PUBLICATIONS DU DÉPARTEMENT DE MATHÉMATIQUES DE LYON

MICHEL CAYROL

Ensembles infinis en programmation

Publications du Département de Mathématiques de Lyon, 1985, fascicule 2B
« Compte rendu des journées infinitistes », , p. 45-56

<http://www.numdam.org/item?id=PDML_1985___2B_45_0>

ENSEMBLES INFINIS EN PROGRAMMATION

par Michel CAYROL

Introduction.

Nous allons présenter le langage PSIL.. Initialement, ce langage a été conçu pour permettre à l'utilisateur de manipuler des objets infinis. Alors qu'un étudiant de terminale concevra sans difficulté que la liste des nombres pairs s'obtienne en multipliant tous ses éléments par 2, un programmeur sera en général très embarassé pour écrire un algorithme produisant effectivement cette liste. Mettre $\mathbb{N}$ à la disposition de l'utilisateur - avec tout ce que cela implique - a constitué notre premier objectif. La pratique de notre langage nous a conduit à une autre conception de la programmation. Nous avons découvert qu'il est souvent plus efficace de calculer f{x} à partir de f{X} - où X est la liste de toutes les valeurs de x - plutôt que d'effectuer un calcul direct. Nous avons développé un type de calcul très simple, permettant à partir de la définition classique d'une fonction récursive de produire automatiquement -utilitaire ens- une équation dont les paramètres sont des listes (en général infinies). Il est aussi possible de programmer directement à partir des nouveaux concepts introduits, et ce en concevant directement les équations caractérisant les listes souhaitées. Le système d'équations obtenu constitue en lui-même un programme utilisable pour calculer les éléments de la liste. Les performances de ce programme sont en général bien supérieures à celles du programme initial (*chargé de calculer les valeurs de la fonction*). La démarche décrite peut facilement se généraliser à des tableaux infinis et produire ainsi des constructions de dimension deux ou plus. Ce programme ne peut toutefois pas être exécuté par un interprète classique. La plupart des contraintes qu'il doit respecter correspondent aux propriétés de certains dialectes de LISP [McCarthy] (*à liaison statique*). Nous les analyserons, les étendrons, et décrirons rapidement un modèle dont une version plus élaborée a été rédigée en MACLISP (Moon).

I. La programmation en intelligence artificielle :

Les langages utilisés en intelligence artificielle sont caractérisés par leur adéquation au traitement formel. Ce qui implique la manipulation de structures avec des outils appropriés tels que sélecteurs, constructeurs ..... Une liste peut être considérée comme une suite d'éléments accessibles individuellement au moyen des fonctions car, cadr, caddr, cadddr.....

On ne peut échapper à sa structure linéaire que par un artifice de représentation. Ainsi un tableau sera constitué d'une liste d'éléments, chacun d'entre eux correspondant par exemple à une ligne. Dans tous les cas, il n'est pas possible de manipuler de structure infinie. Si la construction est syntaxiquement acceptable, l'interprète entre dans un calcul de durée infinie comme le montre l'exemple suivant (rédigé en LISP).

```
(defun n ()                          (1)
     (e 0))

(defun e(x)
    (cons x (e (1+ x))))             (2)
```

La valeur de (n) résultera de l'évaluation de (e 0), laquelle évaluation entraine celle de (e 1), qui implique à son tour celle de (e 2) .....

PSIL accepte de traiter l'exemple précédent, et la valeur de (n) représente bien la liste de tous les entiers naturels.

II Une description informelle de PSIL

L'interprète PSIL est conçu autour de la classique boucle "read-eval-print". Il va donc lire une expression, l'évaluer avant de retourner la valeur résultant de cette évaluation. Mais ici, le mécanisme de l'évaluation sera plus complexe que dans un LISP classique. Disons schématiquement que c'est la phase finale d'impression qui sera chargée d'achever l'évaluation. Le mécanisme fin de cette évaluation est décrit dans "conception et réalisation d'un modèle pour le traitement d'objets infinis dénombrables" (CAYROL) nous ne le détaillerons pas ici. Nous ne présenterons aussi qu'un sous-ensemble de PSIL suffisant pour la justification de notre propos.

Les objets :

Essentiellement, les atomes et les listes.

Les atomes : les nombres et les symboles atomiques, un identificateur

étant constitué d'une suite finie de caractères ordinaires $\{a,b,-----z,A,B,----Z,0,1,---9\}$, le premier caractère n'étant pas un chiffre.

24,3.5 sont des nombres,

T12, factorielle sont des symboles atomiques. Les symboles atomiques autre que t et nil pourront être utilisés en tant que variables. Nous appellerons liste de variable toute suite finie de variables distinctes. Elle sera représentée par un objet PSIL.

Les listes : Une liste est une suite finie d'objets. Nous représenterons la suite $\{u_i\}$ de 3 éléments par$(u_1\ u_2\ u_3)$. Il existe deux fonctions appelées sélecteurs qui sont notées en LISP, car et cdr.

Elles sont définies par :

$$car\{f\} = f(1)$$
$$cdr\{f\} = g \text{ avec } g(x) = f(x+1)\ .$$

La fonction cons appelée constructeur a pour argument un objet et une liste. Elle se définit par :

$$cons\{u,f\} = g \quad \text{avec} \quad g\{1\} = u \quad \text{et} \quad g\{x\} = f\{x-1\} \quad \text{pour tout } x > 1.$$

Notons ici que toute liste non vide peut donc s'exprimer par le cons de son car et de son cdr. Nous noterons parfois $cons\{u,f\}$ par $(u \circ f)$.

III L'évaluation des objets :

3.1. le cadre de l'évaluation.

C'est un processus produisant à partir d'un objet <obj 1> et d'un "environnement" <env 1> un objet <obj 2> appelé valeur de <obj 1> par rapport à <env 1>. L'exposé de ce processus nécessite l'introduction d'éléments que nous nommerons environnement, lambda-expressions, fermetures et suspensions. Tous ces éléments seront représentés par des objets PSIL.

3.1.1. Les environnements.

Un environnement est une fonction définie dans l'ensemble des variables et prenant ses valeurs dans l'ensemble des objets. Il est possible de définir naturellement un nouvel environnement <env 2> à partir d'une liste <lv> de n variables ($n \geq 0$), une suite <lob> de n objets et un environnement initial <env 1> . La fonction nvenv se définit par :

$$\text{nvenv}\{\langle lv\rangle, \langle lob\rangle, \langle env1\rangle]\} = \langle env2\rangle$$

avec <env2>{<u>} = s'il existe un entier i>0 tel que <u> = $\langle lv\rangle_i$

alors $\langle lob\rangle_i$

sinon <env 1>{<u>} .

3.1.2. Les lambda-expressions :

Nous noterons

[*L*,<lv>,<objet >]

la lambda-expression définie par un couple constitué d'une liste de variables et d'un objet.

3.1.3. Les fermetures:

Nous noterons

[*F*,<lv>,<objet>,<env>]

La fermeture définie par un triplet constitué d'une liste de variables, d'un objet, et d'un environnement. Elle peut s'obtenir en conjugant une lambda-expression et un environnement.

3.1.4. Les suspensions :

Nous noterons

[*S*,<objet>,<env>]

La suspension définie par le couple constitué d'un objet et d'un environnement.

3.2. le fonctionnement de l'interprète.

Similaire à l'interprète LISP, l'interprète PSIL lit une expression, entreprend son évaluation, pour enfin imprimer la valeur résultante. L'évaluation se fera en deux phases :

1°) après la lecture,

2°) lors des accès (on appelle accès l'application des fonctions de sélection 1 et -1 dont le mécanisme est détaillé dans le 3.2.2.2.

3.2.1. Les atomes :

Les nombres ainsi que t et nil sont des constantes (c.à.d. conservés dans le processus d'évaluation).

Les atomes identifiant les primitives de PSIL, cons, 1+, 1-, +, -, *,/, inter, union, quote, if ..... sont aussi des constantes.

Les autres symboles atomiques (variables) sont évaluées par rapport à l'environnement courant. Ainsi [*V*,<objet1>,<ev>] = > <objet 2> signifie que l'évaluation de <objet1> par rapport à l'environnement <ev> produit la valeur <ev> {<objet1>} notée ici <objet2> , le premier membre dénote donc la valeur d'un objet par rapport à un environnement.

3.2.2. Les listes :

De manière générale, l'évaluation d'une liste correspond à l'application de la fonction dénotée par la valeur du premier élément de la liste. Le calcul de [*V*,(<obj1> <obj2> -----), <ev>] commence donc par celui de [*V*,<obj1>,<ev>] .

Nous allons passer en revue les principales fonctions de PSIL.

3.2.2.1. la fonction quote : La fonction quote permet de bloquer l'évaluation d'un objet, elle se caractérise par :

si [*V*,<objet1>,<ev>] = > quote

alors [*V*,(<objet1> <objet2>), <ev>] = > <objet2> .

En PSIL comme en LISP, (quote<objet>) s'écrit aussi '<objet>.

exemple : [*V*, '(a b 5 (toto est malade)), E] =>(a b 5 (toto est malade)).

3.2.2.2. Les fonctions arithmétiques :

si [*V*,<objet1>,<ev>] => 1 +
alors [*V*,(<objet1><objet2>),<ev>] => [*V*,<objet2>,<ev>] + 1.

si [*V*,<objet1>,<ev>] => 1 -
alors [*V*,(<objet1><objet2>),<ev>] => [*V*,<objet2>,<ev>]-1

si [*V*,<objet1>,<ev>] => +
alors [*V*,(<objet1><objet2><objet3>),<ev>]=>
[*V*,<objet2>,<ev>] + [*V*,<objet3>,<ev>]

si [*V*,<objet1>,<ev>] => -
alors [*V*,(<objet1><objet2><objet3>),<ev>] =>
[*V*,<objet2>,<ev>] - [*V*,<objet3>,<ev>]

si [*V*,<objet1>,<ev>] => *
alors [*V*,(<objet1><objet2><objet3>),<ev>] =>
[*V*,<objet2>,<ev>]* [*V*,<objet3>,<ev>]

si [*V*,<objet1>,<ev>] => /
alors [*V*,(<objet1><objet2><objet3>),<ev>] =>
[*V*,<objet2>,<ev>] / [*V*,<objet3>,<ev>]

.....
.....

3.2.2.2. Les sélecteur numériques : Nous utiliserons 1 et -1 en notation indicielle afin d'exprimer plus simplement la notion d'accès.

$(\langle u\rangle.\langle v\rangle)_1$ = *(sera égal par définition à)*

si<u> = [*S*,<obj>,<env>]
alors [*V*,<obj>,<env>]
sinon <u> .

de même

$(\langle u\rangle.\langle v\rangle)_{-1}$ =

si <v> = [*S*,<obj>,<env>]
alors [*V*,<obj>,<env>]
sinon <v>

$(\langle u\rangle.\langle v\rangle)_n$ = si n>1 alors $\{(\langle u\rangle.\langle v\rangle)_{-1}\}_{n-1}$

$(\langle u\rangle.\langle v\rangle)_n$ = si n<-1 alors $\{(\langle u\rangle.\langle v\rangle)_{-1}\}_{n+1}$

$(\langle u\rangle.\langle v\rangle)_0$ = (<u>.<v>).

L'évaluation s'exprime maintenant très simplement.

si [*V*,<objet1>,<ev>] =>n

alors [*V*,(<objet1><objet2>), <ev>] => $[*V*,\langle objet2\rangle,\langle env\rangle]_n$

Exemple : [*V*,((1+2), '(a b c)),<env>] => [*V*, '(a b c),<env>]$_3$ => $(a\ b\ c)_3$ =>

$\{(a\ b\ c)_{-1}\}_2$ => $(b\ c)_2$ => $\{(b\ c)_{-1}\}_1$ => $(c)_1$ => c.

3.2.2.3.cons.

si [*V*,<objet1>,<ev>] = cons
alors [*V*,(<objet1> <objet2> <objet3>), <ev>] =>
([*S*,<objet2>,<ev>]. [*S*,<objet3>,<ev>]).

3.2.2.4.if

si [*V*,<objet1>,<ev>] = if
alors [*V*,(<objet1> <objet2> <objet3>),<ev>] =>
si [*V*,<objet1>,<ev>] ≠ nil
alors [*V*,<objet2>,<ev>]
sinon [*V*,<objet3>,<ev>] .

3.2.2.5.lambda-expression

si <objet1> = [*L*,<lv>,<objet2>]
alors [*V*,<objet1>,<ev>] => [*F*,<lv>,<objet2>,<ev>] .

3.2.2.6.Fermeture.

Si [*V*,<objet1>,<ev1>] = [*F*,<lv>,<objet>,<ev2>]
alors [*V*,(<objet1><objet2>...<objetn>),<ev1>] =>
[*V*,<objet>,nvenv{<lv>,<lob>,<ev2>}]
ou <lob> est la liste des [*V*,<objet$_i$>,<ev1>] .

3.2.3 Un exemple :

Considérons dans l'environnement <ev> défini par :

<ev>{f} = [*F*,(x),(cons x(f(1+x))),<ev>],

le calcul de [*V*,(2(f 0)),<evb>]. Les règles du 3.2.3 seront référencées par leur numéro dans le paragraphe.

[*V*,(2(f0)),<evb>]=> [*V*,(f 0),<evb>]$_2$ => {[*V*,(f 0),<evb>]$_{-1}$}$_1$
application de la définition *(2)*.
[*V*,(f 0),<evb>]=> [*V*,(cons x(f(1+x))).<ev1>] *(6)*
avec <ev1> = nenv{(x),([*V*,0,<evb>]),<ev>} soit
([*S*,x,<ev1>].[*S*,(f(1+x)),<ev1>]) *(3)*. Nous en déduisons que
[*V*,(f 0),<evb>]$_{-1}$ => [*V*,(f(1+x)),<ev1>] => [*V*, (cons x (f(1+x))) .<ev2>] *(6)*
avec <ev2> = nvenv{(x),([*V*,(1+x),<ev1>],<ev>} = nvenv{(x),(1) <ev>}.
Donc {[*V*,(f 0),<evb>]$_{-1}$}$_1$ => *(3)* ([*S*,x,<ev2>].[*S*,(f(1+x)),<ev2>])$_{-1}$.
Il nous reste à calculer ([*S*,x,<ev2>].[*S*,(f(1+x)),<ev2>])$_1$, ce qui produit
[*V*,x,<ev2>] *(2)* => 1.

On peut démontrer facilement que [*V*,(n(f 0)),<evb>]= n-1 pour tout valeur de N* indépendamment de <evb>. C'est donc que (f 0)- via le processus d'évaluation - représente la liste de tous les entiers naturels.

IV La pratique du langage :

Pour des raisons pratiques de lisibilité et d'efficacité, PSIL intégre de nombreux outils qui n'ont pas été décrits dans le modèle du III.

Le symbole $ sera le générateur le plus utilisé d'objets "infinis". $<x> produira (=>) une liste "infinie" dont tous les éléments sont identiques à [*V*,<x>,<ev>] ou <ev> dénote l'environnement courant. Il existe aussi des outils permettant de donner une valeur à une variable dans l'environnement courant c'est-à-dire de faire évoluer l'environnement.

[*V*,(<=> x <u>), <ev>] (a)

=> x

mais dorénavant <ev> = nvenv{(x),(<v1>}, <evold>} ou <v1> = [*V*,<u>,<evold>] quand <evold> dénote l'environnement <ev> antérieur au calcul de (a).
Nous allons nous connecter à la console et suivre le déroulement d'une session PSIL. Le texte frappé par l'utilisateur sera dans ce type de caractère, Les réponses de l'interprète seront dans ce type de caractère, enfin *ce type de caractère* servira pour les explications. L'impression d'une liste "infinie" pose toujours quelques problèmes ... elle sera interrompue par l'envoi à la console du caractère "break" qui se traduira par l'affichage d'un message.

? : n

=(0 1 2 3 4 5 6 7 8 9 10 11 12 13 Break dans PSIL *(:n est une variable prédéfinie. Elle "contient" la liste de "tous" les entiers naturels).*

? $ 4

=(4444444444444444444444444 Break dans PSIL

?(<=> : rep5 $5) *(la variable rep5 contient une liste infinie de 5)*

=:rep5.

? ('(+ - * /)'(1 2 3 4)'(56 3 2 2))

= (57 - 1 6 2) *(Cette fois ci, la valeur v_1 du premier élément de la liste à évaluer n'est ni une fonction primitive du langage, ni une fermeture. PSIL considère donc qu'il doit appliquer en séquence chaque élément de v_1 aux éléments homologues des valeurs v_i des autres éléments de liste à évaluer. Ici on appliquera + à 1 et 56, - à 2 et 3, ...)*

? (<=> : pairs($* $2:n))

= : pairs *(contient la liste de tous les nombres pairs)*

? (<=> : mult3 ($* $3 : n))

=:mult3

? : mult3

=(0 3 6 9 12 15 18 21 Break dans PSIL

? (<=> = mult2-3 (inter : pairs : mult3))

=mult2-3

```
? : mult2-3
=(0 6 12 18 24 30 36 Break dans PSIL
(mieux la liste de tous les nombres premiers)
= (ds erat
  (%1 (: lst)
      (cons(1 : lst)
           (erat(diff(-1 : lst)
                      ($*(1 : lst)
                        (-2 : n)))))))
=erat  (ds définit une fermeture qui sera la valeur de erat. L'environnement
associé contient les valeurs de erat et de :n)
? (<=> : cribe(erat(-2 : n)))
=(2 3 5 7 11 13 17 19 23 29 31 37 Break dans PSIL
? (<=> : pascal($cons $ 1 (cons $0 (($$ + ($-1 : pascal) : pascal))))
=:pascal  (cette expression curieuse produit les valeurs du triangle de Pascal)
? (10 : pascal)
=(1 9 36 84 126 126 84 36 9 1 0 0 0 0 Break dans PSIL (ligne n° 10 du tableau)
? ($3 : pascal)
=(0 0 1 3 6 10 15 21 Break dans PSIL
? (<=> : fib(cons1 (cons1 ($ + :fib(-1 : fib)))))
=(1 1 2 3 5 8 13 21 34 55 89 Break dans PSIL
(L'expression de fib découle immédiatement - et automatiquement dans PSIL -
de la définition récursive de FIB. Toutefois elle supprime systématiquement
tous les appels redondants autorisant des calculs jusqu'alors impossibles
pratiquement. Maclisp peine pour (fib 40), PSIL évalue sans difficulté (FIB 2000))
? (<=> : biz(entrelace : n : biz)) (la fonction entrelace ne sera pas détaillée ici,
entrelacer ( 1 2 3 4 5 6 ...) et (a b c d e f ....) c'est produire (1 a 2 b 3 c
4 d ....)
=(0 0 1 0 2 1 3 0 4 2 5 1 6 3 7 0 8 4 9 2 10 5 11 1 12 Break dans PSIL
.......
```

Cette session est exemplaires dans la mesure où elle illustre les objectifs du programmeur PSIL. Le calcul de f{x} avec x prenant ses valeurs dans une liste <l> souvent moins avantageux que le calcul de $f\{<l>\}_{p\{x,<>\}}$ ou p{x,<l>} représente la place de x dans la suite <l>. Il suffira donc de définir l'équation -ou le

système- produisant f{<1>} , la sélection numérique "accédant" à l'élément souhaité. Il est possible d'automatiser le passage entre la définition de f{x} et celle de f{<1>} . Ainsi à partir de l'expression

```
(<=>(pascal(1+x)(1+y))
    ($+(pascal(1+x)y)
       (pascal x y)))
```

l'utilitaire ens est capable de réclamer les informations qui lui manquent - 1° colonne et 1° ligne du tableau - afin de produire et de valider

```
(<=>:pascal($cons$1(cons$0($$+($-1:pascal):pascal)))) .
```

En associant au développement $a_0+a_1x+a_2x^2+.....$, la liste $(a_0a_1a_2.....$, il est possible d'écrire en PSIL une fonction integ à deux arguments : la liste et une constante c (d'intégration) qui produit la liste :

$(c\ a_0\ a_1/2\ a_2/3\ a_3/4\ .....$

En reprenant notre session à la console

```
? (<=> : sin(integ 0 : cos))
=:sin
? (<=> : cos(integ 1 ($chs : sin)))
=:cos

=(0. 1.0 0.0-0.16666667 0.0 8.3333333e-3 0.0-1.984 Break dans PSIL
```

(ce sont les coefficients du développement en série du sinus à partir desquels PSIL calcule les lignes trigonométriques d'un angle)

V PSIL ET L'INTELLIGENCE ARTIFICIELLE :

Héritier de LISP, PSIL affectionne particulièrement ce domaine. Nous l'avons testé tant pour écrire des algorithmes de "pattern matching", que pour définir des espaces d'état. Ses possibilités pourraient encore être accrues en concevant directement l'interprête sans passer par sa rédaction en LISP. Actuellement il existe deux versions : un mini-PSIL qui "tourne" sur un micro-ordinateur (à base de Z80) et PSIL qui est écrit en MACLISP et qui est implanté sur le site du Centre Inter-Universitaire de Calcul de Toulouse. Nous étudions son implémentation sur le Macintosh dans la configuration à 512K car nous pensons que les vertus pédagogiques du langage ne doivent pas être négligées.

Bibliographie sommaire :

1 CAYROL M.

Conception et pratique d'un modèle pour le traitement d'objets infinis dénombrables , rapport interne U.P.S. (1984).

2 GREUSSAY P.

Contribution à la définition interprétative et à l'implémentation des lambda-langages. Thèse d'état Université de Paris VII (1977).

3 MACCARTY et AL.

LISP 1.5 Programmers's Manual. M.I.T. Press Cambridge Massachussets (1965).

4 SUSSMAN C.J. STELLE G.L.

The revised report on SCHEME a dialect of LISP , A.I. Meno n° 452 (1978).

5 VUILLEMIN J.

Syntaxe, Sémantique et Axiomatique d'un langage de programmation simple. Birkhauser Verlag, Basel und Stutgart (1975).

6 WAND M.

Induction, recursion and programming, North Holland (1980).

7 WISE D.S.

Interpreters for functional programming and its applications, Cambridge University Press (1982).

Michel CAYROL

L.S.I. UNIVERSITE Paul SABATIER

118, route de Narbonne

31062 TOULOUSE CEDEX