数理解析研究所講究録
第561巻 1985年 136-141

Linkの2変数多項式のmuSIMP/muMATHによる計算

阪大理 村上 順
(Jyun Murakami)

Linkの2変数多項式を求めるプログラムをmuSIMP/muMATHで作ったので紹介します。

特徴 このプログラムの特徴は、一度計算した多項式を憶えていることです。muSIMP/muMATHによるプログラミングはプログラムが数学的な定義にかなり近い形で書けることもあり，非常に生産性が高く，有用です。しかし，どうしても実行速度では、他のコンパイラ言語にはかないません。そこでメモリーや変数の管理に煩わされることなく，色々と憶えていくことが出来るというLISP系言語の特徴を生かして，実行速度を上げる工夫をしました。

プログラムリストの☆の所を見て下さい。☆1～☆3を加えるだけで，記憶するようになります。

☆1では，まずLinkを表わすBraid群の元の語を表わすlist NORMALIZE(REDUCTION(REVERSE(LEX2,LEX1)))からCOMPRESSという関数により，このlistの要素を並べた名前を持つアトムを作り，EX3に代入します。

#2 では、EVAL(EX3) により、EX3に代入されたアトムの値を調べます。そして、それがEX3に代入されたアトムそのものとは違っていたら、それが求める多項式です。等しい時は、そのアトムにはまだ何も代入されてないので、実際に多項式を計算します。

そして#3で、いま求めた多項式を、ASSIGN を用いて、EX3に入っているアトムの値とします。

この3つの steps により、コンピューターはどんどん賢くなります。

使い方　まず、次のことをしておきます。

A> MUSIMP　　muSIMP の起動

・・・　カンマ　セミコロン　行末には改行キーを打って下さい。

? RDS(ARITH, MUS); RDS(ALGEBRA, ARI);

muMATH のうち必要なものを読み込む。

? RDS(プログラムの入ったfile); プログラムの読み込み

? SAVE(ファイル名); 今の状態を保存する。

? SYSTEM(); muSIMP/muMATH を終わる。

>A

これだけのことをしておくと、次からは以下の繰り返しで計算できます。

>A MUSIMP ファイル名　により以前に記憶した多項式を計算機に入れます。そして

クォーテーション・マーク
↓
?LP(n,'list);　により多項式を求めます。最後は

?SAVE(ファイル名);　により現在の状態を保存してから

?STSTEM();　で終わります。

>A

ここで，LPの引数に出てくるnは，Linkを表わすBraidLの糸の数から1をひいた整数です。また，listは，Lを生成元で表わしたものです。

$L=\sigma_{i_1}^{\varepsilon_1}\cdot\sigma_{i_2}^{\varepsilon_2}\cdots\sigma_{i_r}^{\varepsilon_r}$ $(\varepsilon_k=\pm1,\ 1\le i_k\le n)$ のとき $(\varepsilon_1 i_1,\varepsilon_2 i_2,\cdots,\varepsilon_r i_r)$ とします。例えば，  は糸の数が2本で，σ_1^3 とかけるBraidで表現できるので，LP(1,'(1,1,1));により多項式が求まるし，  は糸が3本で，$(\sigma_1\sigma_2^{-1})^2$ と表わされるので，LP(2,'(1,-2,1,-2));で求まります。

皆さんも是非真似してみて下さい。

(参考)

Linkの2変数多項式について　別項の山田氏の記事

muSIMP/muMATHについて　muSIMP/muMATHのマニュアル

```
%  Set EXPAND mode  %
TRGEXPD: -2$
PWREXPD: 6$
NUMNUM: DENDEN: DENNUM: BASEXP: EXPBAS: 30$
NUMDEN: 0$

FUNCTION NORMALIZE (LEX1, %Local% LEX2, LEX3),  %  Get the smallest one of  %
   LEX3: LEX2: LEX1,                            %  the cyclic permutations  %
   LOOP                                         %  of LEX1.                 %
      BLOCK
         WHEN  ORDERED (LEX3, LEX2),  LEX2: LEX3  EXIT,
      ENDBLOCK,
      LEX3: APPEND(REST(LEX3), LIST(FIRST(LEX3))),
      WHEN  LEX1 = LEX3,  EXIT,
   ENDLOOP,
   LEX2,
ENDFUN$

FUNCTION CONTAIN (N, %Local% EX1, LEX4 %Global LEX1, LEX2, LEX3%),
   LOOP                                  % Search the first N in LEX3.      %
      WHEN  ATOM (LEX3), 0  EXIT,        %                  N is not found.%
      WHEN  ABS (EX1: POP (LEX3)) = N,   %                  N is found.    %
           LOOP   % Search the nearest N-1 after the first N before second N.%
              WHEN  ATOM (LEX3),  PUSH (EX1, LEX2),  1  EXIT, % 2nd N isn't  %
              WHEN  ABS (FIRST (LEX3)) = N,     % 2-nd N is found before N-1.%
                    PUSH (EX1, LEX2),  LEX1: APPEND (LEX3, LEX1),  2  EXIT,
              WHEN  ABS (FIRST (LEX3)) = N - 1, %              N-1 is found.%
                    PUSH (EX1, LEX2),
                    LOOP                                %      Search 2-nd N.%
                       WHEN  ABS (EX1: POP (LEX3)) = N,  %    2-nd N is found.%
                             LEX1: APPEND (LEX3, LEX1),  LEX3: FALSE,
                             LOOP
                                WHEN  ATOM (LEX4),  PUSH (EX1, LEX1),  2  EXIT,
                                WHEN  ABS (FIRST (LEX4)) = N - 1,
                                      LEX3: REVERSE(LEX4),  PUSH(EX1,LEX1),
                                      LENGTH(LEX3)+2  EXIT,
                                PUSH (POP (LEX4), LEX1)
                             ENDLOOP  EXIT,
                       PUSH (EX1, LEX4),
                       WHEN  ATOM (LEX3),  LEX1: REVERSE (LEX4, LEX1),  1  EXIT
                    ENDLOOP EXIT,
              PUSH (POP (LEX3), LEX2)
           ENDLOOP EXIT,
      PUSH (EX1, LEX2)
   ENDLOOP
ENDFUN$

FUNCTION LP (N, LEX1),
   WHEN  CHECK (N, LEX1),  JONES0 (N, NORMALIZE(REDUCTION(LEX1)))  EXIT,
   PRINTLINE ('"Invalid data")
ENDFUN$
```

```
)NCTION JONESO (N, LEX3, %Local% LEX1, LEX2, EX1, EX2, EX3, ANS),
 M: N,
 LOOP
    WHEN  ANS,    EXIT,
    WHEN  ZERO (N),   1  EXIT,
    WHEN  ZERO (EX1: CONTAIN (M)),
          JONESO (N - 1, REVERSE (LEX2, LEX1))*(-x - y)  EXIT,
    BLOCK
       WHEN  EX1 = 1,
             N: N - 1,  M: M - 1,  LEX3: REVERSE (REST (LEX2), LEX1),
             LEX2: LEX1: FALSE  EXIT,
       WHEN  EX1 = 2,
             WHEN  FIRST (LEX1) = FIRST (LEX2),
              ☆1 EX3: COMPRESS (NORMALIZE(REDUCTION(REVERSE (LEX2, LEX1)))),
              ☆2 WHEN  NOT EX3 = (ANS: EVAL(EX3)),  EXIT,
                 BLOCK
                    WHEN  FIRST (LEX1) > 0,
                        ANS:
                           JONESO(N, REVERSE(REST(LEX2), REST(LEX1)))*(-y/x)
                         + JONESO(N, REVERSE(REST(LEX2), LEX1))*(-1/x)  EXIT,
                       ANS: JONESO(N, REVERSE(REST(LEX2), REST(LEX1)))*(-x/y)
                          + JONESO(N, REVERSE(REST(LEX2), LEX1))*(-1/y)
                 ENDBLOCK,
              ☆3 ASSIGN (EX3, ANS)  EXIT,
             LEX3: REVERSE (REST(LEX2), REST(LEX1)),  LEX2: LEX1: FALSE  EXIT,
       WHEN  EX1 = 3,
             PUSH (FIRST (LEX3), LEX1),
             WHEN  FIRST(LEX2) * FIRST(PUSH(POP(LEX1),LEX2)) > 0,
                   EX1: SIGN(POP(LEX1))*(M - 1),  PUSH (POP (LEX2), LEX1),
                   PUSH (POP (LEX2), LEX1),  PUSH (EX1, LEX1),  M: N,
                   LEX3: REVERSE (LEX2, LEX1),  LEX2: LEX1: FALSE  EXIT,
             WHEN  FIRST(LEX1) * FIRST(PUSH(POP(LEX2),LEX1)) > 0,
                   EX1: SIGN(POP(LEX2))*(M - 1),  PUSH (POP (LEX1), LEX2),
                   PUSH (POP (LEX1), LEX2),  PUSH (EX1, LEX1),  M: N,
                   LEX3: REVERSE (LEX2, LEX1),  LEX2: LEX1: FALSE  EXIT,
       ☆1    EX3: COMPRESS (NORMALIZE(REDUCTION(REVERSE(LEX2, LEX1)))),
       ☆2    WHEN  NOT EX3=(ANS: EVAL (EX3)),  EXIT,
             PUSH (-POP (LEX1), LEX1),
             BLOCK
                WHEN  FIRST (LEX2) < 0,
                      ANS: JONESO (N, REVERSE (LEX2, REST(LEX1)))*(-1/x)
                         + JONESO (N, REVERSE (LEX2, LEX1))*(-y/x)  EXIT,
                ANS: JONESO (N, REVERSE (LEX2, REST(LEX1)))*(-1/y)
                   + JONESO (N, REVERSE (LEX2, LEX1))*(-x/y)
             ENDBLOCK,
       ☆3    ASSIGN (EX3, ANS)  EXIT,
      M: M - 1
   ENDBLOCK
 ENDLOOP
)FUN$


ICTION CHECK (N, LEX1),                          % If LEX1 contains an element %
 LOOP                                            % whose absolute value is     %
    WHEN  ATOM (LEX1) EXIT,                      % greater than N, then FALSE,  %
    WHEN  ABS (POP (LEX1)) > N,  FALSE  EXIT     %                  else TRUE   %
 ENDLOOP
)FUN$
```

```
FUNCTION REDUCTION (LEX1, %Local% LEX2),          % Change the word LEX1 %
   LOOP                                           % to its reduced word  %
      WHEN  ATOM (LEX1),  LEX1: REVERSE (LEX2)  EXIT,  %  -1             %
      LOOP                                        %   u.g.g .v --> u.v   %
         WHEN  NOT FIRST (PUSH (POP (LEX1), LEX2)) = -FIRST (LEX1),  EXIT,
               POP (LEX1),  POP (LEX2),
               WHEN  ATOM (LEX2)  EXIT,
               WHEN  ATOM (LEX1)  EXIT,
      ENDLOOP
   ENDLOOP                                        %        -1            %
   LOOP                                           %   g.w.g  --> w       %
      WHEN  ATOM (LEX1),  EXIT,
      WHEN  ATOM (REST (LEX1))  EXIT,
      LEX2: LEX1,
      LOOP
         WHEN  ATOM (RREST (LEX2)),  EXIT,
         POP (LEX2)
      ENDLOOP,
      WHEN  NOT (FIRST (LEX1) = -FIRST (REST (LEX2)))  EXIT,
      POP (LEX1),  REPLACER (LEX2, FALSE)
   ENDLOOP,
   LEX1
ENDFUN$


FUNCTION SIGN (X),
   WHEN  X > 0,  1  EXIT,
   -1
ENDFUN$

RDS()$
```