

CHOUN TONG LIEU

Point-fixe sur un ensemble restreint

RAIRO. Informatique théorique et applications, tome 20, n° 4 (1986),
p. 383-394

http://www.numdam.org/item?id=ITA_1986__20_4_383_0

© AFCET, 1986, tous droits réservés.

L'accès aux archives de la revue « RAIRO. Informatique théorique et applications » implique l'accord avec les conditions générales d'utilisation (<http://www.numdam.org/conditions>). Toute utilisation commerciale ou impression systématique est constitutive d'une infraction pénale. Toute copie ou impression de ce fichier doit contenir la présente mention de copyright.

NUMDAM

Article numérisé dans le cadre du programme
Numérisation de documents anciens mathématiques

<http://www.numdam.org/>

POINT-FIXE SUR UN ENSEMBLE RESTREINT (*)

par Choun Tong LIEU ⁽¹⁾

Communiqué par J. E. PIN

Résumé. – Dans cet article, nous présentons une preuve constructive de l'existence d'opérateurs de point-fixes normalisateurs sur des sous-classes de l'ensemble des λ -termes. Le domaine d'application de ce résultat couvre en particulier les objets couramment manipulés en Programmation Fonctionnelle.

Abstract. – This article presents a constructive proof of the existence of normalizing fixed-point operators restricted to λ -terms subclasses. Domains where fixed-point operators can be defined cover usual areas of Functional Programming applications.

INTRODUCTION

L'étude des point-fixeurs, appelés aussi opérateurs paradoxaux, présente toujours un grand intérêt aussi bien en Lambda-Calcul qu'en Logique Combinatoire. Ces opérateurs possèdent, comme leurs noms l'indiquent, une propriété de point-fixe qui est la suivante : $(YF) := (F(YF))$ où F est un terme quelconque et où « $:=$ » est le signe d'égalité faible.

Citons Yc que Curry [6] appelle le Combinateur Paradoxal et Yt de Turing [9] définis de la manière suivante :

$$Yc = \lambda f. (\lambda x. (f (x x))) \lambda x. (f (x x))$$

$$Yt = \lambda x f. (f (x x f)) \lambda x f. (f (x x f)).$$

Mais ces deux point-fixeurs, appliqués à un terme F quelconque, ne possèdent pas en général de forme normale. Cela pose de sérieux problèmes lors de l'implémentation des systèmes se réclamant du Lambda-Calcul ou de la

(*) Reçu juin 1985, révisé mai 1986.

(1) L.I.T.P., Université Paris-VI, Aile 45-55, 2^e étage, porte 9, 4, place Jussieu, 75230 Paris Cedex 05.

Logique Combinatoire, car l'existence de forme normale y est, en général, la condition première de la terminaison d'un processus de calcul.

Pour répondre à ce problème, certains auteurs proposent de choisir des types bien particuliers de réduction permettant ainsi à (YF) d'être implémentable.

En Logique Combinatoire, Bellot [2] a proposé l'utilisation d'un point-fixeur extentionnel « Yb » construit en utilisant un algorithme particulier d'abstraction. Le terme « $(Yb F)$ » a une forme normale pour la réduction faible. Malheureusement, il n'a pas d'équivalent en Lambda-Calcul (Mezghiche [8]).

Dans un travail précédent, nous avons proposé d'y construire directement et « a la LISP » certaines fonctions récursives ayant toutes une forme normale, ce qui permet de suggérer l'utilisation possible d'un « point-fixeur » pour notre ensemble de listes et de nombres [7].

Afin de conserver les procédés usuels d'abstraction et de réduction, nous proposons ici de nous limiter à des ensembles restreints du Lambda-Calcul, en particulier à un système n'incluant que des listes et des nombres.

Dans cet article, nous construisons un « point-fixeur » unique, noté « Yl », travaillant sur ces ensembles. C'est-à-dire que pour tout élément p appartenant à ces deux ensembles, on a :

- $Yl F p := F (Yl F) p$;
- si F a une forme normale, alors $(Yl F)$ a aussi une forme normale.

PRÉLIMINAIRES ET NOTATIONS

Dans cet article, nous utilisons uniquement le Lambda-Calcul muni de la β -réduction [1] dénotée par « $:=$ » et de sa notion de forme normale.

L'égalité dénotée par « $:=$ » est déduite de cette réduction.

Le symbole « \equiv » dénote l'égalité syntactique.

Le symbole « Λ » représente l'ensemble des termes du Lambda-Calcul.

1. DÉFINITION : Soit $Q \subset \Lambda$, on dit que Q est uniformément soluble si :

$$\exists M_1, \dots, M_n \in \Lambda / \forall q \in Q, \quad q M_1 \dots M_n := \lambda x. (x).$$

(les M_1, \dots, M_n sont appelés solvants de Q).

2. THÉORÈME : Soit $Q \subset \Lambda$, avec Q uniformément soluble :

(a) $\exists Y_Q \in \Lambda$ tel que $\forall F \in \Lambda, \forall q \in Q, Y_Q F q := F(Y_Q F) q$;

(b) si F et les M_i ont des formes normales, alors $(Y_Q F)$ a aussi une forme normale.

Preuve : Prenons x, y et z n'appartenant pas aux M_1, \dots, M_n et définissons :

$$\alpha \equiv \lambda xy. (y (x x y));$$

$$\tau \equiv \lambda xyz. (z M_1 \dots M_n \alpha x y z)$$

où les M_1, \dots, M_n sont des solvants de Q .

Prenons $Y_Q \equiv \tau \tau$.

On a : $\tau \tau := \lambda yz. (z M_1 \dots M_n \alpha \tau y z)$:

(a) $\forall q \in Q, \forall F \in \Lambda$, nous avons :

$$\begin{aligned} Y_Q F q &:= q M_1 \dots M_n \alpha \tau F q \\ &:= \alpha \tau F q \\ &:= F(\tau \tau F) q \\ &:= F(Y_Q F) q; \end{aligned}$$

(b) Si F et les M_1, \dots, M_n ont des formes normales, on a :

$$Y_Q F := \lambda z. (z M_1 \dots M_n \alpha \tau F z).$$

On en déduit que $(Y_Q F)$ a aussi une forme normale.

On trouvera chez Berarducci [3] un résultat analogue : l'énoncé est ici précisé, afin de tenir compte des contraintes sur les M_1, \dots, M_n pour la partie (b) du théorème.

Nous nous intéressons maintenant à des ensembles uniformément solubles particuliers permettant de modéliser des listes et des nombres. Nous y construisons des solvants, et en appliquant le théorème, nous donnons la λ -expression correspondant à « Y_Q ».

3. DÉFINITION : $\forall m \in \mathbb{N}, \forall n \in \mathbb{N}^*$,

Soit $Q_{m,n}$ l'ensemble défini par :

$$\begin{aligned} - Q_{m,n} &= \{\lambda x_1 \dots x_k. (x_j T_1 \dots T_p), \\ &\text{avec } 1 \leq k \leq n, 0 \leq p \leq m \text{ et } 1 \leq j \leq k \text{ et } T_1, \dots, T_p \in \Lambda\}. \end{aligned}$$

Il est important de remarquer que chaque élément de $Q_{m,n}$ est en forme normale de tête en raison de la variable x_j .

Soient K_m et I des termes définis par :

- $K_m \equiv \lambda x_1 \dots x_{m+1}. (x_{m+1})$
- $I \equiv \lambda x. (x)$.

4. PROPOSITION : (a) $\exists Y_{m,n} \in \Lambda / \forall F \in \Lambda, \forall q \in Q_{m,n} Y_{m,n} F q := F(Y_{m,n} F) q$;
 (b) si F a une forme normale, alors $(Y_{m,n} F)$ a aussi une forme normale.

Preuve : (obtenue par l'application du théorème 2.)

Nous nous proposons ici de construire les solvants de $Q_{m,n}$ et ensuite $Y_{m,n}$.

Pour tout élément q de $Q_{m,n}$ considérons le terme $(q K_m \dots K_m I \dots I)$ où K_m apparaît n fois et I m fois.

Nous savons que q est de la forme $\lambda x_1 \dots x_k. (x_j T_1 \dots T_p)$ où k, j et p sont bornés d'après la définition de $Q_{m,n}$.

Nous avons :

$q K_m \dots K_m I \dots I := K_m R_1 \dots R_p K_m \dots (n-k) \text{ fois} \dots K_m I \dots (m) \text{ fois} \dots I$
 où $\forall u \in [1 \dots p], R_u \equiv [K_m/x_1, \dots, K_m/x_k] T_u$.

si $\underline{(n-k)} \leq \underline{(m-j)}$:

$$q K_m \dots K_m I \dots I := I \dots (j+n-k+1) \text{ fois} \dots I \\ := I$$

si $\underline{(n-k)} > \underline{(m-j)}$:

$q K_m \dots K_m I \dots I := K_m \dots (n-k-m+j) \text{ fois} \dots K_m I \dots (m) \text{ fois} \dots I$
 $:= K_m \dots ((n-k-m+j) \text{ modulo } m) \text{ fois} \dots K_m I \dots (m) \text{ fois} \dots I$

– si $(n-k-m+j)$ est divisible par $m+1$, alors on a :

$$q K_m \dots K_m I \dots I := I \dots (m+1) \text{ fois} \dots I \\ := I$$

– si $(n-k-m+j)$ n'est pas divisible par $m+1$ et soit r le reste de cette division, on a :

$$q K_m \dots K_m I \dots I := I \dots (r) \text{ fois} \dots I \\ := I$$

Construction de $Y_{m,n}$

$$\alpha \equiv \lambda xy. (y (x x y))$$

$$\tau_{m,n} \equiv \lambda xyz. (z K_m \dots K_m I \dots I \alpha x y z)$$

où K_m et I apparaissent respectivement n et m fois.

Et prenons : $Y_{m,n} \equiv \tau_{m,n} \tau_{m,n}$

Nous avons :

$$\begin{aligned}
 (a) \quad Y_{m,n} &:= \lambda yz. (z K_m \dots K_m I \dots I \alpha \tau_{m,n} y z) \\
 Y_{m,n} F q & \\
 &:= q K_m \dots K_m I \dots I \alpha \tau_{m,n} F q \\
 &:= \alpha \tau_{m,n} F q \\
 &:= F(\tau_{m,n} \tau_{m,n} F) q \\
 &:= F(Y_{m,n} F) q
 \end{aligned}$$

$$(b) \quad Y_{m,n} F := \lambda z. (z K_m \dots K_m I \dots I \alpha \tau_{m,n} F z).$$

On en déduit que si F a une forme normale, alors $(Y_{m,n} F)$ a aussi une forme normale.

5. APPLICATION AUX LISTES ET AUX NOMBRES

(1) DÉFINITION : L'ensemble des listes L est modélisé de la manière suivante :

- $NIL \equiv \lambda xy. (y)$, $NIL \in L$ (pour la liste vide);
- soient $q \in \Lambda$ et $l \in L$ (avec $x \notin q$ et $x \notin l$), alors $\lambda x. (x q l) \in L$ (présentée sous forme de doublet).

Nous remarquons que $\lambda x. (x q l)$ peut être construit à partir des termes D , q et l où D est défini par $\lambda xyz. (z x y)$ et utilisé comme combinateur du couple abstrait par Boehm [4].

(2) DÉFINITION : L'ensemble numérique N est défini de la manière suivante :

- $\underline{0} \equiv \lambda xy. (y)$, $\underline{0} \in N$;
- soit $\underline{n} \in N$, alors $\underline{n+1} \equiv \lambda xy. (x(\underline{n} x y))$, $\underline{n+1} \in N$.

C'est le classique système d'énumération de Church [5].

Pour manipuler ces objets, nous donnons en Annexe de cet article les définitions de quelques fonctions primitives.

(3) PROPOSITION : (a) $\exists Y_{2,2} \in \Lambda / \forall F \in \Lambda, \forall q \in \{L \cup N\}$,

$$Y_{2,2} F q := F(Y_{2,2} F) q.$$

(b) si F a une forme normale, alors $(Y_{2,2} F)$ a aussi une forme normale.

Nous savons que l'union de ces deux ensembles est incluse dans

$$Q_{2,2} = \{ \lambda x_1 \dots x_k. (x_j T_1 \dots T_p), \text{ avec } 1 \leq k \leq 2, 0 \leq p \leq 2 \text{ et } 1 \leq j \leq k \}.$$

En effet, le lecteur remarquera que tous les éléments appartenant à L ou à N appartiennent aussi à $Q_{2,2}$.

Nous avons donc $Y_{2,2}$ comme point-fixeur sur L et N par application de la proposition 4.

(4) Construction $Y_{2,2}$

Posons :

$$\begin{aligned} K_2 &\equiv \lambda x y z. (z) \\ \alpha &\equiv \lambda x y. (y (x x y)) \\ \tau_{2,2} &\equiv \lambda x y z. (z K_2 K_2 I I \alpha x y z) \\ Y_{2,2} &\equiv \tau_{2,2} \tau_{2,2} \end{aligned}$$

On a :

$$\begin{aligned} Y_{2,2} F &:= \lambda z. (z K_2 K_2 I I \alpha \tau_{2,2} F z) \\ Y_{2,2} F \lambda x y. (y) &:= \lambda x y. (y) K_2 K_2 I I \alpha \tau_{2,2} F \lambda x y. (y) \\ &:= K_2 I I \alpha \tau_{2,2} F \lambda x y. (y) \\ &:= \lambda z. (z) \alpha \tau_{2,2} F \lambda x y. (y) \\ &:= F(\tau_{2,2} \tau_{2,2} F) \lambda x y. (y) \\ &:= F(Y_{2,2} F) \lambda x y. (y). \end{aligned}$$

Ce qui veut dire que :

$$\begin{aligned} Y_{2,2} F \text{NIL} &:= F(Y_{2,2} F) \text{NIL} \\ Y_{2,2} F \underline{0} &:= F(Y_{2,2} F) \underline{0} \\ Y_{2,2} F \lambda x. (x p l) &:= \lambda x. (x p l) K_2 K_2 I I \alpha \tau_{2,2} F \lambda x. (x p l) \\ &:= K_2 I I \alpha \tau_{2,2} F \lambda x. (x p l) \\ &:= \alpha \tau_{2,2} F \lambda x. (x p l) \\ &:= F(\tau_{2,2} \tau_{2,2} F) \lambda x. (x p l) \end{aligned}$$

$$\begin{aligned}
 & := F(Y_{2,2} F) \lambda x. (x p l) \\
 Y_{2,2} F \underline{n+1} & := \underline{n+1} K_2 K_2 I I \alpha \tau_{2,2} F \underline{n+1} \\
 & := K_2 (\underline{n} K_2 K_2) I I \alpha \tau_{2,2} F \underline{n+1} \\
 & := I \alpha \tau_{2,2} F \underline{n+1} \\
 & := \alpha \tau_{2,2} F \underline{n+1} \\
 & := F(\tau_{2,2} \tau_{2,2} F) \underline{n+1} \quad :=: F(Y_{2,2} F) \underline{n+1}.
 \end{aligned}$$

6. EXEMPLES D'APPLICATION A LA PROGRAMMATION FONCTIONNELLE

Nous allons donner deux exemples de son utilisation avec des fonctions définies récursivement (« a la LISP ») :

$$\begin{aligned}
 \text{APPEND} & =_{\text{Def}} \lambda x y. ((\text{NULL } x) \\
 & \quad y \\
 & \quad (\text{CONS}(\text{CAR } x) (\text{APPEND} (\text{CDR } x) y))).
 \end{aligned}$$

Il est aisé de démontrer que la fonction « APPEND » fait la concaténation de deux listes.

Grâce à la fonctionnelle suivante :

$$\begin{aligned}
 A & \equiv \lambda f x y. ((\text{NULL } x) \\
 & \quad y \\
 & \quad (\text{CONS}(\text{CAR } x) (f (\text{CDR } x) y))),
 \end{aligned}$$

définissons :

$$\text{APPEN} \equiv (Y_{2,2} A),$$

ce qui donne :

$$\text{APPEN} := \lambda z. (z K_2 K_2 I I \alpha \tau_{2,2} A z).$$

Comme A a une forme normale, on en déduit que APPEN a aussi une forme normale.

Et on a :

$$\begin{aligned}
 (\text{APPEN NIL } p) & := (\text{NIL } K_2 K_2 I I \alpha \tau_{2,2} A \text{ NIL } p) \\
 & := (\alpha \tau_{2,2} A \text{ NIL } p)
 \end{aligned}$$

$$\begin{aligned}
& := (A (\tau_{2,2} \tau_{2,2} A) \text{NIL } p) \\
& := p \\
(\text{APPEN } \lambda x. (x \ q \ l) \ p) & := (\lambda x. (x \ q \ l) \ K_2 \ K_2 \ I \ I \\
& \quad \alpha \tau_{2,2} \ A \ \lambda x. (x \ q \ l) \ p) \\
& := (\alpha \tau_{2,2} \ A \ \lambda x. (x \ q \ l) \ p) \\
& := (A (\tau_{2,2} \tau_{2,2} \ A) \ \lambda x. (x \ q \ l) \ p) \\
& := (\text{CONS } q \ ((\tau_{2,2} \tau_{2,2} \ A) \ l \ p)) \\
& := : (\text{CONS } q \ (\text{APPEN } l \ p)) \\
\text{FACTORIELLE} & =_{\text{Def}} \lambda x. ((\text{ZERO } x) \\
& \quad \underline{1} \\
& \quad (* \ x \ (\text{FACTORIELLE} \ (\text{PRED } x))))).
\end{aligned}$$

Grâce à la fonctionnelle suivante :

$$\begin{aligned}
F & \equiv \lambda f x. ((\text{ZERO } x) \\
& \quad \underline{1} \\
& \quad (* \ x \ (f \ (\text{PRED } x))))),
\end{aligned}$$

définissons :

$$\text{FACT} \equiv (Y_{2,2} F).$$

Ce qui donne :

$$\text{FACT} := \lambda z. (z \ K_2 \ K_2 \ I \ I \ \alpha \tau_{2,2} \ F \ z).$$

Comme F a une forme normale, on en déduit que FACT a aussi une forme normale (d'après 5.3).

Donc :

$$\begin{aligned}
(\text{FACT } \underline{0}) & := (\underline{0} \ K_2 \ K_2 \ I \ I \ \alpha \tau_{2,2} \ F \ \underline{0}) \\
& := (\alpha \tau_{2,2} \ F \ \underline{0}) \\
& := (F (\tau_{2,2} \tau_{2,2} \ F) \ \underline{0}) \\
& := \underline{1} \\
(\text{FACT } \underline{n+1}) & := (\underline{n+1} \ K_2 \ K_2 \ I \ I \ \alpha \tau_{2,2} \ F \ \underline{n+1}) \\
& := (\alpha \tau_{2,2} \ F \ \underline{n+1})
\end{aligned}$$

$$\begin{aligned} &:= (F (\tau_{2,2} \tau_{2,2} F) \underline{n+1}) \\ &:= (* \underline{n+1} ((\tau_{2,2} \tau_{2,2} F) \underline{n})) \\ &:= (* \underline{n+1} (\text{FACT} \underline{n})) \end{aligned}$$

7. THÉORÈME : Soit Q un ensemble uniformément soluble avec des solvants normalisables et pour tout q appartenant à Q , il existe un terme Y_q construit à partir de Y_Q tel que :

$$\forall F \in \Lambda, \quad (Y_q F) := (F (Y_q F)).$$

Preuve : Définissons : $Y_q \equiv \lambda f. (Y_Q \lambda xy. (f (x y)) q)$

Pour tout F , on a alors :

$$\begin{aligned} (Y_q F) &:= (Y_Q \lambda xy. (F (x y)) q) \\ &:= (\lambda xy. (F (x y)) (Y_Q \lambda xy. (F (x y)) q)) \\ &:= (F (Y_Q \lambda xy. (F (x y)) q)) \\ &:= (F (Y_q F)). \end{aligned}$$

Il est important de remarquer que pour un F quelconque, $(Y_q F)$ n'a pas en général de forme normale.

Cette proposition nous permet de passer de notre point-fixeur dans un ensemble restreint à un point-fixeur général.

ANNEXE

Modèle des fonctions primitives sur N et L en Lambda Calcul.

| | |
|---------------------|---|
| I | $\equiv \lambda x. (x).$ |
| $0, \text{NIL}$ | $\equiv \lambda xy. (y).$ |
| $\bar{1}$ | $\equiv \lambda xy. (x y).$ |
| $\bar{\text{VRAI}}$ | $\equiv \lambda xy. (x).$ |
| FAUX | $\equiv \lambda xy. (y).$ |
| NON | $\equiv \lambda x. (x \text{ FAUX VRAI})$ négation. |
| CONS | $\equiv \lambda xyz. (z x y)$ construction d'une liste. |
| NULL | $\equiv \lambda x. (x \lambda xyzuv. (v) \lambda xy. (x))$ test d'égalité à NIL pour L . |
| car | $\equiv \lambda x. (x \lambda yz. (y)).$ |
| crd | $\equiv \lambda x. (x \lambda yz. (z)).$ |
| CAR | $\equiv \lambda x. ((\text{NULL } x) x (\text{car } x))$ donne le premier élément d'une liste. |
| CDR | $\equiv \lambda x. ((\text{NULL } x) x (\text{cdr } x))$ donne la liste sans son premier élément. |
| NZERO | $\equiv \lambda xy. (x \lambda z. (y))$ test de différence avec 0. |
| ZERO | $\equiv \lambda x. (\text{NON } (\text{NZERO } x))$ test d'égalité avec 0. |
| $+$ | $\equiv \lambda xyzv. (x z (y z v))$ l'addition pour N . |
| $*$ | $\equiv \lambda xyz. (x (y z))$ la multiplication pour N . |
| SUCC | $\equiv \lambda x. (+ \bar{1} x)$ la fonction successeur pour N . |
| PPRED | $\equiv \lambda xy. ((\text{ZERO } y) (x \bar{1}) (+ \bar{1} (x \bar{1}))).$ |
| PRED | $\equiv \lambda x. (x \text{ PPRED } \lambda xyz. (z) \bar{\lambda xy. (y)})$ prédécesseur pour N . |

Preuves du modèle proposé

$$(\text{NON FAUX}) := (\lambda xy. (y) \text{ FAUX VRAI})$$

$$:= \text{VRAI}$$

$$(\text{NON VRAI}) := (\lambda xy. (x) \text{ FAUX VRAI})$$

$$:= \text{FAUX}$$

$$(\text{CONS } ql) := \lambda z. (z ql)$$

$$(\text{NULL NIL}) := (\text{NIL } \lambda xyzuv. (v) \lambda xy. (x))$$

$$:= \lambda xy. (x)$$

$$:= \text{VRAI}$$

$$(\text{NULL } \lambda x. (x ql)) := (\lambda x. (x ql) \lambda xyzuv. (v) \lambda xy. (x))$$

$$:= (\lambda xyzuv. (v) ql \lambda xy. (x))$$

$$:= \lambda uv. (v)$$

$$:= \text{FAUX}$$

$$(\text{car } \lambda x. (x q l)) := (\lambda (x q l). \lambda yz. (y))$$

$$:= q$$

$$(\text{cdr } \lambda x. (x q l)) := (\lambda x. (x q l) \lambda yz. (z))$$

$$:= l$$

$$(\text{CAR NIL}) := (\text{NULL NIL NIL } -)$$

$$:= \text{NIL}$$

$$(\text{CAR } \lambda x. (x q l)) := (\text{NULL } \lambda x. (x q l) - (\text{car } \lambda x. (x q l)))$$

$$:= (\text{FAUX } - q)$$

$$:= q$$

$$(\text{CDR NIL}) := (\text{NULL NIL NIL } -)$$

$$:= \text{NIL}$$

$$(\text{CDR } \lambda x. (x q l)) := (\text{NULL } \lambda x. (x q l) - (\text{cdr } \lambda x. (x q l)))$$

$$:= (\text{FAUX } - l)$$

$$:= l$$

$$(\text{NZERO } \underline{0}) := \lambda y. (\underline{0} \lambda z. (y))$$

$$:= \lambda y. (\lambda v. (v))$$

$$:= \text{FAUX}$$

$$(\text{NZERO } \underline{n+1}) := \lambda y. (\underline{n+1} \lambda z. (y))$$

$$:= \lambda y. (\lambda v. (\lambda z. (y) (\underline{n} \lambda z. (y) v))$$

$$:= \lambda y. (\lambda v. (y))$$

$$:= \text{VRAI}$$

$$(+ \underline{m} \underline{n}) := \lambda zv. (\underline{m} z (\underline{n} z v))$$

$$:= \lambda zv. (z (z (\dots (z v) \dots)))$$

$$:= \underline{m+n}.$$

$$(* \underline{m} \underline{n}) := \lambda z. (\underline{m} (\underline{n} z))$$

$$:= \lambda z. (\lambda y. (\underline{n} z (\underline{n} z (\dots (\underline{n} z y) \dots))))$$

$$:= \lambda z. (\lambda y. (z (z (\dots (z y) \dots))))$$

$$\begin{aligned}
 & := \underline{m * n} \\
 (\text{SUCC } \underline{m}) & := (+ \underline{1} \underline{m}) \\
 & := \underline{1 + m} \\
 (\text{PRED } \underline{0}) & := (\underline{0} \text{PPRED } \lambda xyz. (z) \lambda xy. (y)) \\
 & := (\lambda xyz. (z) -) \\
 & := \lambda yz. (z) \\
 & := \underline{0} \\
 (\text{PRED } \underline{m + 1}) & := (\underline{m + 1} \text{PPRED } \lambda xyz. (z) \lambda xy. (y)) \\
 & := (\text{PPRED} (\text{PPRED} \dots \\
 & \quad (\text{PPRED } \lambda xyz. (z)) \dots) \lambda xy. (y)) \\
 & := (\text{PPRED} (\text{PPRED} \dots \\
 & \quad (\text{PPRED } \lambda xyz. (z)) \dots) \underline{1}) \text{ (avec } m \text{ fois PPRED)} \\
 & := (+ \underline{1} (\text{PPRED} \dots \\
 & \quad (\text{PPRED } \lambda xyz. (z)) \dots \underline{1})) \text{ } m - 1 \text{ fois PPRED} \\
 & := (+ \underline{1} (+ \underline{1} (+ \dots (+ \underline{1} \underline{0}) \dots)) \text{ (avec } m \text{ fois } \underline{1})) \\
 & := \underline{m}
 \end{aligned}$$

BIBLIOGRAPHIE

1. H. P. BARENDREGT, *The Lambda Calculus, Its Syntax and Semantics*. Studies in Logic and Foundations of Mathematics, vol. 103, North-Holland Publishing Company.
2. P. BELLOT, *Propriétés logico-combinatoires des Systèmes de programmation sans variables*, Thèse de 3^e cycle, Paris-VI, L.I.T.P., 84-30, 1984.
3. A. BERARDUCCI, *Programmazione Funzionale e Rappresentabilità in alcuni Sistemi di Logica Combinatoria*, Tesi di laurea in Matematica, Roma, 1983.
4. C. BOEHM, *Modèle arithmétique de la Logique Combinatoire*, Lambda-Calcul et sémantique formelle des langages de programmation, L.I.T.P.-E.N.S.T.A., 1979, p. 97, 108.
5. A. CHURCH, *The Calculi of Lambda conversion*, Annals of Math. Studies, 6, Princeton University Press, 1941.
6. H. B. CURRY, *Combinatory Logic*, vol. I, North Holland, Amsterdam, 1958.
7. C. T. LIEU, *Étude d'une Extension Combinatoire des Entiers naturels. Application à la Programmation fonctionnelle*. Thèse de 3^e cycle, Paris-VI, L.I.T.P., 84-27, 1984.
8. M. MEZGHICHE, *Intertraduction entre le Lambda-Calcul et la Logique Combinatoire*, Thèse de 3^e cycle, Paris-VII, L.I.T.P., 83-25, 1983.
9. A. M. TURING, *The p-functions in λ-K-conversion*, J. Symbolic Logic, vol. 2, 1937, p. 164.