数式処理 *J.JSSAC (2006)*
Vol. 12, No. 3, pp. 3 - 16

特別寄稿

中西流プログラミングを読み解く

和田 英一*

IIJ 技術研究所

2000 年 8 月
小川貴英君撮影

私は今，病気のこともあって，サバティカルをとっております．おかげで小さなプログラムを「凝って」作る楽しみに浸ることができています．bit のナノピコ教室で遊んでいた慶応のメンバーたちと会う機会がありました．昔，和田先生の音頭で「TULIPS」,「KING」,「KISS」,「WINK」が東大に集まったことが話題になりまして，再び会うことはできないだろうかという提案がありました．今，このような集まりを持っても生産的なことはないのかもしれませんが，プログラミングという作業が世の中から忘れ去られてしまう前に思い出話をするのもわるくないかな，などと思っています．

私は週に 1 度慶應病院に通っている状態で，ふだんは酒も飲めるようになっております (食べることだけがちょっと不自由ですが)．メールは家内のアドレスを通して交換できるようにしております．また中西を思い出していただければ幸いです．

こういう悲しいメイルを残し，中西君はわれわれの前から去っていった．本稿は中西君の追悼である．中西君の書いたきっぷ問題と Ackermann 関数のプログラムを読み，ハックの後を偲びたい．中西君といえば目でみる GC も有名だが [1]，プログラムにアクセス出来ぬし，また CPU が 6502 だと読む気も削がれる．

*wada@u-tokyo.ac.jp

きっぷ問題

慶応義塾大学理工学部教授の中西正和君 (1943-05-16〜2000-11-04) が情報処理学会誌のプログラムのページに「数楽オリンピックへのいじわる」を寄稿した [2]. 電車のきっぷの番号の 4 つの数字と加減乗除を使い 10 を作るゲームがある. これを全部求めるプログラムである. そしてこのゲームに終止符を打とうというものである.

私自身もこういうプログラムを何度も書いた. 特に最近は計算機が高速なので, 0 から 9 までの 4 数字の組合わせ ${}_{10}\mathrm{H}_4$ のそれぞれの 4! 通りの全順列を用意し, 4 項演算の 5 形式の 3 演算子に加減乗除をすべて与えて計算すればよい. ${}_{10}\mathrm{H}_4 \times 4! \times (5 \times 4^3) = 715 \times 24 \times 320 = 5491200$ 回の計算で終る. だからプログラムはあまり工夫せずに書いてしまう. しかし中西君のは不思議なプログラムであった. もともとは Fortran なのだが, C に書き直すと次のようになる. 説明用に行番号をつけた. これをこれから解読する.

```
 0 #include <stdio.h>
 1 #include <math.h>
 2 main (){
 3  float e[93];
 4  int ir[60], ib[3][6], ia[4][24], kc[24];
 5  float test=10.0,a,b,c,d,apb,amb,atb,cpd,cmd,ctd,bs,cs,ds,
 6    cpds,cmds,asb,bscpd,bscmd,ss,bpcs,bmcs;
 7  int kk,ll,mm,nn,i,j,k,l,m,n,ka,kb;
 8  printf("\n");
 9  for(kk=1;kk<=10;kk++)
10   for(ll=kk;ll<=10;ll++)
11    for(mm=ll;mm<=10;mm++)
12     for(nn=mm;nn<=10;nn++){
13      k=kk-1;l=ll-1;m=mm-1;n=nn-1;
14      printf("********%3d%3d%3d%3d ********\n",k,l,m,n);
15      for(j=0;j<3;j++){ib[j][j]=l;ib[j][j+3]=l;}
16      ib[1][0]=m;ib[0][1]=m;ib[0][2]=m;ib[2][3]=m;ib[2][4]=m;ib[1][5]=m;
17      ib[2][0]=n;ib[2][1]=n;ib[1][2]=n;ib[1][3]=n;ib[0][4]=n;ib[0][5]=n;
18      n=0;
19      for(l=0;l<6;l++)
20       for(ka=0;ka<4;ka++){
21        kb=0;
22        for(m=0;m<4;m++)
23         if(m==ka)ia[m][n]=k;else{ia[m][n]=ib[kb][l];kb=kb+1;}
24        n=n+1;}
```

```
25      /* permutation of a,b,c,d*/
26      for(l=0;l<24;l++){
27       kc[l]=0;
28       for(m=0;m<4;m++)kc[l]=kc[l]*10+ia[m][l];}
29      for(l=0;l<23;l++){
30       m=l+1;
31       j=kc[l];
32       if(l>=0)for(n=m;n<24;n++)if(kc[n]==j)kc[n]=-1;}
33      /* start testing */
34      for(kb=0;kb<24;kb++)if(kc[kb]>=0){
35       a=ia[0][kb]; b=ia[1][kb]; c=ia[2][kb]; d=ia[3][kb];
36       apb=a+b; amb=a-b; atb=a*b; cpd=c+d; cmd=c-d; ctd=c*d;
37       bs=b; cs=c; ds=d;
38       if(b==0.0)bs=-3.1414213;
39       if(c==0.0)cs=2.2361592;
40       if(d==0.0)ds=-5.2101415;
41       cpds=cpd;cmds=cmd;asb=a/bs;bscpd=b/cs+d;bscmd=b/cs-d;ss=-12.36067;
42       if(cpd==0.0)cpds=ss;
43       if(cmd==0.0)cmds=ss;
44       if(bscpd==0.0)bscpd=ss;
45       if(bscmd==0.0)bscmd=ss;
46       bpcs=b+c;bmcs=b-c;
47       if(bpcs==0.0)bpcs=ss;
48       if(bmcs==0.0)bmcs=ss;
49      /* evaluate expressions */
50       e[0] = apb + cpd;              e[1] = apb + cmd;
51       e[2] = (apb + c) * d;          e[3] = (apb + c) / ds;
52       e[4] = apb - cpd;              e[5] = (apb - c) * d;
53       e[6] = (apb - c) / ds;         e[7] = apb * c + d;
54       e[8] = apb * c - d;            e[9] = apb * ctd;
55       e[10] = apb * c / ds;          e[11] = apb / cs + d;
56       e[12] = apb / cs - d;          e[13] = apb / cs / ds;
57       e[14] = (amb - c) * d;         e[15] = amb * c + d;
58       e[16] = amb * c - d;           e[17] = amb * c * d;
59       e[18] = amb * c / ds;          e[19] = amb / cs + d;
60       e[20] = atb + cpd;             e[21] = atb + cmd;
61       e[22] = (atb + c) * d;         e[23] = (atb + c) / ds;
```

```
62      e[24] = atb - cpd;              e[25] = (atb - c) * d;
63      e[26] = (atb - c) / ds;         e[27] = atb * c + d;
64      e[28] = atb * c - d;            e[29] = atb * ctd;
65      e[30] = atb * c / ds;           e[31] = atb / cs + d;
66      e[32] = atb / cs - d;           e[33] = atb / cs / ds;
67      e[34] = asb + cpd;              e[35] = asb + cmd;
68      e[36] = (asb + c) * d;          e[37] = (asb + c) / ds;
69      e[38] = (asb - c) * d;          e[39] = asb / cs + d;
70      e[40] = apb - ctd;              e[41] = apb - c / ds;
71      e[42] = apb * cpd;              e[43] = apb * cmd;
72      e[44] = apb / cpds;             e[45] = apb / cmds;
73      e[46] = amb * cmd;              e[47] = atb + ctd;
74      e[48] = atb + c / ds;           e[49] = atb - ctd;
75      e[50] = atb - c / ds;           e[51] = atb / cpds;
76      e[52] = atb / cmds;             e[53] = asb + c / ds;
77      e[54] = a / bscpd;              e[55] = a / bscmd;
78      e[56] = (a - b * c) * d;        e[57] = (a - b / cs) * d;
79      e[58] = a / bpcs + d;           e[59] = a / bmcs + d;
80      e[60] = - a / bscmd;
81     /* start testing */
82      k=0;
83      for(i=0;i<61;i++)if(fabs(e[i]-test)<0.0001){ir[k]=i;k=k+1;}
84      if(k>0){
85       for(n=0;n<4;n++)printf("%2d ",ia[n][kb]);
86       for(j=0;j<k;j++)printf("%3d ",ir[j]);printf("\n");}}}}
```

「元のプログラムには演算時間短縮のための工夫がしてあったが，掲載するページの関係をその部分を削除し，その結果 IBM 7040 WATFOR で 11 分 30 秒かかった」そうだ．この工夫も知りたいところである．

ここでは彼の考えを追跡してみる．計算は実数で行なう．プログラム 5 行目 `test=10.0` は 10 以外の値を作ることにも対応している．9 行目から制御変数 `kk`，`ll`，`mm`，`nn` を `kk` は 1 から，あとは直前の制御変数の値から 10 まで変えて，4 つの数の全組合わせを作る．Fortran だから 1 から始めたのか．実際に使うのはそれからそれぞれ 1 を引いた `k`，`l`，`m`，`n` である (13 行目)．しかしこれらの変数は順列の表に入れると他の仕事に使われている．

順列の生成

15 行目からこの 4 つの数の順列を作る．学会誌のプログラムのページには中西君の permutation のプログラムも載っている [3]．これは中西君の LISP 入門 [4] と同様で，M 式 Lisp で書いて

ある.

```
comb1[a;x;y]=
        [null[y]->cons[append[a;
                cons[x;y]];nil];
        t->cons[append[a;cons[car[y];nil]];
            comb1[append[a;cons[car[y];nil]];
                x;cdr[y]]]];

comb2[x;y]=[null[y]->nil;
        t->append[comb1[nil;x;car[y]];
            comb2[x;cdr[y]]]];

perm[x]=[null[cdr[x]]->cons[x;nil];
        t->comb2[car[x];perm[cdr x]]]].
```

これは LISP 入門にある順列生成と同じやり方で, (a b c) の順列は (b c) の順列 ((b c) (c b)) のそれぞれに (comb2 のループで分解), a を (comb1 のループで) 先頭, 中間, 末尾に分配する方式である*. 上の Lisp のプログラムは再帰的に書いてあるが, 話題のプログラムでは 4 変数の順列だし, Fortran なので直接書いた. まず l, m, n の 3 個の順列は数え上げる (15〜17 行目). その結果配列 ib は以下のようになる. ここでも 2 項の順列 ((m n) (n m)) に l を分配しているのが見て取れる.

```
ib 0 1 2 3 4 5
0  l m m l n n
1  m l n n l m
2  n n l m m l
```

次はこのそれぞれの列に k を分配して 4 変数の順列を配列 ia に作る (18 〜24 行目).

```
ia 0 1 2 3 4 5 6 7 8 9...23
0  k l l l k m m m k m    n
1  l k m m m k l l m k    m
2  m m k n l l k n n n    l
3  n n n k n n n k l l    k
```

*Knuth の TAOCP 4 巻 2 分冊には Plain Change の順列生成法としてこういう方法が書いてある

ところで k, l, m, n のなかに 0, 0, 0, 0 のように重複があると, 順列にも同じものが出てきて, 再度おなじチェックはしたくない. それを調べるのが kc である. まず 26 行目〜28 行目で順列を 10 進法の数に変換し kc へ置く. 次に kc のなかに, 同じ数値が以前にあればその値を −1 にする (32 行目). −1 に対応の順列は以下でテストを省略する. 34 行目からいよいよテスト開始である. まずすべての順列について, kc の値が >= 0 なら (0, 0, 0, 0 はテストしなければならない) 配列 ia から kb 番目の要素を取り出し, 実数 a, b, c, d とする (35 行目).

計算式の最小集合

中西方式では 320 通りの計算はしない. プログラムの終りの方に 61 個の配列 e にいろいろな式で値を計算し, それを入れておき, あとでそのそれぞれが 10 に近いかどうか調べている. 結合則も交換則も考慮しない全 320 通りにはたしかに同じ計算もあるわけで, 当時のおそい計算機ではなんとか同じものは排除して, 計算時間を短縮しようと努力したのはわかる.

その 61 通りの式を以下に示す. 番号は配列 e の添字と一致している.

```
 0 (+ (+ a b) (+ c d))   1 (+ (+ a b) (- c d))   2 (* (+ (+ a b) c) d)
 3 (/ (+ (+ a b) c) d)   4 (- (+ a b) (+ c d))   5 (* (- (+ a b) c) d)
 6 (/ (- (+ a b) c) d)   7 (+ (* (+ a b) c) d)   8 (- (* (+ a b) c) d)
 9 (* (+ a b) (* c d))  10 (/ (* (+ a b) c) d)  11 (+ (/ (+ a b) c) d)
12 (- (/ (+ a b) c) d)  13 (/ (/ (+ a b) c) d)  14 (* (- (- a b) c) d)
15 (+ (* (- a b) c) d)  16 (- (* (- a b) c) d)  17 (* (* (- a b) c) d)
18 (/ (* (- a b) c) d)  19 (+ (/ (- a b) c) d)  20 (+ (* a b) (+ c d))
21 (+ (* a b) (- c d))  22 (* (+ (* a b) c) d)  23 (/ (+ (* a b) c) d)
24 (- (* a b) (+ c d))  25 (* (- (* a b) c) d)  26 (/ (- (* a b) c) d)
27 (+ (* (* a b) c) d)  28 (- (* (* a b) c) d)  29 (* (* a b) (* c d))
30 (/ (* (* a b) c) d)  31 (+ (/ (* a b) c) d)  32 (- (/ (* a b) c) d)
33 (/ (/ (* a b) c) d)  34 (+ (/ a b) (+ c d))  35 (+ (/ a b) (- c d))
36 (* (+ (/ a b) c) d)  37 (/ (+ (/ a b) c) d)  38 (* (- (/ a b) c) d)
39 (+ (/ (/ a b) c) d)  40 (- (+ a b) (* c d))  41 (- (+ a b) (/ c d))
42 (* (+ a b) (+ c d))  43 (* (+ a b) (- c d))  44 (/ (+ a b) (+ c d))
45 (/ (+ a b) (- c d))  46 (* (- a b) (- c d))  47 (+ (* a b) (* c d))
48 (+ (* a b) (/ c d))  49 (- (* a b) (* c d))  50 (- (* a b) (/ c d))
51 (/ (* a b) (+ c d))  52 (/ (* a b) (- c d))  53 (+ (/ a b) (/ c d))
54 (/ a (+ (/ b c) d))  55 (/ a (- (/ b c) d))  56 (* (- a (* b c)) d)
57 (* (- a (/ b c)) d)  58 (+ (/ a (+ b c)) d)  59 (+ (/ a (- b c)) d)
60 (- (/ a (- (/ b c) d)))
```

中西君がどうしてここに辿り着いたかを知りたく, 私も実質的な計算はどのくらいあるかと思って列挙してみたのが次の表である. 最初は 2 変数の場合で 4 通りしかない. 式の右に付随する

(0 . 18) のようなものは， 変数を 0 から 9 まで変えた場合にこの式のとりうる値の範囲である．

2 変数の次に 18 通りの 3 変数の式を並べた． さらにその下が 96 通りの 4 変数のリストである．4 変数の式の左端の数は対応する中西君の式の番号である． 式の番号のないものは式の値が 10 に満たないものである． 10 に満たないものは 36 通りある． その他に番号がかっこに入っている式が 3 つある． これらはかっこ内の番号を等価な式である．

```
(+ a b)   (0 . 18)        (* a b)   (0 . 81)
(- a b)   (−9 . 9)        (/ a b)   (0 . 9)

(+ (+ a b) c))   (0 . 27)        (* (* a b) c))   (0 . 729)
(+ (* a b) c))   (0 . 90)        (* (+ a b) c))   (0 . 162)
(+ (/ a b) c))   (0 . 18)        (* (- a b) c))   (−81 . 81)
(- (+ a b) c))   (−9 . 18)       (/ (* a b) c))   (0 . 81)
(- (- a b) c))   (−18 . 9)       (/ (/ a b) c))   (0 . 9)
(- (* a b) c))   (−9 . 81)       (/ (+ a b) c))   (0 . 18)
(- (/ a b) c))   (−9 . 9)        (/ (- a b) c))   (−9 . 9)
(- a (* b c)))   (−81 . 9)       (/ a (+ b c)))   (0 . 9)
(- a (/ b c)))   (−9 . 9)        (/ a (- b c)))   (−9 . 9)

 0 (+ (+ (+ a b) c) d)   (0 . 36)        29 (* (* (* a b) c) d)   (0 . 6561)
20 (+ (+ (* a b) c) d)   (0 . 99)         9 (* (* (+ a b) c) d)   (0 . 1458)
34 (+ (+ (/ a b) c) d)   (0 . 27)        17 (* (* (- a b) c) d)   (−729 . 729)
27 (+ (* (* a b) c) d)   (0 . 738)        2 (* (+ (+ a b) c) d)   (0 . 243)
 7 (+ (* (+ a b) c) d)   (0 . 171)       22 (* (+ (* a b) c) d)   (0 . 810)
15 (+ (* (- a b) c) d)   (−81 . 90)      36 (* (+ (/ a b) c) d)   (0 . 162)
31 (+ (/ (* a b) c) d)   (0 . 90)         5 (* (- (+ a b) c) d)   (−81 . 162)
39 (+ (/ (/ a b) c) d)   (0 . 18)        14 (* (- (- a b) c) d)   (−162 . 81)
11 (+ (/ (+ a b) c) d)   (0 . 27)        25 (* (- (* a b) c) d)   (−81 . 729)
19 (+ (/ (- a b) c) d)   (−9 . 18)       38 (* (- (/ a b) c) d)   (−81 . 81)
58 (+ (/ a (+ b c)) d)   (0 . 18)        56 (* (- a (* b c)) d)   (−729 . 81)
59 (+ (/ a (- b c)) d)   (−9 . 18)       57 (* (- a (/ b c)) d)   (−81 . 81)
47 (+ (* a b) (* c d))   (0 . 162)       42 (* (+ a b) (+ c d))   (0 . 324)
48 (+ (* a b) (/ c d))   (0 . 90)        43 (* (+ a b) (- c d))   (−162 . 162)
53 (+ (/ a b) (/ c d))   (0 . 18)        46 (* (- a b) (- c d))   (−81 . 81)
 1 (- (+ (+ a b) c) d)   (−9 . 27)       30 (/ (* (* a b) c) d)   (0 . 729)
21 (- (+ (* a b) c) d)   (−9 . 90)       10 (/ (* (+ a b) c) d)   (0 . 162)
35 (- (+ (/ a b) c) d)   (−9 . 18)       18 (/ (* (- a b) c) d)   (−81 . 81)
```

4	(- (- (+ a b) c) d)	(−18 . 18)	33	(/ (/ (* a b) c) d)	(0 . 81)
	(- (- (- a b) c) d)	(−27 . 9)		(/ (/ (/ a b) c) d)	(0 . 9)
24	(- (- (* a b) c) d)	(−18 . 81)	13	(/ (/ (+ a b) c) d)	(0 . 18)
	(- (- (/ a b) c) d)	(−18 . 9)		(/ (/ (- a b) c) d)	(−9 . 9)
	(- (- a (* b c)) d)	(−90 . 9)		(/ (/ a (+ b c)) d)	(0 . 9)
	(- (- a (/ b c)) d)	(−18 . 9)		(/ (/ a (- b c)) d)	(−9 . 9)
28	(- (* (* a b) c) d)	(−9 . 729)	3	(/ (+ (+ a b) c) d)	(0 . 27)
8	(- (* (+ a b) c) d)	(−9 . 162)	23	(/ (+ (* a b) c) d)	(0 . 90)
16	(- (* (- a b) c) d)	(−90 . 81)	37	(/ (+ (/ a b) c) d)	(0 . 18)
32	(- (/ (* a b) c) d)	(−9 . 81)	6	(/ (- (+ a b) c) d)	(−9 . 18)
	(- (/ (/ a b) c) d)	(−9 . 9)		(/ (- (- a b) c) d)	(−18 . 9)
12	(- (/ (+ a b) c) d)	(−9 . 18)	26	(/ (- (* a b) c) d)	(−9 . 81)
	(- (/ (- a b) c) d)	(−18 . 9)		(/ (- (/ a b) c) d)	(−9 . 9)
	(- (/ a (+ b c)) d)	(−9 . 9)		(/ (- a (* b c)) d)	(−81 . 9)
	(- (/ a (- b c)) d)	(−18 . 9)		(/ (- a (/ b c)) d)	(−9 . 9)
	(- a (* (* b c) d))	(−729 . 9)		(/ a (+ (+ b c) d))	(0 . 9)
	(- a (* (+ b c) d))	(−162 . 9)		(/ a (+ (* b c) d))	(0 . 9)
(15)	(- a (* (- b c) d))	(−81 . 90)	54	(/ a (+ (/ b c) d))	(0 . 81)
	(- a (/ (* b c) d))	(−81 . 9)		(/ a (- (+ b c) d))	(−9 . 9)
	(- a (/ (/ b c) d))	(−9 . 9)		(/ a (- (- b c) d))	(−9 . 9)
	(- a (/ (+ b c) d))	(−18 . 9)		(/ a (- (* b c) d))	(−9 . 9)
(19)	(- a (/ (- b c) d))	(−9 . 18)	55	(/ a (- (/ b c) d))	(−81 . 81)
	(- a (/ b (+ c d)))	(−9 . 9)		(/ a (- b (* c d)))	(−9 . 9)
(59)	(- a (/ b (- c d)))	(−9 . 18)	60	(/ a (- b (/ c d)))	(−81 . 81)
40	(- (+ a b) (* c d))	(−81 . 18)	51	(/ (* a b) (+ c d))	(0 . 81)
41	(- (+ a b) (/ c d))	(−9 . 18)	52	(/ (* a b) (- c d))	(−81 . 81)
49	(- (* a b) (* c d))	(−81 . 81)	44	(/ (+ a b) (+ c d))	(0 . 18)
50	(- (* a b) (/ c d))	(−9 . 81)	45	(/ (+ a b) (- c d))	(−18 . 18)
	(- (/ a b) (* c d))	(−81 . 9)		(/ (- a b) (+ c d))	(−9 . 9)
	(- (/ a b) (/ c d))	(−9 . 9)		(/ (- a b) (- c d))	(−9 . 9)

上の表は左右 2 段になっている． 左右は+と*， -と/を交換した形式になっている． 2， 3， 4 変数の各表で， 10 が作れる， 作れないの行が左右で揃うのは不思議である．また加減算が並ぶ場合は減算を後に， 乗除算が並ぶ場合は除算を後に， さらになるべくスタックは使わないように式を書いてある． これは私が普段やっている方法に従っただけである．

中西方式では `a + b`， `a - b` など何度も現れそうなので， それらはあらかじめ計算し， `apb`, `amb` などに置いておく．

問題は 0 による除算で， b， c， d など分母になる可能性のあるものは， bs， cs， ds などに整数値からかなり離れた数をいれておき， 除算にはこの方を使う． asb(なぜ adb でないか，a slash b のつもりか)

変わっているのは 60 番で， 普通なら (/ a (- b (/ c d))) とあるべきところ (- (/ a (- (/ b c) d))) と書いている． この式は大変重要で 1， 1，5， 8 は (/ 8 (- 1 (/ 1 5))) でしか 10 が作れない． a の前にマイナスがついているのはこれだけだが， (- (/ b c) d) がすでに bscmd として計算してあるので， それを再利用したかったのではないかと思っている．

こういう準備をしておいて 50 行目からいろいろな式を計算し， 結果を配列 e にいれる． 例えば e[0] には a + b + c + d を置く． こうして e[60] まで 61 種の値をおき， 最後は 83 行目で結果が 10 に充分近いか見る．

なお最初の宣言をみると e の配列は 93 個とってある． それは「10 以外の任意の整数用にするときは， あと 32 個の代入文 (決して 10 にならない式) を加え， TEST の値を与えるようにすればよい」と書いてあるためだ．

式の変形による最小集合の作成

320 通りの計算式を作り， 結合則， 交換則を利用し， 式を変形して上の 96 通りを得ることもできる． しかしかっこをつけた等価な 3 式を見つけるのは難しい．

```
(setq ops '(+ - * :))
(mapcar ops '(lambda (x)
 (mapcar ops '(lambda (y)
  (mapcar ops '(lambda (z)
   (princ `(,x (,y (,z @ @) @) @)) (terpri)
   (princ `(,x (,y @ (,z @ @)) @)) (terpri)
   (princ `(,x (,y @ @) (,z @ @))) (terpri)
   (princ `(,x @ (,y (,z @ @) @))) (terpri)
   (princ `(,x @ (,y @ (,z @ @)))) (terpri)))))))
```

これで変数を@で表示した． 全順列を使うから順番は問題とならないからだ．

```
(+ (+ (+ @ @) @) @)
(+ (+ (- @ @) @) @)
(+ (+ (* @ @) @) @)
(+ (+ (: @ @) @) @)
...
```

のような式が得られる． utilisp の match の機能を利用するので， 除算はエスケープ用の back-

slash(\) ではなく colon(:) にしてある． match の clause は加減と乗除で対称なので， 半分しか記載しない．

```
(defun mod3 (x) ;3 変数の式の処理
  (lets ((y (match x
    (('+ ('- a b) c) `(- (+ ,a ,c) ,b))
    (('+ '@ (b c d)) `(+ (,b ,c ,d) @))
    (('- a ('+ b c)) `(- (- ,a ,b) ,c))
    (('- a ('- b c)) `(+ (- ,a ,b) ,c))
    ; 乗除についても同様
    (t x))))
   (cond ((equal x y) y) (t (mod3 y)))))

(defun mod22 (x) ;(+ (+ @ @) (* @ @)) のようなパターンの処理
  (match x
    (('+ ('+ '@ '@) a) `(+ (+ ,a @) @))
    (('+ ('- '@ '@) a) `(- (+ ,a @) @))
    (('+ a ('+ '@ '@)) `(+ (+ ,a @) @))
    (('+ a ('- '@ '@)) `(- (+ ,a @) @))
    (('+ (': '@ '@) ('* '@ '@)) `(+ (* @ @) (: @ @)))
    (('+ (a '@ '@) (b '@ '@)) `(+ (,a @ @) (,b @ @)))
    (('- ('- '@ '@) a) `(- (- @ ,a) @))
    (('- a ('+ '@ '@)) `(- (- ,a @) @))
    (('- a ('- '@ '@)) `(- (+ ,a @) @))
    ; 乗除についても同様
    (t x)))

(defun modsub (x)
  (cond ((null x) x)
        ((atom (cadr x)) (list (car x) (cadr x) (mod3 (caddr x))))
        ((atom (caddr x)) (list (car x) (mod3 (cadr x)) (caddr x)))
        (t (mod22 x))))

(defun mod (foo)
(setq exs '())
(mapcar foo '(lambda (ex0) (princ ex0)
(lets ((ex1 (mod22 ex0))
```

```
        (ex2 (modsub ex1))
        (ex3 (match ex2
         (('+ '@ a) `(+ ,a @))
         (('* '@ a) `(* ,a @))
         (t ex2)))
        (ex4 (mod3 ex3))
        (ex5 (modsub ex4)))
   (princ ex5) (terpri)
(setq z (assoc ex5 exs))
(cond (z nil)
      (t (setq exs (cons (list ex5) exs)))))))
(mapcar (reverse exs) '(lambda (x) (princ x) (terpri))))
```

320 通りのすべての組を生成し，それでテストするのは簡単である．しかし中西流は必要最小限の式を探し，それだけのテストをする．これだけの式で果たして充分であるか，確信をもっていうのは大変である．私なりに 61 個 (10 のできないのも入れて 93 個) が得られたがなかなかここに収束しなかった．

Ackermann 関数

原始帰納的関数ではない帰納的関数として知られている Ackermann 関数は

```
(define (A x y)
  (cond ((= x 0) (+ y 1))
        ((= y 0) (A (- x 1) 1))
        (else (A (- x 1) (A x (- y 1))))))
```

と定義する[†]．中西君は Ackermann 関数でも再帰を使わずに $0 \le x \le 10, 0 \le y$ の値を得るプログラムを Fortran で書いた．私はこのプログラムを証明の例に使った [5]．このプログラムはプログラムのページには見当たらない．どこから探してきたか今は分からない．

```
      INTEGER FUNCTION F(IX,IY)
      DIMENSION M(21)
      DO 1 I=1,21
    1 M(I)=1-2*MOD(I,2)
    2 F=M(1)+2
      DO 3 I=1,21,2
      M(I)=M(I)+1
```

[†]Sussman の本の Ackermann 関数は多少違う形である．

```
      IF(I.GT.2*IX.OR.M(I).LT.M(I+1))IF(M(2*IX+1)-IY)2,4,4
    3 M(I+1)=F
    4 RETURN
      END
```

配列 M は奇数番目と偶数番目を異なる目的に使っているので，奇数を ys，偶数を bs とし，また返り値を置く F を bs[0] とする．bs[i]=A(ys[i],i) の関係にある．つまり ys は bs の y 座標を示す．以下の左上が C に書き直したプログラム，右はプログラム 6 行目の直前のスナップショット，左下は Ackermann 関数の表である．

```
 0 nack(int x,int y){
 1  int i,ys[11],bs[11];
 2  for(i=0;i<11;i++)ys[i]=-1;
 3  for(i=1;i<11;i++)bs[i]=1;
 4 l1: bs[0]=ys[0]+2;i=0;
 5 l2: ys[i]=ys[i]+1;
 6   if((i>=x)||(ys[i]<bs[i+1]))
 7   {if(ys[x]>=y)goto l3; else goto l1;}
 8   bs[i+1]=bs[0];i=i+1;if(i<=10)goto l2;
 9 l3:return(bs[0]);}
```

\x	0	1	2	3
y	bs0	bs1	bs2	bs3
0	1	2	3	5
1	2	3	5	13
2	3	4	7	29
3	4	5	9	61
4	5	6	11	125
5	6	7	13	253
6	7	8	15	...
7	8	9	17	...
8	9	10	19	...

	F	M1	M2	M3	M4	M5
	i	bs0	ys0	bs1	ys1	bs2	ys2
0	0	1	0	1	-1	1	-1
1	0	2	1	1	-1	1	-1
2	1	2	1	2	0	1	-1
3	0	3	2	2	0	1	-1
4	1	3	2	3	1	1	-1
5	2	3	2	3	1	3	0
6	0	4	3	3	1	3	0
7	1	4	3	4	2	3	0
8	0	5	4	4	2	3	0
9	1	5	4	5	3	3	0
10	2	5	4	5	3	5	1
11	0	6	5	5	3	5	1
12	1	6	5	6	4	5	1
13	0	7	6	6	4	5	1
14	1	7	6	7	5	5	1
15	2	7	6	7	5	7	2
16	0	8	7	7	5	7	2
17	1	8	7	8	6	7	2
18	0	9	8	8	6	7	2
19	1	9	8	9	7	7	2
20	2	9	8	9	7	9	3

$A(2,3)$ を計算したいとする．上の表から 9 である．しかしそれはその上の $A(2,2)$ が 7 なので，$A(1,7)$ が 9 だからである．それはさらにその上の $A(1,6)$ が 8 なので，$A(0,8)$ から分かる．一方は $A(2,2)$ が 7 なのは $A(2,1)$ を知らなければならない．

この事情をスナップショットで見ると，15 行目から bs2 は 7，ys2 は 2 で，$A(2,2) = 7$ を示し，ys1 が 7 になるのを待つ．bs1 は 7，8，9 と増え，同時に ys1 も 5，6，7 と増える．(C プログラムの 5 行目で増やしている．) ここで ys1 が 7 になったので，bs1(=9) が bs2 の値として使える．C プログラムでは (8 行目で)bs[0] からコピーしている．bs1 が 19 行目で 9 になったのは，17 行目から bs1=8，ys1=6 で待っていたところ，18 行目で bs0=9，ys0=8 となったからである．

中西流は bs[i] が決ったとすると，ys[i] を 1 増やす．隣りの bs[i+1] は ys[i] が増えて同じになるのを待っているのが，ys[i]<bs[i+1] ならまだ駄目ということで，bs[0] の増加 (4 行目) へ戻る．同じになれば bs[i+1] の値は確定したので更新し，次の i の準備をし，l2 へ戻り，ys も 1 だけ増やす．

あとは初期化のところを注意する．Ackermann 関数の定義から，$y = 0$ のとき，1 行左の $y = 1$ の値を使うことになっている．そこで bs は 1 を待っているということで 1 に初期化，初期化に続いて ys を増やして 0 にする必要があり，そのため ys は −1 に初期化しておく．

これで分かるように，中西プログラムは状態を配列 bs，ys に保ちつつ，Ackermann 関数をボトムアップに計算していく．

中西流プログラミングの総括

2 つのプログラムを読んだだけで総括するのも気が引けるが，いちおうまとめておく．1960 年代の終り頃の計算機はまだ結構遅く，記憶容量もたいしたことはなかった．60 年代後半に使われた科学用，制御用計算機 Facom 270-20 を見ると 1 語 16 ビットで加減算は 4.8 μ秒，乗算 20 μ秒，除算 38 μ秒，記憶容量は最大 32K 語であった．また中西君が KLISP を実装した東芝の Tosbac 3400-30 は京大で開発した KT-Pilot を商用化したもので，1 語 24 ビット，加減算は 7.8 μ秒，記憶容量は最大 256K 語であった．(KLISP は 16K 語の計算機に実装した)

またプログラム言語はアセンブリ言語か再帰呼出しも出来ない (最適化コンパイラもない)Fortran が主流で，Lisp は IBM の計算センターで (か手作りして) 使える程度であった．

従って計算に時間がかかるのを我慢するか，プログラムを思いきってハックしなければならなかった．その一例が中西君のきっぷ問題である．これだけの式でよいという結論を得るにはかなり時間がかかったであろう．しかし徹底的にオプティマイズするのが中西方式であった．ただオプティマイズには虫を取り込む危険性が常につきまとう．正当性の証明も少し大きいプログラムになると実用にならない．それを敢えて実行するのが中西君の身上であった．

中西君の FORTRAN プログラムは当然ながら再帰呼出しをせず，状態をテーブルで覚えておく方式である．これは Knuth の TAOCP の流儀にも似ている．Knuth はアルゴリズムの効率を正確に計算するため機械語を使った．

中西君は LISP は M-式で書いた．KLISP に M-式を読み込む機能があったから．

プログラムハックが語り合えないのは残念至極である．

参 考 文 献

[1] 中西正和: 目でみる GC, ゴミ集め (ガーベジコレクション) の基礎と動向チュートリアル資料, 1992 年 11 月 20 日 情報処理学会.

[2] 中西正和: 数楽オリンピックへのいじわる, 情報処理,Vol.11,No.7,pp.426-427(1970 年 7 月).

[3] 中西正和, 大駒誠一: プログラムのページ, 情報処理, Vol.10,No.2(1969 年 2 月).

[4] 中西正和: LISP 入門 システムとプログラミング近代科学社 1977.

[5] System-5:プログラムの調べ方, 数学セミナー, Vol.11,No.4,pp.65-67(1972 年 4 月).

[6] 中西正和: KLISP の拡張機能とその応用, 情報処理, Vol.11,No.10, pp.619-623.