

```
1 ;Samedi 6 Avril 2002 11:18:19 file : symeval.vlisp ;
2
3 ;----- pretty-print, par [pg];
4 ; transformation en evaluateur 24.3.02 (hw);
5
6 (setq --x-- careful careful nil)
7
8 (df pretty (l) pretty
9 (goPretty) (setq activity `prettyFtn) (mapc l (lambda (f) (terpri) (pprint f))))
10
11 (de symeval (exp) symeval
12 (goSymEval)
13 (setq activity `symb)
14 (let ((x exp) (y (interprete exp))) (if (equal x y) x (self y (interprete y))))))
15
16 (de pprint (f ;; x) pprint
17 (status print 3)
18 ; bit 0 : '"" , bit 1 : pas d'espace.;
19 (setq lmargin 0)
20 (setq x (cdr (assoc (ftype f) '((7 . de) (8 . df) (9 . macro)))))
21 (and x (interprete [x f . (fval f)]))
22 (terpri)
23 (status print 0)
24 f)
25
26 (de interprete (l ;; al xx) interprete
27 (initializeInterprete)
28 (cond
29 ((null l) (casNil))
30 ((atom l) (casAtom))
31 ((and (eq (car l) quote) (null (cddr l))) (casQuote))
32 ((and (listp (car l)) (eq (caar l) lambda)) (casLambda))
33 (t (initGeneralCase)
34 (let
35 ((x
36 (let ((f (and (litatom (car l)) (get (car l) activity))))
37 (if f (apply f nil) (defaultCase)))) (exitGeneralCase x))))))
38
39 (de goSymEval () goSymEval
40 ; pour initialiser l'ensemble des fonctionalites;
41 ; pour l'interprete symbolique;
42 (de casNil nil nil)
43 (de casAtom () casAtom
44 (let (val (assoc l al)) (if val (cadr val) l)))
45 (de casQuote nil l)
46 (de casLambda () casLambda
47 (setq
48 l
49 ['let
50 (let ((v (cadar l)) (a (cdr l)))
51 (if (null v) nil [(nextl v) (nextl a)] . (self v a)))
52 . (cddar l)])
53 (interprete l))
```

```

54 (de initGeneralCase nil nil)
55 (de initializeInterprete nil nil)
56 (de exitGeneralCase (x)
57   x)
58 (de defaultCase ()
59   (cond
60     ((or (null (get (car l) 'body)) (listp (car l)))
61      [(interprete (nextl l) al) . (mapcar l '(lambda (x) (interprete x al)))]))
62     (t (let (largs (mapcar (cdr l) '(lambda (x) (interprete x al))))
63         (let (al1 (redpair (car l) (get (car l) 'vars) largs))
64           (if (eq al1 'no)
65               [(car l) . largs]
66               (interprete (get (car l) 'body) (nconc al1 al))))))))))
67 (put 'cdr
68   'symb
69   '(lambda ()
70     (let ((x (interprete (cadr l) al)))
71       (cond ((null x) nil) ((equal (car x) 'cons) (caddr x)) (t ['cdr x])))))
72 (put 'car
73   'symb
74   '(lambda ()
75     (let ((x (interprete (cadr l) al)))
76       (cond ((null x) nil) ((equal (car x) 'cons) (cadr x)) (t ['car x])))))
77 (put 'null
78   'symb
79   '(lambda ()
80     (let ((x (interprete (cadr l) al)))
81       (cond ((null x) t) ((and (listp x) (eq (car x) 'cons)) nil) (t ['null x])))))
82 (put 'if
83   'symb
84   '(lambda ()
85     (let ((x (interprete (cadr l) al)))
86       (cond ((or (eq x t) (and (listp x) (eq (car x) 'cons))) (interprete (caddr l) al)) ((null x) (interprete (cadr (caddr l)) al))
87         (t ['null . [x . (caddr l)]])))
88 (put 'de 'symb '(lambda () (eval l) (analyse (cadr l)) (cadr l))))
89
90 (de analyse (f ;; x vars body decvars)
91   (setq x (fval f) vars (car x) body (cadr x))
92   (let (e body)
93     (cond
94       ((atom e)
95        ((member (car e) '(car cdr 1-))
96         (if (atom (cadr e))
97             (and
98              (member (cadr e) vars)
99              (or (member (cadr e) decvars) (newl decvars (cadr e))))
100            (self (cadr e))))
101       (t (self (nextl e)) (self e))))
102 (put f 'vars vars)
103 (put f 'body body)
104 (put f 'decvars decvars))
105
106 ; ----- controleur de deploiement -----;
107
108 (de redpair (fun vars largs ;; decvars res v l)
109   (setq decvars (get fun 'decvars))

```

*exitGeneralCase**defaultCase**analyse**redpair*

---

```

110 (while vars
111   (setq v (nextl vars) l (nextl largs))
112   (if (member v decvars) (and l (neq l 0) (neq (car l) 'cons) (neq (car l) '1+) (exit 'no)))
113   (newl res [v l]))
114 res)
115
116 (de app (x y) app)
117 (if (null x) y [(car x) . (app (cdr x) y)])
118
119 (analyse 'app)
120
121 (de rev (x) rev)
122 (if (null x) nil (app (rev (cdr x)) [(car x) . nil]))
123
124 (analyse 'rev)
125
126 (de goPretty () goPretty)
127 ; pour initialiser l'ensemble des fonctionalites;
128 ; pour le pretty-print;
129 (de initializeInterprete initializeInterprete)
130 (status print 3) ; pour les appels externes ;
131 (de casNil () casNil)
132 (princh "()")
133 (de casAtom () casAtom)
134 (prin1 l)
135 (de casQuote () casQuote)
136 (princh "") (interprete (cadr l))
137 (de casLambda () casLambda)
138 (setq
139   l
140   ['let
141    (let ((v (cadar l)) (a (cdr l)))
142      (if (null v) nil [(nextl v) (nextl a)] . (self v a)))
143    . (cddar l))
144    (interprete l))
145 (de initGeneralCase initGeneralCase)
146 (setq xx (outpos)) (princh "")
147 (de exitGeneralCase exitGeneralCase) (x)
148 (and l (princh " . ") (princh l) (princh ""))
149 (de defaultCase defaultCase)
150 (t+3) (interprete (nextl l)) (while (listp l) (p-p1)) (t-3))
151 (defmacro p-p1 () p-p1)
152 '(progn (princh " ") (interprete (nextl l)))
153 (defmacro t+3 () t+3)
154 '(setq lmargin (+ lmargin 3))
155 (defmacro t-3 () t-3)
156 '(setq lmargin (- lmargin 3))
157 (defmacro p-progn (?) p-progn)
158
159 '(if (and (null (cdr l)) (null ,?))
160     (p-p1)
161     ; un seul argument;
162     (t+3)
163     ; plusieurs;
164     (while (listp l)
165       (if (> lmargin (outpos)) (outpos lmargin) (terpri)))

```

---

```
166      (interpret (nextl l))
167      (t-3)))
168 (defmacro p-cond ()                                p-cond
169   '(progn
170     (t+3)
171     (while (listp l)
172            (terpri)
173            (princh "(")
174            (let (l (nextl l)) (interpret (nextl l)) (if l (p-progn)))
175            (princh ")"))
176     (t-3)))
177 (mapc '(progn ; type progn; prog1 and exit or)
178       (lambda (x) (put x 'prettyFtn '(lambda () (interpret (nextl l)) (p-progn)))))
179 (mapc '(lambda ; type while; escape if ifn let mapc mapcar while until)
180       (lambda (x) (put x 'prettyFtn '(lambda () (interpret (nextl l)) (p-p1) (p-progn t)))))
181 (mapc '(de ; type def; df dm dmc)
182       (lambda (x)
183         (put x 'prettyFtn '(lambda () (interpret (nextl l)) (p-p1) (p-p1) (p-progn t)))))
184 (mapc '(cond ; type cond;)
185       (lambda (x) (put x 'prettyFtn '(lambda () (interpret (nextl l)) (p-cond)))))
186 (mapc '(selectq ; type selectq;)
187       (lambda (x) (put x 'prettyFtn '(lambda () (interpret (nextl l)) (p-p1) (p-cond)))))
188
189 (setq careful --x--)
190
191
```

## Cross Reference

**analyse** de 90  
**app** de 116  
**goPretty** de 126  
**goSymEval** de 39  
**interprete** de 26  
**pprint** de 16  
**pretty** de 8  
**redpair** de 108  
**rev** de 121  
**symeval** de 11

1	<b>--x--</b>	6 189
2	<b>?</b>	157 159
3	<b>a</b>	50 51 51 141 142 142
4	<b>activity</b>	9 13 36
5	<b>al</b>	26 44 61 61 62 66 70 75 80 85 86
6	<b>all</b>	63 64 66
7	<b>analyse</b>	88 90 119 124
8	<b>app</b>	116 117 119 122
9	<b>body</b>	60 66 90 91 92 103 103
10	<b>casAtom</b>	30 43 133
11	<b>casLambda</b>	32 46 137
12	<b>casNil</b>	29 42 131
13	<b>casQuote</b>	31 45 135
14	<b>decvars</b>	90 99 99 104 104 108 109 109 112
15	<b>defaultCase</b>	37 58 149
16	<b>defmacro</b>	151 153 155 157 168
17	<b>e</b>	92 94 95 96 98 99 99 100 101 101
18	<b>exitGeneralCase</b>	37 56 147

---

19	<b>f</b>	9 9 16 20 21 21 24 36 37 37 90 91 102 103 104
20	<b>fun</b>	108 109
21	<b>goPretty</b>	9 126
22	<b>goSymEval</b>	12 39
23	<b>initGeneralCase</b>	33 54 145
24	<b>initializeInterprete</b>	27 55 129
25	<b>interprete</b>	14 14 21 26 53 61 61 62 66 70 75 80 85 86 136 144 150 152 166 174 178 180 183 185 187
26	<b>l</b>	8 9 26 29 30 31 31 32 32 36 36 44 44 45 48 50 50 52 53 60 60 61 61 62 63 63 65 66 70 75 80 85 86 87 88 88 88 108 111 112 112 112 112 113 134 136 139 141 141 143 144 148 148 150 150 152 159 164 166 171 174 174 174 174 178 180 183 185 187
27	<b>larg</b>	62 63 65 108 111
28	<b>macro</b>	20
29	<b>no</b>	64 112
30	<b>p-cond</b>	168 185 187
31	<b>p-p1</b>	150 151 160 180 183 183 187
32	<b>p-progn</b>	157 174 178 180 183
33	<b>pprint</b>	9 16
34	<b>pretty</b>	8
35	<b>prettyFtn</b>	9 178 180 183 185 187
36	<b>redpair</b>	63 108
37	<b>res</b>	108 113 114
38	<b>rev</b>	121 122 124
39	<b>symb</b>	13 68 73 78 83 88
40	<b>symeval</b>	11
41	<b>t+3</b>	150 153 162 170
42	<b>t-3</b>	150 155 167 176
43	<b>v</b>	50 51 51 51 108 111 112 113 141 142 142 142

---

44           **val**           44 44 44

45           **vars**          63 90 91 98 102 102 108 110 111

46           **x**            14 14 14 16 20 21 21 35 37 56 57 61 61 62 62 70 71 71 71 71 75  
76 76 76 76 80 81 81 81 81 85 86 86 86 87 90 91 91 91 116 117  
117 117 121 122 122 122 147 178 178 180 180 182 183 185 185  
187 187

47           **xx**          26 146

48           **y**            14 14 14 14 116 117 117

*;Samedi 6 Avril 2002 11:18:20 end of file : symeval.vlisp ;*