

```

1: 【Prolog(JunPrologInterpreter)マニュアル】           2009/04/23現在
2:
3: Dec-10のPrologの記法に準拠したPrologの処理系(インタプリタ)です。
4: すべてが前置記法で、op述語による前置／中置／後置などは定義できま
5: せんが、ユニフィケーションやバックトラックなどの機構を備えた立派
6: なPrologの処理系です。
7:
8: 定数
9:
10: Prologの定数には以下のようなものがあります。
11:
12:     記号
13:     数
14:     文字列
15:     リスト
16:     オブジェクト
17:
18: 記号は、小文字で始まる英数字です。もし、記号の中に大文字や特殊
19: 文字を含ませたい場合には、「'」と「'」で囲って記述します。「'」
20: を文字列の中に書きたい場合には「''」というように重ねて書きます。
21:     aoki
22:     atsushi
23:     symbol12345
24:     'AOKI Atsushi'
25:     'You are ''cool''.'
26:
27: 数は、Smalltalkの数オブジェクトと同様な書き方をします。
28:     123
29:     -123
30:     123.456
31:     -123.456
32:     123.456e7
33:     -123.456e-7
34:     16r123
35:
36: 文字列は、「"」と「"」で囲って記述します。「"」を文字列の中に
37: 書きたい場合には「"""」というように重ねて書きます。
38:     "AOKI Atushi"
39:     "You are ""cool""."
40:
41: リストは、「[]」「|」「,」「[]」を使用して表わします。空リストは
42: 「[]」のように角カッコを二つ合わせて書きます。ドットペアは
43: 「[AOKI|Atsushi]」のように表わします。「AOKI」が頭部(head)であり、
44: 「Atsushi」が尾部(tail)です。「|」の代わりに「,」を用いて
45: 「[AOKI,Atsushi]」と表わすと、「AOKI」が頭部になり、「[Atsushi]」
46: が尾部になります。「[AOKI,Atsushi]」と「[AOKI|Atsushi]」は同義
47: です。また、「[]」はSmalltalkオブジェクトの「nil」と同じです。
48:     []
49:     [AOKI|Atsushi]
50:     [AOKI,Atsushi]
51:     [AOKI|Atsushi]
52:
53: オブジェクトは、「{」と「}」で囲ってSmalltalkのメッセージ式を
54: 記述します。「{」と「}」で囲まれた部分は、まずSmalltalk内部で実
55: 行され、その実行結果がPrologの定数として扱われます。なお、「{」

```

```

56: は「{nil}」と同義です。
57:     {100@100}
58:     {Image fromUser}
59:     {'aaa'} at: 2 put: $z; yourself}
60:
61: 変数
62:
63:     変数は、大文字で始まる英数字です。また、「~」も変数で匿名変数
64:     と呼ばれます。
65:     AOKI
66:     Atsushi
67:     Variable1234
68:     ~
69:     匿名変数は、変数が必要だけれども、その変数の値がどうでもよい場
70:     合に使います。たとえば、誰がsmalltalkを好きであるのかを調べたい
71:     のだけれど、それが誰であるかを知る必要のない場合です。
72:     likes(aoki,smalltalk).
73:     likes(aoki,prolog).
74:     likes(watanabe,smalltalk).
75:     likes(watanabe,lisp).
76:     likes(sakoh,lisp).
77:     likes(sahara,prolog).
78:     ?- likes(~,smalltalk).
79:     yes
80:     ?- likes(~,prolog).
81:     yes
82:     ?- likes(~,lisp).
83:     yes
84:     ?- likes(~,c).
85:     no
86:     同様に、誰でもいいけどprologやlispを好きな人がいるかという質問
87:     「?- likes(~,prolog).」 「?- likes(~,lisp).」は成功するでしょうが、
88:     一方、誰でもいいけどcが好きな人がいるかという質問「?- likes(~,c).」
89:     は失敗し、誰もc言語が好きでないことが直ちに分かります。
90:     なぜ、匿名変数に「~(チルダ)」を使い、通常の「_(アンダーバー)」
91:     を使用しないのかというと、昔のSmalltalkの処理系は「_」を「←」と
92:     みなし、代入を表わす特殊な記号として扱っていました。そして、最新
93:     のSmalltalkは、「_」をメッセージ名の中に含めることができるように
94:     なっています。そのため、SmalltalkとPrologのインタフェースに配慮
95:     して、匿名変数を「~」にしています。
96:
97: システム述語
98:
99:     Prologのシステム内にあらかじめ組み込まれている述語をシステム述
100:    語と呼び、それらは以下の10個のグループに大別されます。
101:
102:    組み込みシステム述語(systemPredicatesNo0)
103:    基本的なシステム述語(systemPredicatesNo1)
104:    比較演算に関するシステム述語(systemPredicatesNo2)
105:    算術演算に関するシステム述語(systemPredicatesNo3)
106:    節や項に関するシステム述語(systemPredicatesNo4)
107:    高階のシステム述語(systemPredicatesNo5)
108:    入出力に関するシステム述語(systemPredicatesNo6)
109:    デバッグのためのシステム述語(systemPredicatesNo7)
110:    その他のシステム述語(systemPredicatesNo8)

```

```

111:         ユーザ定義のシステム述語(systemPredicatesNo9)
112:
113:     上記の10個のグループに大別されて組み込まれているシステム述語を
114:     それぞれ順番に説明していきます。
115:
116: 組み込みシステム述語(systemPredicatesNo0)
117:
118: ！
119:     カット(述語!)は常に成功する述語です。しかし、大きな副作用があ
120:     ります。目標を充足させるにあたって、後戻り(バックトラック)が生じ
121:     るような場合、その後戻りのための選択枝を再考察の必要のないもの
122:     として捨ててしまいます。そのため、解に無関係な再充足目標をあらかじ
123:     め除去しますので、実行の効率を上げることができます。また、後戻り
124:     のために使用されている記憶領域を解放しますから、メモリの節約にも
125:     なります。しかし、Prologの後戻りを故意に無効にする述語ですから、
126:     適切に使用することが肝要です。
127:     カットを適切に使用するケースは三つあると思われます。その第一は、
128:     目標に対する正しい方向が発見されたことをPrologに伝える場合です。
129:     第二は、ある目標を直ちに失敗させたい場合です。第三は、後戻りによ
130:     る別の解の生成を停止したい場合です。
131:
132: true
133:     常に成功する述語です。実際には必要のない述語ですが、便宜上存在
134:     しています。
135:
136: fail
137:     常に失敗する述語です。故意に後戻り(バックトラック)を起こすこと
138:     ができるので、使用する場面が多かろうと思います。たとえば、
139:     append(X,Y,[1,2,3])の目標に対するすべての解を求めたい場合には、
140:     次のようになります。
141:         output(X,Y) :- write(X), write(", "), write(Y), nl.
142:         ?- append(X,Y,[1,2,3]), output(X,Y), fail.
143:     このプログラムは、充足したXとYをoutput(X,Y)で出力させ、その後
144:     のfailによって後戻りを起こさせ、XとYに充足可能なすべての解を求めて
145:     います。
146:     また、カット(!)との組み合わせも有効です。「..., !, fail.」は
147:     「実行がここまで進んだらもうよい(もう結構)」ということを表わして
148:     います。たとえば、ある目標Gが充足したことを否定する述語not(G)は、
149:     カット(!)と失敗(fail)を組み合わせると、以下のように記述することが
150:     できます。
151:         not(G) :- call(G), !, fail.
152:         not(G).
153:     第一の節を実行している間に目標Gが失敗すれば、「..., !, fail.」
154:     に届く前に後戻りが起こり、第二の節の「not(G).」が充足され、目標G
155:     が成功したことになります。反対に、第一の節を実行している間に目標
156:     Gが成功すれば、それまでの選択枝が!によって除去され、その後のfail
157:     によって第一の節が失敗し、結局のところ目標Gが失敗したことになり
158:     ます。
159:
160: var(X)
161:     Xが代入されていない変数の時に成功します。
162:         ?- is(X,100), var(X).
163:         no
164:         ?- is(X,Y), var(X).
165:         X = X3 ,

```

```

166:         Y = X3
167:         yes
168:     ある変数が代入されていない時に、ユーザに入力をうながすには、
169:     「..., var(X), read(X), ...」というイディオムが便利です。
170:
171: send(X,Y,Z)
172: send(X,Y,Z,A)
173:     Smalltalkのメッセージを送信する述語です。Xをレシーバにして、Y
174:     というメッセージを、Zという引数リストを付けて送信し、その結果をA
175:     にユニファイします。レシーバに「self」の記号を指定すると、Prolog
176:     のインタプリタのインスタンス(a PrologInterpreter)がレシーバにな
177:     ります。
178:         image({Image fromUser}).
179:         point({50@50}).
180:         point({100@100}).
181:         point({150@150}).
182:         gc({ScheduledControllers activeController view graphicsContext}).
183:         ?- image(Image),
184:            point(Point),
185:            gc(GraphicsContext),
186:            send(Image,displayOn:at:,[GraphicsContext,Point],Image).
187:
188: 基本的なシステム述語(systemPredicatesNo1)
189:
190: repeat
191:     繰り返しを行なう述語です。
192: nonvar(X)
193:     Xが未代入の変数であれば失敗し、代入済みの変数もしくは定数なら
194:     ば成功します。
195:
196: integer(X)
197:     Xが整数ならば成功し、それ以外ならば失敗します。
198:
199: float(X)
200:     Xが単精度浮動小数点数ならば成功し、それ以外ならば失敗します。
201:
202: double(X)
203:     Xが倍精度浮動小数点数ならば成功し、それ以外ならば失敗します。
204:
205: fraction(X)
206:     Xが分数ならば成功し、それ以外ならば失敗します。
207:
208: number(X) :- send(self,number:[X]).
209:     Xが数ならば成功し、それ以外ならば失敗します。
210:
211: symbol(X) :- send(self,symbol:[X]).
212:     Xが記号ならば成功し、それ以外ならば失敗します。
213:
214: string(X) :- send(self,string:[X]).
215:     Xが文字列ならば成功し、それ以外ならば失敗します。
216:
217: list(X) :- send(self,list:[X]).
218:     Xがリストならば成功し、それ以外ならば失敗します。空リストは成
219:     功します。
220:

```

221: dotp(X)  
222: Xがセル(ドットペア)ならば成功し, それ以外ならば失敗します。  
223: 空リストは失敗します。  
224:  
225: atom(X)  
226: Xがアトムならば成功し, それ以外ならば失敗します。アトムとは,  
227: 記号, 文字列, 空リストのことです。  
228:  
229: atomic(X)  
230: Xがアトムまたは数ならば成功し, それ以外ならば失敗します。  
231:  
232: structure(X)  
233: Xがアトムまたは数ならば失敗し, それ以外ならば成功します。  
234:  
235: 比較演算に関するシステム述語(systemPredicatesNo2)  
236:  
237: ==(X,Y) :- send(X,=[Y]).  
238: SmalltalkでXとYが等しいとき成功し, それ以外ならば失敗します。  
239:  
240: ¥==(X,Y) :- ==(X,Y),!,fail.  
241: SmalltalkでXとYが等しいとき失敗し, それ以外ならば成功します。  
242:  
243: =(X,X)  
244: XとYが等しいとき成功し, それ以外ならば失敗します。  
245:  
246: ¥=(X,Y)  
247: XとYが等しいとき失敗し, それ以外ならば成功します。  
248:  
249: >(X,Y)  
250: XがYより大きいならば成功し, それ以外ならば失敗します。  
251:  
252: >=(X,Y)  
253: XがYより大きいか等しいならば成功し, それ以外ならば失敗します。  
254:  
255: <(X,Y)  
256: XがYより小さいならば成功し, それ以外ならば失敗します。  
257:  
258: =<(X,Y)  
259: XがYより小さいか等しいならば成功し, それ以外ならば失敗します。  
260:  
261: 算術演算に関するシステム述語(systemPredicatesNo3)  
262:  
263: +(X,Y,Z)  
264: XとYを加算したものがZと等しいならば成功し, それ以外ならば失敗  
265: します。  
266:  
267: -(X,Y,Z)  
268: XからYを減算したものがZと等しいならば成功し, それ以外ならば失  
269: 敗します。  
270:  
271: \*(X,Y,Z)  
272: XとYを乗算したものがZと等しいならば成功し, それ以外ならば失敗  
273: します。  
274:  
275: //(X,Y,Z)

276: XをYで除算して余りを切り捨てたものがZと等しいならば成功し、そ  
277: れ以外ならば失敗します。  
278:  
279: /(X,Y,Z)  
280: XをYで除算したものがZと等しいならば成功し、それ以外ならば失敗  
281: します。  
282:  
283: ¥¥(X,Y,Z)  
284: XをYでモジュロ演算したものがZと等しいならば成功し、それ以外な  
285: らば失敗します。  
286:  
287: is(X,Y)  
288: XとYが等しいとき成功し、それ以外ならば失敗します。  
289:  
290: is(Z,F(X,Y))  
291: 加減乗除において、XとYを加減乗除したものがZと等しいならば成功  
292: し、それ以外ならば失敗します。  
293:  
294: 節や項に関するシステム述語(systemPredicatesNo4)  
295:  
296: listing  
297: 登録されているすべての述語の節を出力し、常に成功します。  
298:  
299: listing(X)  
300: Xで指定される記号を述語としてもつ節を出力し、成功します。対応  
301: する述語がないならば失敗します。  
302:  
303: systemListing  
304: 登録されているすべてのシステム述語の節を出力し、常に成功します。  
305:  
306: systemListing(X)  
307: Xで指定される記号をシステム述語としてもつ節を出力し、成功しま  
308: す。対応するシステム述語がないならば失敗します。  
309:  
310: editing  
311: 登録されているすべての述語を編集するエディタを開き、常に成功し  
312: ます。  
313:  
314: consult  
315: Prologのプログラムを読み込むためのウィンドウが開き成功します。  
316: このウィンドウ内でPrologのプログラムを書き、acceptすると、書かれ  
317: た節を最後に追加します。  
318:  
319: consult(X)  
320: Xで指定されるファイルからPrologのプログラムを読み込み、読み込  
321: んだ節を最後に追加し、成功します。対応するファイルが存在しないな  
322: らば失敗します。  
323:  
324: reconsult  
325: Prologのプログラムを読み込むためのウィンドウが開き成功します。  
326: このウィンドウ内でPrologのプログラムを書き、acceptすると、書かれ  
327: た節をこれと同じ述語の節に置き換えます。  
328:  
329: reconsult(X)  
330: Xで指定されるファイルからPrologのプログラムを読み込み、読み込

```

331: んだ節をこれと同じ述語の節に置き換え, 成功します。対応するファイ
332: ルが存在しないならば失敗します。
333:
334: saving
335:   登録されているすべての述語の節を入力ダイアログで指定されたファ
336:   イルへ出力し, 常に成功します。
337:
338: saving(X)
339:   Xで指定される記号を述語としてもつ節を入力ダイアログで指定され
340:   たファイルへ出力し, 成功します。対応する述語がないならば失敗しま
341:   す。
342:
343: userPredicates(X)
344:   登録されているすべての述語をリストにしてXとユニファイします。
345:
346: systemPredicates(X)
347:   登録されているすべてのシステム述語をリストにしてXとユニファイ
348:   します。
349:
350: predicates([X|Y])
351:   登録されているすべての述語をリストにしてXとユニファイし, 登録
352:   されているすべてのシステム述語をリストにしてYとユニファイします。
353:
354: functor(T,F,A)
355:   この述語は, Tが関数子FとN個の引数をもつ構造を意味します。
356:   ?- functor(likes(aoki,smalltalk),F,N).
357:       F = likes ,
358:       N = 2
359:   yes
360:   ?- functor(aoki,F,N).
361:       F = aoki ,
362:       N = 0
363:   yes
364:
365: arg(N,S,T)
366:   この述語は, 構造SのN番目の引数がTであることを意味します。常に
367:   NとSが代入されていなければなりません。
368:   ?- arg(2,likes(aoki,smalltalk),A).
369:       A = smalltalk
370:   yes
371:
372: =..(X,L)
373:   この述語は「ユニブ」と呼ばれ, LはXの関数子と引数からなるリスト
374:   を意味します。
375:   ?- =..(likes(aoki,smalltalk),L).
376:       L = [likes,aoki,smalltalk]
377:   yes
378:   ?- =..(X,[likes,aoki,smalltalk]).
379:       X = likes(aoki,smalltalk)
380:   yes
381:
382: name(A,L)
383:   記号Aの文字列はリストLであることを意味します。
384:   ?- name(apple,L).
385:       L = "apple"

```

```

386:         yes
387:         ?- name(A,[97,112,112,108,101]).
388:         A = apple
389:         yes
390:         ?- name(A,"apple").
391:         A = apple
392:         yes
393:
394: remove
395:     登録されているすべての述語の節を削除します。この述語は常に成功
396:     します。
397:
398: remove(X)
399:     Xで指定される記号を述語としてもつ節を削除し、成功します。対応
400:     する述語がないならば失敗します。
401:
402: clause(X)
403:     Xで指定される節を登録されている節にユニファイします。
404:     concat([],X,X).
405:     concat([A|X],Y,[A|Z]) :-
406:         concat(X,Y,Z).
407:     ?- clause([[concat|Arguments]|Body]).
408:     Arguments = ([],X6,X6) ,
409:     Body = [] ;
410:     Arguments = ([A11|X11],Y11,[A11|Z11]) ,
411:     Body = concat(X11,Y11,Z11) ;
412:     no
413:     ?- clause([concat|Arguments]|Body).
414:     Arguments = ([],X6,X6) ,
415:     Body = [] ;
416:     Arguments = ([A11|X11],Y11,[A11|Z11]) ,
417:     Body = concat(X11,Y11,Z11) ;
418:     no
419:
420: asserta(X)
421:     Xで指定される節をその述語で示される節の先頭に加えます。
422:     ?- remove,
423:         asserta([likes(aoki,smalltalk)]),
424:         asserta([likes(watanabe,smalltalk)]),
425:         listing.
426:     likes(watanabe,smalltalk).
427:     likes(aoki,smalltalk).
428:     yes
429:
430: assert(X)
431: assertz(X)
432:     Xで指定される節をその述語で示される節の末尾に加えます。
433:     ?- remove,
434:         assertz([on(cat,dog)]),
435:         asserta([on(hen,cat)]),
436:         assertz([on(dog,donkey)]),
437:         assertz([above(X,Y) :- on(X,Y)]),
438:         assertz([above(X,Y) :- on(X,Z), above(Z,Y)]),
439:         listing.
440:     ?- above(X,dog).

```



```

441:         above(X,Y) :-
442:             on(X,Y).
443:         above(X,Y) :-
444:             on(X,Z),
445:             above(Z,Y).
446:         on(hen,cat).
447:         on(cat,dog).
448:         on(dog,donkey).
449:         X = X1 ,
450:         Y = Y1 ,
451:         Z = Z1
452:         yes
453:         ?- above(X,dog).
454:         X = cat ;
455:         X = hen ;
456:         no
457:
458: retract(X)
459:     Xで指定される節を登録されている節にユニファイし、成功すると、
460:     その節は削除されます。
461:     concat([],X,X).
462:     concat([A|X],Y,[A|Z]) :-
463:         concat(X,Y,Z).
464:     ?- listing(concat). % ちゃんと登録されているかな
465:     concat([],X,X).
466:     concat([A|X],Y,[A|Z]) :-
467:         concat(X,Y,Z).
468:     yes
469:     ?- retract([concat(|Arguments)|Body]). % 消すぞ
470:     Arguments = ([,X6,X6) ,
471:     Body = [] ;
472:     Arguments = ([A11|X11],Y11,[A11|Z11]) ,
473:     Body = concat(X11,Y11,Z11) ;
474:     no
475:     ?- listing(concat). % もうなくなっているはず
476:     no
477:
478: 高階のシステム述語(systemPredicatesNo5)
479:
480: call(G)
481:     Gを目標として充足できるならば成功となり、充足できないならば失敗
482:     となります。
483:
484: not(G)
485:     Gを目標として充足できるならば失敗となり、充足できないならば成
486:     功となります。
487:
488: or(X,Y)
489:     XとYの選言を表わします。Xが成功するかまたはYが成功すれば、選言
490:     は成功となります。Xが失敗の時に、Yの充足が試みられます。この時、
491:     Yも失敗すると選言が失敗をします。
492:
493: and(X,Y)
494:     XとYの連言を表わします。Xが成功しかつYも成功するならば、連言は
495:     成功となります。Xが成功したのに、Yが失敗すると、Xの再充足が試み

```

```

496: られます。
497:
498: 入出力に関するシステム述語(systemPredicatesNo6)
499:
500: read(X)
501:   入力ダイアログから入力をXとユニファイします。
502:
503: read(X,M)
504:   入力ダイアログから入力をXとユニファイします。その際に、入力ダ
505:   イアログのメッセージとしてMを使用します。
506:
507: write(X)
508:   Xを出力します。常に成功する述語です。
509:
510: nl
511:   復改を出力します。常に成功する述語です。
512:
513: tab(X)
514:   タブをXだけ出力します。Xが代入されていないならば失敗します。
515:
516: clear
517:   出力をきれいにします。常に成功する述語です。
518:
519: デバッグのためのシステム述語(systemPredicatesNo7)
520:
521: clock(X)
522:   現在の時刻をミリ秒単位でXにユニファイします。
523:
524: verbose(X)
525:   Xがtrueという記号の場合に、充足にかかった時間と試みたゴールの
526:   数を出力します。Xがtrueという記号以外ならば、時間とゴール数の出
527:   力を停止します。
528:   ?- remove.
529:   above(X,Y) :-
530:       on(X,Y).
531:   above(X,Y) :-
532:       on(X,Z),
533:       above(Z,Y).
534:   on(hen,cat).
535:   on(cat,dog).
536:   on(dog,donkey).
537:   ?- verbose(true).
538:   ?- above(X,donkey).
539:   ?- verbose(false).
540:   ?- remove.
541:   yes
542:   ?- verbose(true).
543:   <0 milliseconds, 1 goals>
544:   yes
545:   ?- above(X,donkey).
546:   <0 milliseconds, 4 goals>
547:   X = dog ;
548:   <0 milliseconds, 17 goals>
549:   X = hen ;
550:   <0 milliseconds, 35 goals>

```

```
551:      X = cat ;
552:      <16 milliseconds, 56 goals>
553:      no
554:      ?- verbose(false).
555:      yes
556:
557: gc
558:   ガベジコレクションを行ないます。この述語は常に成功します。
559:
560: inspect(X)
561:   Xをインスペクションします。この述語は常に成功します。
562:
563: spy(X)
564:   Xで指定される記号を述語としてもつ節のトレースを開始します。
565:
566: nospy(X)
567:   Xで指定される記号を述語としてもつ節のトレースを終了します。
568:
569: trace
570:   登録されているすべての節のトレースを開始します。
571:
572: notrace
573:   登録されているすべての節のトレースを終了します。
574:
575: その他のシステム述語(systemPredicatesNo8)
576:
577: append(X,Y,Z)
578:   XとYを接続したリストがZであることを意味します。
579:
580: member(X,Y)
581:   XがYのリストの要素であることを意味します。
582:
583: reverse(X,Y)
584:   XのリストのリバースがYのリストであることを意味します。
585:
586: length(X,Y)
587:   Xのリストの長さがYであることを意味します。
588:
589: nth(X,Y,Z)
590:   XのリストのY番目の要素がZであることを意味します。
591:
592: printlist(L)
593:   Lのリストの要素を順番に出力します。
594:
595: lispAppend(X,Y,Z)
596: lispReverse(X,Y)
597: lispMember(X,Y)
598: lispMember(X,Y,Z)
599: lispAssoc(X,Y)
600: lispAssoc(X,Y,Z)
601: lispNconc(X,Y,Z)
602:   これらはLispの関数を呼び出す述語で、高速ですが、後戻り(バック
603:  トラック)できません。
604:
605: ユーザ定義のシステム述語(systemPredicatesNo9)
```

606:  
 607: 現在、ユーザ定義のシステム述語はありませんが、皆さんで作成して、  
 608: ここに、その機能をお書き下さい。

609:  
 610: システム述語一覧

611:  
 612: ! (systemPredicatesNo0)  
 613: \* (systemPredicatesNo3)  
 614: + (systemPredicatesNo3)  
 615: - (systemPredicatesNo3)  
 616: / (systemPredicatesNo3)  
 617: // (systemPredicatesNo3)  
 618: < (systemPredicatesNo2)  
 619: = (systemPredicatesNo2)  
 620: =.. (systemPredicatesNo4)  
 621: =< (systemPredicatesNo2)  
 622: == (systemPredicatesNo2)  
 623: > (systemPredicatesNo2)  
 624: >= (systemPredicatesNo2)  
 625: and (systemPredicatesNo5)  
 626: append (systemPredicatesNo8)  
 627: arg (systemPredicatesNo4)  
 628: assert (systemPredicatesNo4)  
 629: asserta (systemPredicatesNo4)  
 630: assertz (systemPredicatesNo4)  
 631: atom (systemPredicatesNo1)  
 632: atomic (systemPredicatesNo1)  
 633: call (systemPredicatesNo5)  
 634: clause (systemPredicatesNo4)  
 635: clear (systemPredicatesNo6)  
 636: clock (systemPredicatesNo7)  
 637: consult (systemPredicatesNo4)  
 638: dotp (systemPredicatesNo1)  
 639: double (systemPredicatesNo1)  
 640: editing (systemPredicatesNo4)  
 641: fail (systemPredicatesNo0)  
 642: float (systemPredicatesNo1)  
 643: fraction (systemPredicatesNo1)  
 644: functor (systemPredicatesNo4)  
 645: gc (systemPredicatesNo7)  
 646: inspect (systemPredicatesNo7)  
 647: integer (systemPredicatesNo1)  
 648: is (systemPredicatesNo3)  
 649: length (systemPredicatesNo8)  
 650: lispAppend (systemPredicatesNo8)  
 651: lispAssoc (systemPredicatesNo8)  
 652: lispMember (systemPredicatesNo8)  
 653: lispNconc (systemPredicatesNo8)  
 654: lispReverse (systemPredicatesNo8)  
 655: list (systemPredicatesNo1)  
 656: listing (systemPredicatesNo4)  
 657: member (systemPredicatesNo8)  
 658: name (systemPredicatesNo4)  
 659: nl (systemPredicatesNo6)  
 660: nonvar (systemPredicatesNo1)

```
661:      nospy      (systemPredicatesNo7)
662:      not       (systemPredicatesNo5)
663:      notrace   (systemPredicatesNo7)
664:      nth       (systemPredicatesNo8)
665:      number    (systemPredicatesNo1)
666:      or        (systemPredicatesNo5)
667:      predicates (systemPredicatesNo4)
668:      printlist (systemPredicatesNo8)
669:      read      (systemPredicatesNo6)
670:      reconsult (systemPredicatesNo4)
671:      remove    (systemPredicatesNo4)
672:      repeat    (systemPredicatesNo1)
673:      retract   (systemPredicatesNo4)
674:      reverse   (systemPredicatesNo8)
675:      saving    (systemPredicatesNo4)
676:      send      (systemPredicatesNo0)
677:      spy       (systemPredicatesNo7)
678:      string    (systemPredicatesNo1)
679:      structure (systemPredicatesNo1)
680:      symbol    (systemPredicatesNo1)
681:      systemListing (systemPredicatesNo4)
682:      systemPredicates (systemPredicatesNo4)
683:      tab       (systemPredicatesNo6)
684:      trace     (systemPredicatesNo7)
685:      true      (systemPredicatesNo0)
686:      userPredicates (systemPredicatesNo4)
687:      var       (systemPredicatesNo1)
688:      verbose   (systemPredicatesNo7)
689:      write     (systemPredicatesNo6)
690:      ¥=        (systemPredicatesNo2)
691:      ¥==       (systemPredicatesNo2)
692:      ¥¥        (systemPredicatesNo3)
693:
694: Copyright (C) 1985-2009 AOKI Atsushi
```