DESS d'Informatique
Informatique des Systèmes Autonomes

LimboLisp
Interprèteur LISP pour le système INFERNO

Laurent BENCHADI
Samir GANA
Jean-Luc COSSI
Ghalleb KHELIDJ

Directeurs de Projet :
JEAN MÉHAT ET PATRICK GREUSSAY

2 *novembre* 2001

Université Paris8 StDenis
2, rue de la Liberté - 93526 Saint Denis

Table des matières

1 Introduction **2**

2 Les fonctionnalités **3**
- 2.1 Les fonctions de l'interprèteur LISP de base 3
- 2.2 Les fonctions supplémentaires 6
- 2.3 Fonctionnement de l'interprèteur 8
  - 2.3.1 La gestion de la mémoire 8
  - 2.3.2 Lecture des expressions en entrée 8
  - 2.3.3 Evaluation et affichage du résultat 9

3 Le source **10**
- 3.1 Le module projet.m 10
- 3.2 Le fichier principal projet.b 13

4 Le jeu de tests **35**
- 4.1 Essais des fonctions de base 35
- 4.2 Essais des fonctions supplémentaires 37

5 Les problèmes rencontrés **40**
- 5.1 L'interprète LISP de base en langage C 40
- 5.2 Contraintes du langage Limbo 40

6 Conclusion **42**
- 6.1 Un LISP sous INFERNO 42
- 6.2 Améliorations possibles 42

Chapitre 1

Introduction

Ce projet consiste en la réalisation d'un interprète Lisp en Langage Limbo pour le système d'exploitation nferno. Nous avons donc procédé en plusieurs parties :

- D'abord la partie apprentissage : il a fallu apprendre le langage Lisp et le langage Limbo pour pouvoir les utiliser.
- Ensuite nous avons pris connaissance des sources des interprètes de Patrick GREUSSAY[1] en langage C et en Java.

[1]Enseignant à l'Université Paris8, il nous a donné des cours de LISP.

Chapitre 2

Les fonctionnalités

Nous avons conservé les fonctionnalités de l'interprète Lisp de base qui nous a été fourni, puis nous avons rajouté les fonctions supplémentaires demandées dans le cours de Patrick GREUSSAY.

2.1 Les fonctions de l'interprèteur LISP de base

quote

Elle ramène en valeur l'objet lisp donné en argument tel que.

car

Renvoie le "car" de la liste.

cdr

Renvoie le "cdr" de la liste.

sub1

Enlève "1" à un entier.

add1

Rajoute la valeur 1 à un entier.

defun

Permet de définir une fonction.

print

Affichage d'un objet lisp.

setq

Fonction d'affectation.

eq

Teste l'égalité de deux objets lisp.

if

Condition.

times

Permet de faire une multiplication.

oblist

Affiche tous les atomes.

cons

Crée un doublet.

plus

Teste si le premier élément est plus grand que le second.

less

Teste si le premier élément est inférieur au second.

minus

Soustraction.

div

La division.

caar

C'est la "car " d'un autre "car".

cadr

C'est le "car" d'un "cdr".

cdar

C'est le "cdr" d'un "car".

cddr

C'est le "cdr" d'un "cdr".

last

Renvoie le dernier élément d'une liste.

butlast

Renvoie une liste sans son dernier élément.

max

Renvoie le maximum de deux nombres.

min

Renvoie le minimum de deux nombres.

square

Il permet de faire le carré d'un nombre.

half

Il renvoie la première moitié d'une liste.

2.2 Les fonctions supplémentaires

length

Cette fonction retourne le nombre d'élément d'une liste :

(length '(a b)) -> 2

nconc

Cette fonction concatène deux listes pouvant être de longueur variable :

(ncons '(a b) '(d e f)) -> (a b c d e)

delq

Cette fonction supprime toutes les occurences du premier argument dans la liste passée en deuxième argument :

(delq 2 '(2 2 3 2 4 2 5 2 2 6 2)) - > (3 4 5 6)

equalp

Cette fonction vérifie l'égalité des termes de deux listes :

(equalp '(1 2 3) '(1 2 3)) -> t

nth

Cette fonction retourne l'élément de la liste passée en premier argument situé à la position définie par le deuxième argument :

(nth '(a b c d) 3) -> c

midelt

Cette fonction retourne l'élément médian d'une liste :

(midelt '(a b c d e)) -> c

poly

Elle permet de calculer un polynôme de la forme (poly x (a b c .... n)) -> a + bx + cx^2+ .... nxk
(poly 2 '(3 5 7)) -> 41

val

(val 'b '((a 1) (b 2) (c 3))) retourne la valeur associée à l'atome b, donc 2.

max3

Cette fonction renvoie le plus grand nombre de la liste.
(max3 (3 5 9)) -> 9

listadd

Fait la somme des éléments d'une liste.
(listadd '(1 3 2 5)) -> 11

mconc

Permet de concatener plusieurs listes. Elles doivent toutes se trouver dans une liste unique.

(mconc '((a b) ((c)) (d e f))) -> (a b c d e f)

decimate

De la liste qu'on lui passe, il enlève tout élément dont la position est un nombre impair, sauf le premier élément.

(decimate '(a b c d e f)) -> (a c e)

2.3 Fonctionnement de l'interprèteur

2.3.1 La gestion de la mémoire

Pour la faire, nous avons créé une mémoire virtuelle.

Pour simplifier le fonctionnement, nous allouons un tableau qui simulera la mémoire est qui sera divisé en espace pour chaque type. Les atomes seront stockés dans le début du tableau et seront des liens sur la position de l'atome dans la tables des "p_names", ce qui permettra d'exécuter leur code grâce à la fonction "eval()". Dans la seconde partie, ce seront les doublets qui feront des liens vers d'autres doublets, vers des atomes ou vers des entiers. Ensuite dans la fin du tableau nous allons simuler une pile, qui sera utilisée lors des appels de "pop()" et "push()".

Pour savoir quel est le type d'un élément, il suffira donc de vérifier l'adresse. Si cette adresse se trouve dans la première partie du tableau, ce sera un atome, si elle se trouve dans la seconde, ce sera un doublet et si elle est supérieure au maximum du tableau alloué, ce sera donc un entier.

Bien sûr dans le cas des entier, il conviendra de recadrer la valeur, par exemple pour créer un "int" (c'est à dire que l'adresse est en dehors du tableau), il faut augmenter l'adresse (qui contient la valeur de l'entier) de la valeur de "MAXMEM" (c'est la taille du tableau). Ensuite pour en refaire un entier, il suffit de retirer la valeur de "MAXMEM" à l'adresse pour trouver la valeur de l'entier.

2.3.2 Lecture des expressions en entrée

Cette partie est constituée d'un ensemble de fonctions qui permettent de lire correctement et de stocker la liste saisie au "prompt" du programme. Il s'agit de

la lecture d'un objet lisp.

Une fonction de lecture d'objet, "obj_read", lit un par un (par le biais de la fonction "read_char") les caractères qui sont saisis au "prompt". Lors de cette lecture, les atomes sont reconnus (grâce à la fonction "atom_read" qui lit une chaîne et détecte s'il s'agit d'un atome en la comparant aux éléments stockés dans le tableau "p_names") et stockés ; les listes sont reconnus (par un test sur les parenthèses ouvrantes et fermantes) et crées effectivement grâce à la fonction "cons".

Un objet lisp, résultat de la lecture, est stocké dans le régistre "a0" à des fins d'évaluation.

2.3.3 Evaluation et affichage du résultat

Il s'agit principalement de l'évaluation de l'expression saisie au "prompt" et de l'affichage du résultat obtenu.

La partie évaluation, "eval()" et "eval_car()", permet de déterminer le traitement à effectuer sur la liste saisie au prompt du programme après avoir évalué les éléments qui composent cette liste par les fonctions "eval_1_arg()", "eval_2_args()" ou "eval_3_args()".

C'est dans cette partie que les fonctions de base et celle que nous avons rajoutées sont insérées.

Les fonctions d'affichage du résultat se décomposent en trois principales ; la première est "lisp_print" qui est la fonction qui est appelé lors de la fin de l'exécution d'un programme "LISP". "lisp_print" va appelé la fonction "obj_print" qui va afficher correctement les éléments selon leur type (Atome, Nombre, ou Liste). Dans les deux premiers cas les éléments sont affichés directement mais pour une liste, on fait appel à la fonction "list_print". "list_print" va afficher la première "(" du "LISP" et faire appel pour tous les éléments du résultat la fonction "obj_print" et pour finir afficher ")".

Chapitre 3

Le source

Le source de notre interprète LISP se compose de 2 fichiers : le fichier nous permettant de déclarant les prototypes des fonctions utilisées ainsi que les variables nécessaires, et le fichier qui intègre le programme principal et les fonctions.

3.1 Le module projet.m

```
# All rights/write not reserved
# Octobre 2001. version 1.0.
# LimboLisp par G. KHELIDJ, ghalleb@ghalleb.com
# S. GANA,  versace@mime.up8.edu
# J.L. COSSI, cossi@mime.up8.edu
# L. BENCHADI droopy@mime.up8.edu
# Please send us 4 tee-shirts "Inferno" Thanks.
# Universite Paris VIII
# DESS Informatique des systemes autonomes.
LimboLisp : module {
#### Les fonctions
# le main et la fonction de gestion des evenement
init : fn( ctxt : ref Draw->Context, args : list of string);
toplevel : fn();
# fonctions d'initialisation
```

```
init_atomes : fn();
init_listes : fn();
init_stack : fn();
init_read : fn ();
#  les fonctions de lecture
is_letter : fn (ch : int) : int;
is_digit : fn (ch : int) : int;
is_separator : fn (ch : int) : int;
read_char : fn () : int;
unread_char : fn (ch : int);
lisp_read : fn ();
obj_read : fn () : int;
cons_read : fn () : int;
int_read : fn (ch : int) : int;
atom_read : fn (ch : int) : int;
# fonction d'affichage
lisp_print : fn();
obj_print : fn(obj : int);
list_print : fn(obj :int);
# fonction d'empillage et de depillage
push : fn(x : int);
pop : fn() : int;
# les fonctions d'evaluation
eval : fn();
eval_car : fn();
power : fn(a : int, b :int) :int;
eval_1_arg : fn();
eval_2_args : fn();
eval_3_args : fn();
progn : fn();
evlis : fn();
bind : fn(x :int, y :int);
unbind : fn();
```

```
# les fonctions classique interne de Lisp
CAR : fn(adr : int) : int;
CDR : fn(adr : int) : int;
CAAR : fn(adr : int) : int;
CADR : fn(adr : int) : int;
CDAR : fn(adr : int) : int;
CDDR : fn(adr : int) : int;
# recuperation de la valeur
CVAL : fn(adr : int) : int;
# detection de type
IS_ATOM : fn(adr : int) : int;
IS_CONS : fn(adr : int) : int;
IS_STACK : fn(adr : int) : int;
IS_INT : fn(x : int) : int;
# affectation des valeurs
setcval : fn(adr : int, val : int);
setcar : fn(adr : int, val : int);
setcdr : fn(adr : int, val : int);
# recadrage des entiers
make_int : fn(x : int) : int;
val_int : fn(x : int) : int;
# creation d'un atome
cratom : fn(nom_at : string) : int;
# creation d'une liste
cons : fn(x : int, y : int) : int;
# fonction d'erreur
gc : fn();
#### Les variables
# tableau de memoire virtuelle
mem : array of int;
# buffer de lecture et son pointeur
read_buf :string;
pos_read_buf : int;
```

```
# constante de division de la memoire
B_ATOME :  con 0;
B_CONS :  con 256;
B_STACK : con 7000;
MAXMEM : con 8092;
#  le nombre d'atome en fonction de la memoire virtuelle allouee
MAX_ATOMES : con B_CONS - B_ATOME;
# pointeurs de pile
sp : int;
freelist : int;
# les registres
a0 : int;
a1 : int;
a2 : int;
a3 : int;
a4 : int;
#  les constantes de Lisp
tnil :  con 0;
undefined :  con 1;
t :  con 2;
quote :  con 3;
lambda : con 8
```

3.2 Le fichier principal projet.b

```
implement LimboLisp;
include "projet.m";
include "sys.m";
sys : Sys;
include "draw.m";
draw : Draw;
include "string.m";
chaine : String;
```

```
include "bufio.m";
bufio : Bufio;
mem= array[MAXMEM] of int;
p_names = array[MAX_ATOMES] of {
"nil", "undefined", "t", "quote", "car", "cdr", "add1", "sub1",
"lambda", "de", "print", "setq", "eq", "if", "times", "oblist", "cons",
"plus", "less", "minus", "div", "caar", "cadr", "cdar","cddr",
"last", "butlast", "middle", "max", "min", "square", "half", "length",
"ncons", "delq", "equalp", "nth", "midelt", "poly", "val", "max3",
"listadd", "mconc", "decimetre"
};
init_read () {
pos_read_buf = 80;
}
unread_char (ch : int) {
if (pos_read_buf != 80) {
--pos_read_buf;
}
}
read_char () : int {
ch : int;
stdin := bufio->fopen(sys->fildes(0), bufio->OREAD);
if (pos_read_buf >= 80) {
read_buf= bufio->stdin.gets('\n');
pos_read_buf = 0;
}
ch = read_buf[pos_read_buf++];
if (ch == '\n') {
pos_read_buf = 80;
}
return(ch);
}
lisp_read () {
```

```
a0 = obj_read ();
}
obj_read () : int {
ch : int;
res : int;
do {
ch = read_char();
} while (is_separator(ch) || ch == ')');
if (ch == '\") {
res = obj_read();
res = cons (quote, cons(res, tnil));
return(res);
}
if (is_digit(ch)) {
res = int_read (ch);
}
else {
if (ch == '(') {
return(cons_read ());
}
else {
if (is_letter(ch)) {
res = atom_read(ch);
}
else {
sys->print ("Erreur read, caractere inconnu %d\n", ch);
}
}
}
return res;
}
is_separator (ch : int) : int {
return (chaine->in(ch, " \t\n"));
```

```
}
int_read (ch : int) : int {
val : int;
val = ch - '0';
ch = read_char ();
while (is_digit(ch)) {
ch = ch - '0';
val = val * 10 + ch;
ch = read_char ();
}
unread_char (ch);
return (make_int(val));
}
atom_read (ch : int) : int {
res : int;
atbuf : string;
i : int;
i = 0;
atbuf[i++] = ch;
ch = read_char();
while (is_letter(ch) || is_digit(ch)) {
atbuf[i++] = ch;
ch = read_char();
}
unread_char(ch);
for (i = tnil; i < no_atoms; i++) {
if (atbuf == p_names[i]) {
return (i);
}
}
return( cratom(atbuf));
}
is_letter (ch : int) : int {
```

```
return (chaine->in(ch, "a-zA-Z"));
}
is_digit (ch : int) : int {
return (chaine->in(ch, "0-9"));
}
cons_read () : int {
res : int;
ch : int;
x : int;
res =  cons (tnil, tnil);
x =res;
for (;;) {
while (is_separator(ch = read_char()));
if (ch == ')') {
return (CDR(res));
}
unread_char (ch);
setcdr (x, cons(obj_read(), tnil));
x = CDR(x);
}
}
lisp_print()
{
obj_print(a0);
}
obj_print(obj : int)
{
if(IS_ATOM(obj)) {
sys->print("%s",p_names[obj]);
return;
}
if(IS_INT(obj)) {
sys->print("%d",val_int(obj));
```

```
return;
}
if(IS_CONS(obj)) {
list_print(obj);
return;
}
sys->print("Objet à imprimer inconnu + obj" );
}
list_print(obj : int)
{
sys->print("(");
for(;;) {
obj_print(CAR(obj));
if((obj = CDR(obj)) == 0) break;
sys->print(" ");
if(IS_CONS(obj)) continue;
sys->print(". "); obj_print(obj); break;
}
sys->print(")");
}
push(x : int) {
if (sp == MAXMEM - 1) {
init_stack();
sys->print("Pile debordee\n");
}
mem[++sp] = x;
}
pop() : int {
return(mem[sp--]);
}
init (ctxt : ref Draw->Context, args : list of string)
{
sys= load Sys Sys->PATH;
```

```
bufio= load Bufio Bufio->PATH;
chaine= load String String->PATH;
sys->print("Limbo Lisp par G. KHELIDJ, S. GANA,
J.L. COSSI, L. BENCHADI\n");
sys->print("DESS ISA, Universite Paris VIII\n");
init_stack();
init_atomes();
init_listes();
init_read();
toplevel();
}
toplevel() {
for(;;) {
sys->print("-> ");
lisp_read();
eval();
sys->print("= ");
lisp_print();
sys->print("\n");
}
}
eval() {
  if(IS_INT(a0)) {
  return;
}
  if(IS_ATOM(a0)) {
  if(CVAL(a0) == undefined && a0 != undefined) {
sys->print("Atome %s, valeur indefinie\n",p_names[a0]);
}
  a0 = CVAL(a0); return;
 }
  a4 = CAR(a0);
eval_car();
```

```
}
eval_car(){
  if(IS_INT(a4)) {
  sys->print("Nombre en position fonctionnelle %d\n", val_int(a4));
}
  if(IS_ATOM(a4)) {
  case (a4){
3 =>
  a0 = CADR(a0);
  4 =>
  eval_1_arg();
  a0 = CAR(a0);
5 =>
  eval_1_arg();
  a0 = CDR(a0);
6 =>
eval_1_arg();
a0 = make_int(val_int(a0) + 1);
7 =>
  eval_1_arg();
  a0 = make_int(val_int(a0) - 1);
9 =>
  a0 = CDR(a0);
  a1 = CAR(a0);
  setcval(a1, cons(lambda, CDR(a0)));
  a0 = a1;
10 =>
  eval_1_arg();
  obj_print(a0);
  sys->print("\n");
  11 =>
  push(CADR(a0));
  a0 = CADR(CDR(a0));
```

```
  eval();
  a1 = pop();
    setcval(a1, a0);
12 =>
  eval_2_args();
  if(a1 == a0)
  a0 = t;
  else
  a0 = tnil;
13 => #if
  push(a0);
  a0 = CADR(a0);
  eval();
  a1 = pop();
  push(a1);
  if(a0 == tnil)
  a0 = CADR(CDDR(a1));
  else
  a0 = CADR(CDR(a1));
eval();
14 =>
  eval_2_args();
  a0 = make_int(val_int(a1) * val_int(a0));
15 =>
  a0 = tnil;
  for(a1 = no_atoms-1; a1 >= 0; a1--)
  a0 = cons(a1, a0);
# debut des conneries
16 => # cons
eval_2_args();
a0 = cons(a0, a1);
17 => # plus
eval_2_args();
```

```
a0 = make_int(val_int(a0) + val_int(a1));
18 => # less
eval_2_args();
if(val_int(a0) < val_int(a1))
a0 = t;
else
a0 = tnil;
19 => # minus
eval_2_args();
a0 = make_int(val_int(a0) - val_int(a1));
20 => # div
eval_2_args();
a0 = make_int(val_int(a0) / val_int(a1));
21 => # caar
eval_1_arg(); a0 = CAAR(a0);
22 => # cadr
eval_1_arg(); a0 = CADR(a0);
23 => # cdar
eval_1_arg(); a0 = CDAR(a0);
24 => # cddr
eval_1_arg(); a0 = CDDR(a0);
25 => # last : dernier element de la liste
eval_1_arg();
while (CDR(a0)!= tnil) a0 = CDR(a0);
a0 = CAR(a0);
26 => # butlast : liste sans le dernier element
eval_1_arg();
a1 = a0; # a1 : registre de travail
if (CDR(a1) == tnil){
 setcar(a1, tnil);
 return;
}
while(CDDR(a1)!= tnil) a1 = CDR(a1);
```

```
setcdr(a1, tnil);
27 => # middle
eval_1_arg();
a1 = a0;
if (CDR(a1) == tnil){
setcar(a1, tnil);
return;
}
while(CDDR(a1) != tnil) a1 = CDR(a1);
setcdr(a1, tnil);
a0 = CDR(a0);
28 => # max
eval_2_args();
if(val_int(a0) < val_int(a1)) a0 = a1;
29 => # min
eval_2_args();
if(val_int(a0) > val_int(a1)) a0 = a1;
30 => # square
eval_1_arg();
a0 = make_int(val_int(a0) * val_int(a0));
 31 => # half
# examen du premier MODULE
32 => # length
length := 0;
eval_1_arg();
if(a0 == tnil){ # si liste vide
a0 = make_int(length);
return;
}
while(CDR(a0) != tnil){
length++;
a0 = CDR(a0);
}
```

```
a0 = make_int(++length);
33 => # ncons
eval_2_args();
if(a0 == tnil){ # si 1ere liste vide
 a0 = a1;
 return;
}
a2 = a0;
while(CDR(a2) != tnil) a2 = CDR(a2);
setcdr(a2,a1);
34 => # delq
eval_2_args();
while(val_int(CAR(a1)) == val_int(a0)){ #si en tete de liste
if(CDR(a1) == tnil){
  a0 = tnil;
  return;
}
a1 = CDR(a1);
}
a2 = a1; # sauvegarde du debut
while(CDR(a2) != tnil){
if(val_int(CADR(a2)) == val_int(a0)){
if(CDDR(a2) == tnil){
  setcdr(a2, tnil);
  return;
 }
 setcdr(a2, CDDR(a2));
}
else a2 = CDR(a2); # seulement si pas le delq
}
a0 = a1;
 35 => # equalp
eval_2_args();
```

```
if ((a0 == tnil) || (a1 == tnil)){ # si liste(s) vide(s)
if(a0 == a1)
a0 = t;
else
a0 = tnil;
return;
}
while(CAR(a0) == CAR(a1)){
if((CDR(a0) == tnil) || (CDR(a1) == tnil)){ # si listes != tailles
 if(CDR(a0) == CDR(a1))
  a0 = t;
 else
  a0 = tnil;
 return;
}
a0 = CDR(a0);
a1 = CDR(a1);
}
36 => # nth
nth : int;
eval_2_args();
if (a0 == tnil)
return; # liste vide : nil
if (val_int(a1) == 0){ # le 0eme : nil
a0 = tnil;
return;
}
for(nth = 1; nth < val_int(a1); nth++){
if(CDR(a0) == tnil){ # si pas de Neme : nil
 a0 = tnil;
 return;
}
a0 = CDR(a0);
```

```
}
a0 = CAR(a0);
 37 => # midelt
eval_1_arg();
a1 = a0;
while(CDR(a1) != tnil){
if(CDDR(a1) == tnil){ # si paire : nil
 a0 = tnil;
 return;
}
a1 = CDDR(a1);
a0 = CDR(a0);
}
a0 = CAR(a0);
38 => # poly
eval_2_args ();
puiss := 0;
temp := 0;
while (CDR(a1) != tnil) {
temp += (val_int(CAR (a1)) * power (val_int(a0), puiss++));
a1 = CDR (a1);
}
a0 = make_int(temp + (val_int(CAR (a1)) * power (val_int(a0), puiss)));
39 => # val
eval_2_args();
while(a1 != tnil){
if(CAAR(a1) == a0){
a0 = CAR(CDAR(a1));
return;
 }
 else
a1 = CDR(a1);
}
```

```
a0 = tnil;
40 => # max3
eval_3_args();
if(val_int(a0) < val_int(a1))
if(val_int(a1) < val_int(a2))
a0 = a2;
else
a0 = a1;
else
if(val_int(a0) < val_int(a2))
a0 = a2;
else
a0 = a0;
41 => # listadd
eval_1_arg();
a1 = make_int(val_int(CAR(a0)));
while(CDR(a0) != tnil){
a0 = CDR(a0);
a1 = make_int(val_int(a1) + val_int(CAR(a0)));
}
a0 = a1;
42 => # mconc
eval_1_arg();
a1 = CAR(a0);
a2 = a1;
while(a0 != tnil){
while(CDR(a1) != tnil)
a1 = CDR(a1);
a0 = CDR(a0);
setcdr(a1, CAAR(a0));
}
a0 = a2;
43 => # decimetre
```

```
eval_1_arg();
a1 = a0;
while (CDR(a1) != tnil) {
if(CDDR(a1) == tnil){
setcdr(a1, tnil);
return;
}
setcdr(a1, CDDR(a1));
a1 = CDR(a1);
}
# fin des conneries
* =>
  if(CVAL(a4) == undefined){
  sys->print("Fonction standard inconnue %s", p_names[a4]);
  break;
}
  a4 = CVAL(a4);
  eval_car();
}
}
if (CAR(a4) == lambda) {
  push(a4);
  push(a2);
  push(a0);
  a0 = CDR(a0);
  evlis();
  a1 = a0;
  a0 = pop();
  a2 = pop();
  a4 = pop();
  bind(CADR(a4), a1);
  a0 = CDDR(a4);
  progn();
```

```
  unbind();
  return;
}
}
power (a : int, b :int) :int {
resultat := 1;
for (i := b; i > 0; i--) {
 resultat *= a;
}
return (resultat);
}
eval_1_arg() {
a0 = CADR(a0);
eval();
}
eval_2_args() {
push(a0);
a0 = CADR(CDR(a0));
eval();
a1 = a0;
a0 = pop();
a0 = CADR(a0);
eval();
}
eval_3_args() {
 push(a0); a0 = CADR(CDDR(a0)); eval();
 a2 = a0; a0 = pop();
 push(a0); a0 =CADR(CDR(a0)); eval();
 a1 = a0; a0 = pop();
 a0 = CADR(a0); eval();
}
progn() {
a1 = a0;
```

```
for(;;) {
if(a1 == tnil) {
return;
}
push(a1);
a0 = CAR(a1);
eval();
a1 = pop();
a1 = CDR(a1);
}
}
evlis() {
a2 = cons(tnil, tnil);
a4 = a2;
  push(a4);
  for (;;) {
  if(a0 == tnil) {
  break;
}
  push(a0);
  push(a2);
  a0 = CAR(a0);
  eval();
  a2 = pop();
  setcdr(a2, cons(a0, tnil));
  a2 = CDR(a2);
  a0 = pop();
  a0 = CDR(a0);
 }
  a4 = pop();
  a0 = CDR(a4);
}
# fonction a corriger
```

```
bind(x : int, y :int) {
  z := x;
  push(undefined);
  while(z != tnil) {
  push(CVAL(CAR(z)));
  push(CAR(z));
  z = CDR(z);
}
  while(x != tnil) {
  setcval(CAR(x), CAR(y));
  x = CDR(x);
  y = CDR(y);
 }
}
unbind() {
 x : int;
 y : int;
 for(;;)
 {
 x = pop();
 if(x == undefined)
  return;
 y = pop();
 setcval(x, y);
 }
}
#
definition des car et cdr et de tout ce qui s'en suit
CAR(adr : int) :int {
return(mem[adr]);
}
CDR(adr : int) :int {
return(mem[adr+1]);
}
```

```
CAAR(adr : int) :int {
return(CAR(CAR(adr)));
}
CADR(adr : int) :int {
return(CAR(CDR(adr)));
}
CDAR(adr : int) :int {
return(CDR(CAR(adr)));
}
CDDR(adr : int) :int {
return(CDR(CDR(adr)));
}
#
cval qui retourne le car (qui a donc le meme code que car)
CVAL(adr : int) : int {
return(mem[adr]);
}
# les fonctions de detection de type
IS_ATOM(adr : int) : int {
return(adr >=0 && adr < B_CONS);
}
IS_CONS(adr : int) : int {
return(adr >=B_CONS && adr < B_STACK);
}
IS_STACK(adr : int) : int {
return(adr >=B_STACK && adr < MAXMEM);
}
IS_INT(x : int) : int {
return(x >=MAXMEM);
}
# assignation des valeurs
setcval(adr : int, val :int) {
mem[adr] = val;
}
```

```
setcar(adr : int, val :int) {
mem[adr] = val;
}
setcdr(adr : int, val :int) {
mem[adr+1] = val;
}
# recadrage des entiers
make_int(x : int) : int {
return(x+MAXMEM);
}
val_int(x : int) : int {
return(x-MAXMEM);
}
init_atomes() {
i : int;
for (i=tnil; i< MAX_ATOMES; i++) {
setcval(i, undefined);
}
no_atoms = N_ATOMS;
for (i=tnil; i< MAX_AUTO_EVAL; i++) {
setcval(i, i);
}
}
init_listes() {
i : int;
freelist = tnil;
for (i= B_CONS; i<= B_STACK; i+=2) {
setcdr(i, freelist);
freelist = i;
}
sys= load Sys Sys->PATH;
}
init_stack() {
```

```
sp = B_STACK;
}
cons(x : int, y : int) : int {
res : int;
if (freelist == tnil) {
gc();
}
setcar(freelist, x);
res = freelist;
freelist = CDR(freelist);
setcdr (res, y);
return(res);
}
gc() {
sys->print("Helas, plus de doublets libres!\n");
exit;
}
cratom(nom_at : string) : int {
res : int;
if (no_atoms >= MAX_ATOMES) {
sys->print("Vous avez depasse le nbre maximum d'atomes %d\n!", MAX_ATOMES)
exit;
}
res = no_atoms;
setcval(res, undefined);
p_names[no_atoms++] = nom_at;
return(res);
}
```

Chapitre 4

Le jeu de tests

Pour vérifier le bon fonctionnement de notre programme, nous avons procédé à un jeu de test complet afin de montrer la validités de nos fonctions.

4.1 Essais des fonctions de base

Ce sont les fonctions implémentées par défaut dans l'interprèteur[1] dont nous nous sommes inspirés. Notre programme écrit en langage Limbo s'est comporté de la manière suivante :

// Lancement de l'executable :

```
% projet

Limbo Lisp par G. KHELIDJ, S. GANA, J.L. COSSI, L. BENCHADI
DESS ISA, Universite Paris VIII
-> nil
= nil
-> t
= t

-> undefined
= undefined

-> (oblist)
```

[1] Le programme de Patrick Greussay écrit en langage C

= (nil undefined t quote car cdr add1 sub1 lambda de print setq eq if times oblist)

-> (quote (a b c))
= (a b c)

-> (car '(a bc cd))
= a

-> (cdr '(a (bc) (abcd)))
= ((bc) (abcd))

-> (setq a '(L I S P))
= (L I S P)

-> a
= (L I S P)

-> (car 'a)
= (L I S P)

-> (car a)
= L

-> (cdr a)
= (I S P)

-> (cdr(cdr a))
= (S P)

-> (de carre(x) (times x x))
= carre

-> (carre 9)
= 81

-> (de cube(y) (times (times y y) y))
= cube

-> (oblist)
= (nil undefined t quote car cdr add1 sub1 lambda de print setq eq if times oblist cons plus less minus div caar cadr cdar cddr last butlast middle max min square half length ncons delq equalp nth midelt poly val max3 listadd mconc decimate carre x cube y)

-> (cube 6)
= 216

-> (add1 7)
= 8

-> (sub1 8)
= 7

-> (eq (car'(a bc d)) (car(cdr'(b a d))))
= t

-> (if (eq (car '(b bc d)) (car(cdr '(b a d)))) (print 'EQUAL) (print 'DIFF))
EQUAL
= EQUAL

-> (if (eq (car '(b bc d)) (car(cdr '(b a d)))) (print 'EQUAL) (print 'DIFF))
DIFF
= DIFF
Les résultats obtenus sont satisfaisants.

4.2 Essais des fonctions supplémentaires

Ce sont les fonctions à rajouter pour l'examen. Les tests associés que nous avons pu mettre en oeuvre sont :
-> (last '(a c f t h b g))
= g

-> (middle '(a c f t h b g))
= (a c f t h b)

```
-> (middle '(a c f t h b g))
= (c f t h b)

-> (decimate '(h j u k l i))
= (h u l)

-> (nth '(l a u) 3)
= u

-> (delq 2 '(2 2 2 5 7 9 2 8))
= (5 7 9 8)

-> (middle '(a c f t h b g))
= (c f t h b)

-> (max3 6 8 2)
= 8

-> (poly 2 '(3 2 1))
= 11

-> (div 16 2)
= 8

-> (square 3)
= 9

-> (length '(a b c d e))
= 5

-> (less 11 3)
= nil

-> (less 11 29)
= t

-> (minus 4 2)
```

```
= 2

-> (max 2 3)
= 3

-> (min 23 456)
= 23

-> (equalp '(a c d) '(a c d e))
= nil

-> (equalp '(a c d) '(a c d))
= t

-> (midelt '(a b c d e))
= c

-> (val 2 '((2 a) (3 b)))
= a

-> (ncons '(a b) '(d g))
= (a b d g)

-> (listadd '(1 2 2))
= 5
-> (half '(a b c d))
= (a b)

-> (half '(a b c))
= nil

-> (mconc '((a b)(c d e f)()(g h i jk)))
= (a b c d e f nil g h i jk)

-> (mconc '(a b c    () (d e) ()))
= (a b c nil d e nil)
```

Chapitre 5

Les problèmes rencontrés

Les difficultés rencontrées sont surtout dûes au fait qu'il nous a fallu adopter un nouveau langage sous un nouvel environnement en peu de temps.

5.1 L'interprète LISP de base en langage C

Nous nous sommes principalement aidé du source en langage C pour faire notre programme, notamment pour tout ce qui est de la gestion mémoire. Mais nous n'avons pas pu voir fonctionner l'interprète en langage C, pour plusieurs raisons : l'absence de certaines fonctions, (comme les fonctions "bind" et "unbind"), mais aussi de fonctions erronées ("unread char" notamment qui produit un dépassement d'allocation).

5.2 Contraintes du langage Limbo

Certains Problèmes sont apparus à l'utilisation de plusieurs module. Nous n'avons pas pu affecter une valeur dans un module et la récupérer dans un autre en passant par une variable globale ; nous avons constaté que le programme a réinitialisé la valeur dans chaque module.

– En Limbo nous ne disposons d'aucun accés mémoire, et pas de possibilités d'utilisation de pointeurs. Les références ressemblent aux pointeurs mais n'ont à notre sens que peut d'utilité compte tenu du fait que le langage Limbo est modulaire.

- L'utilisation d"'objets" n'est pas vraiment prévu dans la structure du langage Limbo. Les "pick adt" qui ressemblent à ce que sont les "unions" au langage C, se sont avérés difficiles à mettre en oeuvre dans un projet comme le nôtre.
- Nous n'avons également pas pu nous servir de syntaxes similaires aux "define" du langage C ; il nous a falludonc créer nos propres fonctions d'affectations et et récupération de données de façon séparée (pour exemple, la fonction "CAR" récupère la variable et la fonction "setcar" permet de l'affecter).
- Il existe une limite au niveau des nombres entiers : nous recadrons les entiers (avec la fonction "make_int" qui augmente la valeur de l'entier de "MAXMEM") donc, la valeur maximale d'un entier utilisable est de 2puissance 32 (car un entier est sur 32 bits) moins "MAXMEM" ; sinon nous faisons un dépassement de capacité.

Chapitre 6

Conclusion

Ce projet nous a permis de comprendre le fonctionnement d'un interprète de langage et plus particulièrement un intèrprète Lisp. Nous avons acquis des notions de la structure modulaire du langage Limbo.

6.1 Un LISP sous INFERNO

Nous contribuons à l'extension du systéme d'expoitation embarqué Inferno en lui apportant son premier intèrprète Lisp.

6.2 Améliorations possibles

Un changement de notre méthode de gestion de la memoire virtuelle, qui non seulement limite de nombre d'élément (de doublets ou d'atomes), mais aussi comme nous l'avons expliqué dans les problèmes rencontrés, ne permet pas l'utilisation de tout les entiers contenus dans 32 bits(l'entier maximal utilisable est de 2puissance 32 moins "MAXMEN").

- Nous pourrions par exemple faire un tableau a deux dimensions avec dans la premiere colonne la donnée et dans la seconde, le type codé en "int" (par exemple 1 signifierait que c'est un "int", 2 un atome etc).
- Nous pourrions aussi ajouter d'autres fonctions maintenant que nous maitrisons le fonctionnement de l'intèrprète Lisp.
- Une routine d'erreur avec des messages d'erreur corrects, la cause de l'erreur et pourquoi pas la façon de la rectifier.

- Il serait intéressant aussi de pouvoir utiliser des nombres négatifs mais là aussi le problème est le même que pour les grands entiers, le recadrage donnerait une reponse erronée.
- Il serait convenable aussi de développer un “garbage collector” pour effacer les doublets inutilisés et nettoyer la table de “p_names” en cas d’échec par exemple.
- Il serait judicieux d’utiliser une table “Hash codées” pour les “p_names”, bien que pour 256 atomes, cela ne soit pas vraiment nécéssaire. Le temps de recherche est presque négligeable.