e_n et c_1, \dots, c_n les $f_E(e_i)$ des e_i . On code e_1, \dots, e_n par :

$$(2^{c_1} + 2^{c_1+c_2+1} + \dots + 2^{c_1+\dots+c_n+n-1}) - 1$$

Étant donné un entier N on retrouve c_1 en cherchant la plus petite puissance de 2 dans $N + 1$. On continue avec

$$\frac{N + 1 - 2^{c_1}}{2^{c_1+1}}$$

s'il n'est pas nul

Exemples

– Le numéro de Gödel du programme :

$$\left. \begin{array}{l} T(1, 3) \\ S(4) \\ Z(6) \end{array} \right\} \text{ est } 9007203549970431$$

– le 4127 ème programme est $\left\{ \begin{array}{l} S(2) \\ T(2, 1) \end{array} \right.$

– le code de $\left\{ \begin{array}{l} T(3, 4) \\ S(3) \\ Z(1) \end{array} \right.$

Des fonctions non calculables ?

L'ensemble des fonctions de \mathbb{N} dans \mathbb{N} n'est pas dénombrable alors que l'ensemble des programmes l'est !

On note

- P_n le n i-ème programme
- $\Phi_n^{(k)}$ la fonction que P_n calcule lorsqu'il est vu comme une fonction k variables.
- $W_n^{(k)}$ le domaine de définition de $\Phi_n^{(k)}$, c'est à dire

$$\{(x_1, \dots, x_k), P_n(\{x_i\}) \downarrow\}$$

On prendra souvent $k = 1$ et Φ_n est fonction d'un seul paramètre et W_n son domaine de définition.

Une fonction unaire non calculable

Soit Φ_n le n -ème programme. On définit :

$$f(n) = \begin{cases} \Phi_n(n) + 1 & \text{si } n \in W_n \\ 0 & \text{sinon} \end{cases}$$

On a construit une fonction totale unaire f qui diffère de $\Phi(n)$ au point n . Elle n'apparaît donc pas dans l'énumération des programmes et n'est donc pas calculable.