CBPF-NF-037/97

Tópicos do Software FORM

Patrícia Macedo da C. Jorge e Patrícia Duarte Peres

Ciência da Computação
Universidade Católica de Petrópolis (UCP)
e
PIBIC – DCP/CBPF

Índice

1 Compilando com o *FORM* **1**

2 Estruturas de Comando **5**

3 Números e Estatísticas **9**

4 Matrizes de Dirac **23**

5 Controle de Otimização **29**
5.1 Potências Contáveis 36
5.2 Goto e Labels 38

6 Matrizes Gamma **41**

7 O Comando Modulus **49**

8 Convivendo com Erros **51**

9 Substituições Polinomiais **55**

Capítulo 1

Compilando com o *FORM*

Na maioria dos sistemas,o *FORM* é chamado digitando-se o nome do arquivo-programa (usualmente form).

Após o nome pode haver alguns parâmetros.O último parâmetro é geralmente o nome do arquivo que desejamos que seja executado pelo *FORM*. Antes do nome desse arquivo pode haver algumas opções. Estas opções podem ser dadas pela digitação do nome inteiro ou apenas uma parte. Usualmente (exceto para llog),o primeiro caracter já é suficiente.As opções são caso insensitivo.São elas:

- CHECK

Esta opção diz ao *FORM* para verificar somente a sintaxe do programa.Isto é útil principalmente para programas longos .Compilando primeiro com esta opção,o usuário pode evitar uma situação indesejável de ter o programa abortado por um erro de sintaxe perto do final do programa.

-D NAME

Esta opção declara NAME para ser uma variável do pré-processador com o valor 1. Desta maneira pode ser usada nas sentenças do pré-processador do tipo

ifdef 'name '

É permitido ter mais do que uma destas opções.

-D NAME=STRING

Esta opção declara NAME para ser uma variável do pré-processador com teor "string ".Isto é equivalente a primeira sentença do programa sendo:

CAPíTULO 1 COMPILANDO COM O *FORM*

#define name "string "

É permitido ter mais do que uma destas variáveis do pré-processador definidas.

-HOLD

Quando esta opção é especificada (-h já é suficiente),o *FORM* esperará por uma reação do teclado após terminar a compilação.Isto pode ser útil quando o *FORM* é utilizado em uma janela.

-LOG

Esta opção deve preceder para o nome de um arquivo que contém o programa para o *FORM*. Normalmente, a resposta também é escrita na tela ou no arquivo.Este último modo, possui a limitação que o usuário está avançando.A opção -LOG no nome do arquivo.Em alguns sistemas, somente uma extensão de arquivo é permitida.Nestes sistemas,o *FORM* retirará a extensão antiga antes de colocar a extensão .LOG

-LLOG

É o mesmo que -LOG,mas somente o último bloco de estatísticas é escrito para cada expressão. Isto não afeta a exibição das estatísticas na tela.

-SETUPFILE NAME

"Name" é o nome do arquivo com os parâmetros do setup.Quando o *FORM* começa a execução, ele verifica primeiro se existe o arquivo "form.set " no diretório corrente.Se este for o caso, o parâmetro -S é ignorado junto com seu nome de arquivo.Se não existir o arquivo "form.set "e se existir o parâmetro - setup,o *FORM* tentará abrir o arquivo dado e pegará os parâmetros de setup deste. Esta ordem em que o *FORM* olha por seu arquivo setup possibilita ao usuário definir uma macro para o *FORM* que tem um arquivo setup neste,enquanto é permitido o uso de um arquivo setup local.

-TEMPDIR PATHNAME

O caminho deve apontar para o diretório no qual o *FORM* deve fazer seus arquivos temporários. Na maioria dos sistemas, é aconselhável utilizar um TEMPDIR .Se os arquivos temporários são feitos em um dos diretórios do usuário, o tamanho dos arquivos podem ser restritos pela cota de arquivos do usuário.Existe outra maneira de passar o nome de um diretório para arquivos temporários e ela é descrita na seção onde falaremos do arquivo setup.

Exemplo: nto

form -l foo

Isto manda o *FORM* executar o programa no arquivo 'foo '. A resposta é escrita tanto na tela quanto no arquivo 'foo.log '.

Em sistemas VMS, o que começa uma opção pode ser substituído pelo caracter '/ '. Adicionalmente, todos os sistemas permitem o uso de '-s = filename 'e ' -t = pathname 'ao invés de '-s filename 'e '-t pathname '.

Capítulo 2

Estruturas de Comando

Os comandos do *FORM* são agrupados em MODULOS .Cada módulo é lido e compilado.Se não houver erros o módulo é executado. Um programa pode possuir quantos módulos o usuário desejar.

Os módulos são terminados com uma linha cujo primeiro caracter relevante é um ponto-final (pode haver espaços em branco antes do ponto final).Este é seguido por uma palavra-chave que indica o tipo de módulo.O caso insensitivo destes caracteres de qualquer palavra-chave no *FORM* não é muito importante.O caso só é importante nos nomes de variáveis e, talvez,em nomes de arquivos.Esta última sensibilidade é uma função se o sistema computacional sobre o qual o *FORM* roda é sensitivo ao caso desses caracteres.A maioria dos mainframes são sensistivos a isto,.enquanto,por exemplo,o MS-DOS e derivados não são. Após a palavra-chave pode seguir algumas especificações (inclusas entre parênteses),como: dois pontos seguido por algum texto ou um ponto e vírgula.O ponto e vírgula somente é necessário quando existem opções ou quando existe texto.Os primeiros vinte caracteres do texto são usados como uma mensagem em todas as estatísticas que são empressas durante a execução do módulo.Qualquer outro texto após a palavra-chave é ignorado e considerado comentário.

Módulos nada mais são do que declarações e/ou instruções. O número de declarações que é permitido em um módulo,depende da sua instalação.Ela é influenciada por muitos parâmetros que o usuário pode modificar,se estas limitações tornarem-se muito restritas.Uma limitação típica é 100 declarações por módulo com 32.767 caracteres por declaração e não mais do que 50.000 bytes na resposta compilada do módulo.

Uma sentença pode ser também uma declaração,um conjunto de sentenças

CAPíTULO 2 ESTRUTURAS DE COMANDO

executáveis.Todas as declarações começam com zero ou mais espaços e branco e uma palavra-chave.As declarações devem terminar com um ponto-e-vírgula e podem extender-se por mais que uma linha. As linhas que possuem o caracter * na coluna1 é considerada comentário. Todos os caracteres após o ponto-e-vírgula são vistos como comentário. Muitas declarações começam com uma palavra-chave que por serem muito usadas,se escrevermos uma parte da palavra-chave,esta já será reconhecida pelo *FORM*. Exemplos digitar L ao invés de LOCAL , I ao invés de INDICES,etc.

Os espaços em branco são relevantes no *FORM*. Os parâmetros,em sentenças declarativas,podem ser separados também por espaços em branco ou por por vírgula.Os espaços em branco que são adjacentes a um operador ou um parênteses são ignorados.Portanto,

```
L\ F(ab)=a*b;
```

é lido como:

```
L,F(a,b)=a*b;
```

Quando símbolos aritméticos estão envolvidos,os espaços em branco são irrelevantes.Isto significa que

```
L\ F(a-b)=a*b;
```

é lido como:

```
L,F(a-b)=a*b;
```

Geralmente isto é o que o usuário deseja. Não é o caso no exemplo seguinte:

```
modulus -5;
```

Que é lido como:

```
modulus-5;
```

e uma mensagem de erro aparecerá. O uso da vírgula é necessário aqui:

```
modulus,-5;
```

As regras construídas para separar espaços em branco relevantes dos irrelevantes estão quase sempre em consentimento com a expectação intuitiva.Uma exceção é encontrada na representação de números ou nomes muito longos. Porque

```
L F=12345678
 901234567890
```

é lido como:

```
L,F=12345678,901234567890;
```

Uma maneira de "escapar "do final da linha é a forma padrão do UNIX:

```
L F=12345678\
  901234567890;
```

Isto dará a interpretação apropriada:

```
L,F=12345678901234567890;
```

Na verdade,não é preciso começar na coluna um da segunda linha. Após a barra invertida,todos os espaços no início da "continuação "da linha são ignorados,portanto

```
L       F=12345678\
        901234567890;
```

CAPíTULO 2 ESTRUTURAS DE COMANDO

produzindo o mesmo resultado.

Potenciações são indicadas pelo caracter ^, mas o caracter ** é aceito. Somente símbolos, produtos internos e subexpressões podem ter potenciações implementadas de maneira natural. Quando outros objetos possuem potenciações,pode haver uma ambigüidade ou o objeto é colocado dentro de uma "função expoente". Esta é uma função sem um nome e com dois argumentos. O segundo argumento é o expoente. Para o restante,a sua presença é tolerada e pode ser impressa. Funções com potenciações são impressas como uma seqüência de ocorrências individuais da função. O mesmo acontece para componentes vetoriais.

Uma parte dos comandos "compiladores "são os comandos do pré-processador. O pré-processador pode operar sobre o código antes deste ir para o compilador. É o pré-processador que tenta interpretar os espaços em branco e substitui a seqüência ** por ^. Adicionalmente,há muitas instruções pré-procedurais,cada uma delas começam com o símbolo #, seguida de uma palavra-chave. As instruções pré-procedurais não são terminadas por um ponto-e-vírgula. Eles ocupam somente uma linha (que pode ser extendida com o uso da barra invertida no final da linha).

Variáveis pré-procedurais podem ser reconhecidas facilmente, porque elas devem estar entre aspas simples,quando são usados.Isto permite ao usuário concatenar o teor das variáveis pré-procedurais para forma palavras longas.Conhecer as possibilidades pré-procedurais é importante quando desejamos colocar o *FORM* no limite de suas capacidades.

Existem muitas convenções a respeito do programa e resposta. Um produto vetorial é escrito como $p_1.p_2$ sendo que p_1 e p_2 são os dois vetores.

Índices contraídos são tratados de acordo com a convenção Schoonschip . Quando um vetor é escrito na posição onde um índice é esperado,isto significa que o índice que teríamos naquele local era o mesmo índice do vetor, e aquele índice deve ser somado, de acordo com a Convenção do Somatório de Einstein. Esta notação é também muito útil em computação manual.

Capítulo 3

Números e Estatísticas

Até agora, temos assumidoque todos os coeficientes e potenciações que usávamos não causariam problemas, Na prática, os computadores são finitos, portanto sempre haverá limitações. Adicionalmente, existem as limitações VOLUNTARIAS que pertencem à linguagem *FORM*. A principal limitação é que o *FORM* faz suas aritméticas também sobre números racionais ou sobre um campo finito cuja aritmética e módulo de um número inteiro positivo que não tem de serum número primo.A falta de um números em ponto flutuante é experimentado sobre alguns como um lance, mas o *FORM* não é um programa de avaliação numérica. É um programa para manipulação de fórmulas e só como o usuário, durante os estágios de manipulação de alguma computação, daria aos números transcedentais um nome como PI, isto deveria ser feito em *FORM*. Para estágios de avaliações numéricas, deveríamos utilizar qualquer das linguagens que estão designadas à computação, como FORTRAN, C ou PASCAL.

Sobre circunstâncias ordinárias, a aritmética racional é puramente para coeficientes. Também os argumentos de função podem ser números racionais. A limitação do tamanho do numerador e do denominador é de 2^{1600} na maioria das implementações do *FORM*. Para a maioria das implementações, isto será mais do que suficiente. Em versões futuras, será possível alterar este tamanho máximo, utlizando o comando SETUP.

As regras aritméticas para os coeficientes podem ser alteradas para aritmética modular com a declaração MODULO.

```
modulus 7;
```

CAPíTULO 3 NÚMEROS E ESTATíSTICAS

```
Symbol a;
Local F = 20*a + 21*a^2 + 22*a^3 + 1/5*a^4;
print;
.end
```

```
F =
   6*a + a^3 + 3*a^4;
```

O comando MODULO, acima, força toda a aritmética para ser obtido o módulo sete . Isto significa que também as frações podem ser convertidas para inteiros. Portanto, $\frac{1}{5}$ é um inteiro , que multiplicado por 5, nos dará 1. Note que todos os números são convertidos para a parte positiva.

Adicionalmente, à aritmética regular, o comando módulo pode também controlar as limitações de potenciações através da Fórmula de Fermat, onde $x^p = x$, quando a aritmética é módulo p.

```
modulus 7;
Symbol a;
Local F = 20*a^-5 + a^12;
print;
.end
```

```
F =
   6*a + a^6;
```

Quando as potenciações não puderem ser reduzidas como esta, devemos especificar um número negativo no comando módulo.

```
modulus,-7;
Symbol a;
Local F = 20*a^-5 + a^12;
print;
.end
```

```
F =
   6*a^-5 + a^12;
```

Note que a vírgula entre MODULO e 7 é necessário, pois um espaço em branco não se converteria à vírgula automaticamente. Agora nós sabemos o que fazer com os valores positivos e negativos da declaração MODULO, existe apenas um valor não especificado: zero. O comando MODULO ZERO traz a aritmética de volta a números racionais. A declaração MODULO tem o mesmo valor que outras declarações. Em uma instrução .GLOBAL torna-lá-a uma declaração global.

O comando MODULO tem mais uma opção. Se o número no comando MODULO é seguido por uma vírgula e um segundo número, todos os coeficientes serão impressos como uma potenciação deste segundo número. Isto torna possível, este segundo número ser GERADOR, o que significa que estas potenciações devem gerar todos os números possíveis que não sejam zero.Estas potenciações ficarão guardadas em uma tabela, portanto a quantidade de memória disponível pode colocar uma limitação ao tamanho do MODULO:

```
Symbol x;
modulus 7:3;
Local F = x + 2*x^2 + 3*x^3
    + 4*x^4 + 5*x^5 + 6*x^6;
print;
.end
```

```
F =
   x + 3^2*x^2 + 3^1*x^3 + 3^4*x^4 + 3^5*x^5 + 3^3*x^6;
```

Existe mais manipulações com potenciações simbólicas. Quando um símbolo é declarado, é possível especificar uma margem de potenciações para ele. Neste caso, somente as potenciações dentro desta margem serão admitidas.

```
Symbols x(-4:4), y(:5), z(-5:);
Local F = (1/x+x)^6 + (1+y)^6*y^2
    + z*(1-1/z)^10;
print;
.end
```

```
F =
```

CAPíTULO 3 NÚMEROS E ESTATíSTICAS

```
10 + 6*x^-4 + 15*x^-2 + 15*x^2 + 6*x^4 + y^2 + 6*y^3 + 15*y^4 + 20*y
 + 210*z^-5 - 252*z^-4 + 210*z^-3 - 120*z^-2 + 45*z^-1 + z;
```

Esta margem é especificada entre parênteses,após o nome do símbolo. A potenciação máxima e mínima será separada por vírgula.Se qualquer uma delas não for especificada,o padrão mínimo ou máximo é substituido.Este é no mínimo +/− 10.000, mas o valor exato depende da implementação. Uma potenciação que é maior do que o valor padrão(ou menor que o valor mínimo padrão) resulta em mensagem de erro,porque os valores padrões são usados quando a margem de potenciação não é especificado.Miutos números em *FORM* são restritos,e,então são chamados "inteiros curtos". As margens exatas de valores que esses inteiros podem tomar é dependente da implementação.Ele é no mínimo −32768 e vai até +32767.Um exemplo destes inteiros é o valor que eles podem especificar em uma declaração de índices fixos.

A dimensão desta declaração tornará estes "inteiros curtos "positivos. O pré-processador aritmético é também restrito a este tipo de inteiro.Esta aritmética é independente da programação do comando módulo.Até agora,só temos visto pequenos programas.Quando nós imprimimos as estatísticas destes,havia sempre uma única mensagem estática para cada expressão.Quando os programas tornam-se maiores essas estatísticas podem se tornar um pouco mais confusas também. No seguinte programa,alguns dos parâmetros do *FORM* são estabelecidos para ilustrar as possíbilidades de um "pequeno" programa.

```
*
*
* Este exemplo deve ser feito modificando o arquivo setup para
*
* SmallSize 300000
* LargeSize 500000
*
*
Vectors p1,p2,p3,p4,p5,p6,p7,p8;
Local F = e_(p1,p2,p3,p4,p5,p6,p7,p8)
          *e_(p1,p2,p3,p4,p5,p6,p7,p8);
contract;
.end
```

```
Time =      0.16 sec    Generated terms =      702
              F         Terms left      =      388
                        Bytes used      =    14462

Time =      0.27 sec    Generated terms =     1404
              F         Terms left      =      992
                        Bytes used      =    36884

Time =      0.38 sec    Generated terms =     2106
              F         Terms left      =     1578
                        Bytes used      =    57428

Time =      0.44 sec    Generated terms =     2587
              F         Terms active    =     1561
                        Bytes used      =    56760

Time =      0.55 sec    Generated terms =     2805
              F         Terms left      =     2171
                        Bytes used      =    77686

Time =      0.71 sec    Generated terms =     3507
              F         Terms left      =     2786
                        Bytes used      =    98836

Time =      0.77 sec    Generated terms =     3965
              F         Terms active    =     2677
                        Bytes used      =    95008

Time =      0.88 sec    Generated terms =     4208
              F         Terms left      =     3297
                        Bytes used      =   116480

Time =      1.04 sec    Generated terms =     4912
              F         Terms left      =     3890
                        Bytes used      =   135988

Time =      1.10 sec    Generated terms =     5245
```

CAPíTULO 3 NÚMEROS E ESTATíSTICAS

```
                F        Terms active    =      3764
                         Bytes used      =    131582

Time =      1.21 sec     Generated terms =      5678
                F        Terms left      =      4262
                         Bytes used      =    148976

Time =      1.32 sec     Generated terms =      6377
                F        Terms left      =      4863
                         Bytes used      =    171230

Time =      1.37 sec     Generated terms =      6823
                F        Terms active    =      4861
                         Bytes used      =    171094

Time =      1.48 sec     Generated terms =      7066
                F        Terms left      =      5480
                         Bytes used      =    193470

Time =      1.65 sec     Generated terms =      7755
                F        Terms left      =      6117
                         Bytes used      =    215418

Time =      1.70 sec     Generated terms =      7912
0                F       Terms active    =      6117
                         Bytes used      =    215386

Time =      1.81 sec     Generated terms =      8444
                F        Terms left      =      6757
                         Bytes used      =    237556

Time =      1.92 sec     Generated terms =      9133
                F        Terms left      =      7384
                         Bytes used      =    259134

Time =      2.09 sec     Generated terms =      9548
                F        Terms active    =      7337
                         Bytes used      =    257174
```

```
Time =      2.25 sec    Generated terms =      9822
               F        Terms left      =      7964
                        Bytes used      =    278660

Time =      2.36 sec    Generated terms =     10524
               F        Terms left      =      8580
                        Bytes used      =    301032

Time =      2.42 sec    Generated terms =     10987
               F        Terms active    =      8543
                        Bytes used      =    299720

Time =      2.64 sec    Generated terms =     11213
               F        Terms left      =      9139
                        Bytes used      =    322070

Time =      2.75 sec    Generated terms =     11917
               F        Terms left      =      9792
                        Bytes used      =    345740

Time =      2.80 sec    Generated terms =     12243
               F        Terms active    =      9726
                        Bytes used      =    343260

Time =      2.97 sec    Generated terms =     12619
               F        Terms left      =     10399
                        Bytes used      =    365744

Time =      3.08 sec    Generated terms =     13323
               F        Terms left      =     11082
                        Bytes used      =    388176

Time =      3.13 sec    Generated terms =     13956
               F        Terms active    =     11059
                        Bytes used      =    387132

Time =      3.30 sec    Generated terms =     14027
```

```
                F       Terms left      =      11734
                        Bytes used      =     409904

Time =      3.46 sec    Generated terms =      14732
                F       Terms left      =      12410
                        Bytes used      =     432572

Time =      3.57 sec    Generated terms =      15236
                F       Terms active    =      12340
                        Bytes used      =     430166

Time =      3.68 sec    Generated terms =      15425
                F       Terms left      =      12994
                        Bytes used      =     453644

Time =      3.79 sec    Generated terms =      16138
                F       Terms left      =      13626
                        Bytes used      =     475686

Time =      3.85 sec    Generated terms =      16321
                F       Terms active    =      13613
                        Bytes used      =     475154

Time =      4.07 sec    Generated terms =      16826
                F       Terms left      =      14272
                        Bytes used      =     497928

Time =      4.23 sec    Generated terms =      17518
                F       Terms left      =      14899
                        Bytes used      =     518734

Time =      4.29 sec    Generated terms =      17876
                F       Terms active    =      14899
                        Bytes used      =     518590

Time =      4.40 sec    Generated terms =      18222
                F       Terms left      =      15573
                        Bytes used      =     540168
```

```
Time =      4.51 sec    Generated terms =     18926
               F        Terms left      =     16248
                        Bytes used      =    561964

Time =      4.62 sec    Generated terms =     19245
               F        Terms active    =     16227
                        Bytes used      =    561110

Time =      4.78 sec    Generated terms =     19630
               F        Terms left      =     16902
                        Bytes used      =    582932

Time =      4.95 sec    Generated terms =     20336
               F        Terms left      =     17568
                        Bytes used      =    604956

Time =      5.00 sec    Generated terms =     20854
               F        Terms active    =     17506
                        Bytes used      =    602484

Time =      5.11 sec    Generated terms =     21047
               F        Terms left      =     18121
                        Bytes used      =    623000

Time =      5.33 sec    Generated terms =     21739
               F        Terms left      =     18793
                        Bytes used      =    645180

Time =      5.39 sec    Generated terms =     21978
               F        Terms active    =     18781
                        Bytes used      =    644894

Time =      5.50 sec    Generated terms =     22439
               F        Terms left      =     19443
                        Bytes used      =    666266

Time =      5.66 sec    Generated terms =     23130
```

CAPíTULO 3 NÚMEROS E ESTATíSTICAS

```
                F       Terms left      =      20043
                        Bytes used      =     686532

Time =      5.72 sec    Generated terms =      23579
                F       Terms active    =      20007
                        Bytes used      =     685284

Time =      5.88 sec    Generated terms =      23834
                F       Terms left      =      20681
                        Bytes used      =     707140

Time =      6.05 sec    Generated terms =      24538
                F       Terms left      =      21356
                        Bytes used      =     728986

Time =      6.10 sec    Generated terms =      24954
                F       Terms active    =      21347
                        Bytes used      =     728290

Time =      6.21 sec    Generated terms =      25242
                F       Terms left      =      22021
                        Bytes used      =     750260

Time =      6.38 sec    Generated terms =      25946
                F       Terms left      =      22626
                        Bytes used      =     770702

Time =      6.43 sec    Generated terms =      26245
                F       Terms active    =      22576
                        Bytes used      =     768694

Time =      6.54 sec    Generated terms =      26647
                F       Terms left      =      23248
                        Bytes used      =     791032

Time =      6.76 sec    Generated terms =      27350
                F       Terms left      =      23921
                        Bytes used      =     812474
```

```
Time =      6.82 sec    Generated terms =     27587
              F         Terms active    =     23921
                        Bytes used      =    812514

Time =      6.98 sec    Generated terms =     28054
              F         Terms left      =     24600
                        Bytes used      =    834190

Time =      7.09 sec    Generated terms =     28744
              F         Terms left      =     25211
                        Bytes used      =    855320

Time =      7.15 sec    Generated terms =     39123
              F         Terms active    =     25155
                        Bytes used      =    853510

Time =      7.31 sec    Generated terms =     29446
              F         Terms left      =     25839
                        Bytes used      =    876234

Time =      7.52 sec    Generated terms =     30152
              F         Terms left      =     26511
                        Bytes used      =    897880

Time =      7.58 sec    Generated terms =     30546
              F         Terms active    =     26486
                        Bytes used      =    896638

Time =      7.69 sec    Generated terms =     30851
              F         Terms left      =     27117
                        Bytes used      =    918380

Time =      7.80 sec    Generated terms =     31557
              F         Terms left      =     27794
                        Bytes used      =    941166

Time =      7.91 sec    Generated terms =     31976
```

CAPíTULO 3 NÚMEROS E ESTATíSTICAS

```
                F       Terms active     =      27794
                        Bytes used       =     940990

Time =       8.07 sec   Generated terms  =      32258
                F       Terms left       =      28467
                        Bytes used       =     962882

Time =       8.24 sec   Generated terms  =      32961
                F       Terms left       =      29150
                        Bytes used       =     985038

Time =       8.29 sec   Generated terms  =      33213
                F       Terms active     =      29102
                        Bytes used       =     983066

Time =       8.40 sec   Generated terms  =      33665
                F       Terms left       =      29788
                        Bytes used       =    1005622

Time =       8.62 sec   Generated terms  =      34359
                F       Terms left       =      30428
                        Bytes used       =    1027844

Time =       8.68 sec   Generated terms  =      34987
                F       Terms active     =      30360
                        Bytes used       =    1024938

Time =       8.79 sec   Generated terms  =      35062
                F       Terms left       =      31028
                        Bytes used       =    1046616

Time =       8.90 sec   Generated terms  =      35762
                F       Terms left       =      31671
                        Bytes used       =    1067888

Time =       9.01 sec   Generated terms  =      36278
                F       Terms active     =      31620
                        Bytes used       =    1066052
```

```
Time =      9.12 sec    Generated terms =     36469
               F        Terms left      =     32297
                        Bytes used      =   1088726

Time =      9.34 sec    Generated terms =     37168
               F        Terms left      =     32962
                        Bytes used      =   1110500

Time =      9.39 sec    Generated terms =     37518
               F        Terms active    =     32908
                        Bytes used      =   1108610

Time =      9.50 sec    Generated terms =     37870
               F        Terms left      =     33587
                        Bytes used      =   1130690

Time =      9.67 sec    Generated terms =     38576
               F        Terms left      =     34268
                        Bytes used      =   1152738

Time =      9.72 sec    Generated terms =     39123
               F        Terms active    =     34210
                        Bytes used      =   1150588

Time =      9.94 sec    Generated terms =     39282
               F        Terms left      =     34891
                        Bytes used      =   1172868

Time =     10.05 sec    Generated terms =     39973
               F        Terms left      =     35548
                        Bytes used      =   1194706

Time =     10.11 sec    Generated terms =     40122
               F        Terms active    =     35463
                        Bytes used      =   1191454

Time =     10.22 sec    Generated terms =     40320
```

CAPíTULO 3 NÚMEROS E ESTATíSTICAS

```
                F       Terms left      =      35784
                        Bytes used      =    1202556

Time =      10.27 sec   Generated terms =      40320
                F       Terms active    =      35738
                        Bytes used      =    1211896

Time =      14.56 sec   Generated terms =      40320
                F       Terms in output =      18155
                        Bytes used      =     612384
```

Neste programa, nós calculamos um determinante de Gram 8×8 de .Na verdade calculamos somente o número de termos na resposta, porque nós jogamos fora a resposta, após o programa terminado. A avaliação na força bruta de um determinante 8×8, gera $8\,!$ termos. Porém, após 5289 termos, o *FORM* imprime, espontaneamente,as estatísticas e fará isso toda vez que se buffer estiver cheio. Foi o que aconteceu com o programa.

É claro que seria muito mais simples suprimir todas essas estatísticas e imprimir somente as estatísticas finais. Muitos usuários preferem impressão regular à estatísticas intermediárias. Isto nos permite monitorar o que esta acontecendo e ver se as coisas estão caminhando para uma direção indesejada.

Capítulo 4

Matrizes de Dirac

O *FORM* tem as Matrizes Gamma de Dirac como funções construídas com o nome g_. Ele conhece algumas das propriedades destas matrizes e possui alguns comandos para tirar o traço destas matrizes. Adicionalmente,existem algumas outras denominações, como gi_, g5_, g6_ e g7_ para fazer operações mais compactamente.

```
Indices m1,m2,m3,m4;
Local F = g_(1,m1)*g_(1,m2)*g_(1,m3)*g_(1,m4)
        + g_(1,m1)*g_(1,m2)*g_(1,m3);
print;
.sort
```

```
F =
   g_(1,m1,m2,m3,m4) + g_(1,m1,m2,m3);
```

```
trace4,1;
print;
.end
```

```
F =
   4*d_(m1,m2)*d_(m3,m4) - 4*d_(m1,m3)*d_(m2,m4)
    + 4*d_(m1,m4)*d_(m2,m3);
```

CAPíTULO 4 MATRIZES DE DIRAC

Vemos as matrizes gamma na expressão f como o primeiro argumento que deve ser um índice. No caso acima este é 1. Este índice indica uma "linha de spin ". Matrizes de diferentes linhas de spin comutam entre si e não há nada a fazer com ambas quando o traço é tirado. Quando a resposta é impressa vemos que todas as matrizes estão juntas e na mesma função G- .Isto é feito, quando nenhuma outra função não-comutativa esta entre as matrizes. Por outro lado, pode haver mais do que aquelas linhas de matrizes .O comando "trace4" avalia o traço de todas as matrizes gamma com linha de spin 1, analisando cada termo este são pegos dentro de uma linha que não será considerada, se houver função não-comutativas nela. O usuário deve atentar que qualquer conflito possível será resolvido antes de ser concluído o comando TRACE. A sentença TRACE4, força o traço a ser reinterado em quatro dimensões. Isto usa truques que são particulares à quatro dimensões,permitindo uma avaliação rápida do traço. No exemplo acima, isto significa que a linha com 3 matrizes gamma removida,pois o seu traço é trivialmente zero. O traço de 4 matrizes gamma dá-se em sua forma canônica existe uma variação de comando TRACE: TRACEN, que fornece o traço em uma dimensão não especificada. No exemplo acima, isto não faria muita diferença,observamos o seguinte exemplo:

```
Symbol n;
Index m1;
Vectors p1,p2;
Local F = g_(1,m1)*g_(1,p1)*g_(1,m1)*g_(1,p2);
trace4,1;
print;
.sort

F =
    - 8*p1.p2;

drop F;
Index m4 = n;
Local G = g_(1,m4)*g_(1,p1)*g_(1,m4)*g_(1,p2);
tracen,1;
print;
```

```
.end

G =
   8*p1.p2 - 4*p1.p2*n;
```

No que,substituindo n por 4,torna G igual a F. Em geral, os algorítmos de traço quadri-dimensonais são mais compactos. Quando há índices contraidos, é possível aplicar as propriedades de CHISHOLM. É também possível a utilização de fórmulas de redução na ÁLGEBRA DE DIRAC, porque a estrutura exata da álgebra é conhecida em N-dimensões, as coisas são necessariamente mais vagas, portanto muitos destes algoritmos não são permitidos.

O uso da matriz axial y5 é permitida. Seu símbolo é g5_ e ela possui somente o índice da linha de spin se ela for inserida diretamente em uma linha de matrizes $y6 = 1 + y5$ e $y7 = 1 - y5$. Estas podem ser usadas como $g6\_$ e $g7\_$ ou 6_ e 7_ , dentro de uma linha de matrizes gamma. A matriz unitária é $gi\_$ com somente um índice de linha de spin.

Exemplo: Decaimento de um MUON

```
Vectors pmu,pmuneutrino,pe,peneutrino;
Indices m1,m2;
Symbols emassa,mumassa;
write statistics;
Local M =
      g_(1,pmuneutrino)*g_(1,m1)*g7_(1)
      *(g_(1,pmu)+mumassa)*g_(1,m2)*g7_(1)
      *(g_(2,pe)+emassa)*g_(2,m1,7_)
      *g_(2,peneutrino)*g_(2,m2,7_);
trace4,1;
trace4,2;
print;
.end

M =
   256*pmu.peneutrino*pmuneutrino.pe;
```

O programa acima calcula a matriz de elementos para o decaimento de um MUON dentro de um elétron, um MUON neutrino e um é um neutrino

anti-elétron. Existem duas linhas de spin, e nós temos utilizado a interação de quatro pontos de Fermi. Na segunda linha de spin, temos que colocar a $y7$ junto com a linha de matrizes gamma do vértice em uma linha de matrizes gamma. Porque existem duas linhas de matrizes gamma, temos que emitir dois comandos TRACE. A resposta final é um resultado trivial, podendo este ser encontrado em muitos livros.

Em alguns casos, gostaríamos de ter matrizes gamma que não possuíssem uma linha de spin indexada, mas dois índices que indiquem uma função. Para converter uma linha destas matrizes em uma linha de matrizes gamma, e , se fazendo isto, desejarmos retirar o seu traço, esta tarefa não será muito difícil.

```
Vectors p1,p2,p3,p4;
Indices m1,m2,m3,m4;
CFunction g;
Local F = g(m1,m2,p1)*g(m4,m1,p4)
         *g(m2,m3,p2)*g(m3,m4,p3);
repeat;

    id g(m1,m2?,??)*g(m2?,m3?,???) = g(m1,m3,..,...);

endrepeat;
print;
.sort

F =
   g(m1,m1,p1,p2,p3,p4);


id g(m1,m1,??) = g_(1,..);
print;
.sort

F =
   g_(1,p1,p2,p3,p4);


trace4,1;
```

```
    print;
    .end

  F =
     4*p1.p2*p3.p4 - 4*p1.p3*p2.p4 + 4*p1.p4*p2.p3;


trace4,1;
   print;
   .end

  F =
     4*p1.p2*p3.p4 - 4*p1.p3*p2.p4 + 4*p1.p4*p2.p3;
```

O programa acima mostra todas as técnicas. Primeiro, a função g é usada como uma matriz gamma com dois índices. As matrizes estão juntas em uma linha que é claramente o traço do produto de matrizes. Agora, note que apesar do campo de argumentos wildcard não poderem ser usados na função $g\_$, quando isso acontece no lado esquerdo, podemos usá-los quando estes ocorrem do lado direito e, consequentemente, utilizamos:

```
id g(m1.m1,??) = g_(1,..);
```

Após este capítulo, tornou-se simples tirar o traço. Devemos também notar que não colocamos na wildcard o índice $m1$. Isto permite-nos selecionar as matrizes de somente uma linha de spin. Isto evita, também, a desorganização das matrizes de duas linhas de spin.

Para mais informações sobre as matrizes gamma e quais os algoritmos utilizados para retirar o traço destas, consultar o capítulo MATRIZES GAMMA.

Capítulo 5

Controle de Otimização

O *FORM* é equipado com um número de decisão utilizando instruções. Alguns operam em nível pré-processador. Um deles é a instrução do pré-processador IF. Esta instrução possui um papel importante na preparação de respostar para o compilador. Às vezes, é preciso tomar algumas decisões durante o tempo de execução. Esta decisão pode ser baseada no conteúdo particular de um termo. Os mecanismos para isso são as declarações de loop IF e WHILE. Com a declaração IF é possível executar um número de declarações seletivamente dependente da forma particular de um termo. Esta declaração trabalha um pouco como as linguagens C e FORTRAN: existe uma declaração

- IF
- ELSE
- ELSEIF
- ENDIF

As declarações IF, ELSEIF e WHILE possuem cada qual uma condição entre paress de parentêses. Todas as declarações acima devem ser terminadas por um ; e suas traduções são feitas pelo compilador.

A sintaxe de uma declaração WHILE é:

```
while (expressao);
endwhile;
```

CAPíTULO 5 CONTROLE DE OTIMIZAÇÃO

A sintaxe de uma declaração IF é:

```
if (expressao);
[elseif (expressao);]
[else;]
endif;
```

As declarações ELSE e ELSEIF são opcionais. Pode haver mais do que uma declaração ELSEIF. A expressão em um WHILE, IF ou ELSEIF podem ser constituídas de subexpressões, cada uma sendo do tipo

```
( subexpressao1 operador subexpressao2 )
```

Os blocos básicos são os que colocamos as declarações IF e WHILE em *FORM*, à parte das declarações correspondentes em C e FORTRAN. Estes objetos podem ser:

- Um número ou uma fração

Qualquer número ou fração que permaneça com as limitações que governam quantidades.

coeff[icient]

Esta palavra indica coeficiente do termo corrente.

- MATCH (PATTERN)

O valor do objeto é o número de vezes que pode ser retirado do termo corrente. O exemplo é dado no mesmo formato do lado esquerdo de uma declaração ID. Isto significa que ela pode conter palavras chave como ONCE, ONLY, MANY, SELECT, etc. O exemplo também pode conter wildcards. A interpretação desses exemplos e palavras chave são idênticas à interpretação em uma declaração ID, não havendo, é claro, nenhum sinal de igual em nenhum lado direito.

- COUNT (CONTAR VALORES)

Isto fornece um valor de potências contáveis baseados nos valores individuais dos objetos especificados no campo de argumentos.

Os objetos podem ser combinados com um número de operadores lógicos:

- =

Os valores numéricos de dois objetos são comparados. A resposta é verdadeira, se eles forem iguais.

- !=

O valor numérico de dois objetos são comparados. A resposta é verdadeira se eles não forem iguais.

- >

Os valores numéricos de dois objetos são comparados. A resposta é verdadeira se o primeiro objeto for maior do que o segundo.

- >=

Os valores numéricos de dois objetos são comparados. A resposta é verdadeira se o primeiro objeto for maior ou igual ao segundo.

- <

Os valores numéricos de dois objetos são comparados. A resposta é verdadeira se o primeiro objeto for menor do que o segundo.

- <=

Os valores numéricos de dois objetos são comparados. A resposta é verdadeira se o primeiro objeto for menor ou igual ao segundo.

- &&

O valor lógico de dois objetos são comparados. A resposta é verdadeira se ambos os objetos forem verdadeiros.

- ||

CAPíTULO 5 CONTROLE DE OTIMIZAÇÃO

O valor lógico de dois objetos são comparados. A resposta é verdadeira se um dos objetos for verdadeiro.

A comparação de dois objetos é feita com a instrução <OBJETO 1> <OPERADOR LÓGICO> <OBJETO 2>.Um objeto possui ambos os valores numéricos e lógicos. O valor numérico é, por exemplo, retornado por MATCH, COUNT ou COEFFICIENT. Se um valor numérico tem que ser convertido à um valor lógico, é convertido para FALSO, quando o valor numérico é zero. É convertido para VERDADEIRO, quando o valor numérico não é igual a zero. Quando um valor lógico tem que ser convertido à um valor numérico, FALSO é traduzido em zero e VERDADEIRO é traduzido em um.

```
if (coefficient == 10/3);
   id x = (y+z);
endif;
```

É possível concatenar um número de objetos e operações lógicas sem parênteses.

```
if (match(only,a^2) && match(f?(a?))
                    && match(e_(mu,nu)) );
```

A regra aqui é que cada objeto é avaliado e, então, combinados com o resultado total de tudo à esquerda dele. Isto pode causar alguns problemas:

```
if (match(a^2*b) == 3\ && match(a*b^2) == 4);
```

A declaração, provavelmente, será interpretada de uma maneira diferente daquela que o usuário deseja. Primeiro, o valor retornado de MATCH (a^2*b) é comparado com 3. Vamos assumir que o resultado seja VERDADEIRO. Então o valor retornado de MATCH (a^2*b) é convertido à lógico e o lógico e operação é pego com o resultado prévio. Isto fornece em nosso exemplo o valor resultante VERDADEIRO. Finalmente este valor é VERDADEIRO é convertido em um e comparado com 4. O resultado final é sempre FALSO. A condição pode ser escrita com o uso de parênteses:

```
if ((match(a^2*b) == 3)
&& (match(a*b^2) == 4) );
```

Deve ser notado que se o primeiro objeto de um operador lógico E fornece o valor FALSO, não existe necessidade de se avaliar o segundo objeto. Isto significa que no exemplo acima o segundo MATCH nunca será tentado se o primeiro MATCH falhar ao retornar o valor 3. Uma observação lógica pode ser feita sobre a operação lógica IF. Se o primeiro objeto retornar o valor VERDADEIRO, o segundo objeto não será avaliado.

É possível utilizar uma declaração IF sem um operador lógico. Isto é feito, quando a condição é simples:

Exemplo *a*:

```
s x, a, b, c;
f f;

if ( match(f?(x?simbolo_)) );
   id a^2*b = 2*c;
endif;
```

No exemplo acima, a^2b é substituído por $2 * c$ em todos os termos que contém no mínimo uma função com um único argumento que é um símbolo. É claro que um resultado parecido pode ser obtido com as declarações.

Exemplo *b*:

```
s x, a, b, c;
f f;

repeat;
    id f?(x?simbolo_)a^2*b = 2*c*f(x);
endrepeat;
```

Com expressões maiores, poderíamos notar logo que o método do exemplo a é muito mais rápido e muito mais fácil de ser lido. Para potências maiores de a e b, pode acontecer da solução com a declaração REPEAT não poder ser usada, a menos que o tamanho do espaço de trabalho seja extendido. Há uma teceira solução que evita os problemas do exemplo b, mas o exemplo a ainda é mais simples, rápido e fácil de se entender.

Exemplo *c*:

CAPíTULO 5 CONTROLE DE OTIMIZAÇÃO

```
s x, a, b, c, dummy, dumpow;
f f;

id a*b^2 = dummy;
id f?(x?)*dummy^dumpow? = f(x)*(2*c)^dumpow;
id dummy = a*b^2;
```

É muito mais difícil encontrar maneiras sobre a declaração IF, quando a declaração ELSE toma sua parte na ação:

```
s x, a, b, c;
f f:

if ( (match(f?(x?)) );
   id a*b^2 = 2*c;
else;
   id a^2*b = 2*c;
endif;
```

Deixamos para o leitor a tentativa de utilizar o método *b* para remover a declaração IF no último exemplo.

É permitido concatenar declarações IF. Também é permitido colocar indicações dentro da MARGEM de uma declaração IF. Esta margem consiste das declarações entre um IF e seus ENDIF. Não é permitido colocar instruções de fim de módulo dentro de uma declaração IF.

Às vezes, temos múltiplas condições. A solução com somente uma combinação de declarações IF, ELSE, e, ENDIF é, portanto, deselegante. Por esta razão, a declaração ELSEIF é uma combinação especial de declarações IF e ELSE que não precisam de correspondentes ENDIF extras, como mostrado no próximo exemplo, no qual ambas as construções realizam o mesmo trabalho:

```
if ( count(x,1) == 3 );
    id y = 4;
else;
    if ( count(x,1) > 3\ );
        id y = 5;
    else;
```

```
        id y = 2;
    endif;

if ( count(x,1) == 3);
    id y = 4;
elseif ( count(x,1) > 3);
    id y = 5;
else;
    id y = 2;
endif;
```

No primeiro caso, podemos colocar declarações ENDIF, quando as condições tornam-se complicadas.

Para alguns loops controlados, a declaração WHILE pode ser a melhor opção. a sintaxe é simples:

```
while (condicao);
       declaracoes
endwhile;
```

Funcionalmente, podemos reescrver um loop de diferentes maneiras:

```
repeat;
   if (condicao)
       declaracoes
   endif;
endrepeat;
```

ou

```
label 1;
if (condicao);
    declaracoes
    goto 1;
endif;
```

CAPíTULO 5 CONTROLE DE OTIMIZAÇÃO

O último método é claramente inferior.O método com a declaração REPEAT é, algumas vezes, a melhor opção, quando a condição não é tão simples e envolve uma ou mais declarações ELSE/ENSEIF.

Quando uma declaração IF é seguida de somente uma declaração e ENDIF, é possível utilizar uma notação abreviada no qual o ponto e vírgula após a declaração IF é omitido e também o ENDIF pode se deixado de lado. Uma notação abreviada parecida, existe para um loop WHILE que possua uma única declaração dentro do loop:

```
if ( coeff < 10/3 ) discard;
while ( count(x,1) > 1) id x^2 = y+x;
```

Note que é possível criar loops infinitos. Um loop,usualmente, parará de diversas maneiras:

- Se o espaço de trabalho estiver cheio, o *FORM* começa a reclamar que o tamanho do espaço de trabalho é insuficiente. Às vezes, a reclamação é verdadeira, mas na maioria das vezes, indica um loop impróprio.
- O programa "cai" devido à alguma razão misteriosa. Neste caso, há armazenamento antes do espaço de trabalho estar cheio. Existe implementações de *FORM* que não checam este armazenamento, porque isso causará queda de execução ou usará mais memória. De longe, o largo número de terminações anormais acontecem por isso.
- O *FORM* reclama sobre outros buffers se estes não forem grandes o suficiente.

Limitações: O armazenamento de declarações IF é restrito à dez níveis. Portanto, o armazenamento de parênteses que são usados dentro da parte condicional da declaração IF é feito, mas não estão incluídos os parênteses que são usados dentro das operações MATCH ou COUNT.

5.1 Potências Contáveis

É um método para atribuir um valor para um termo. Este valor é, no caso, mais simples, e devido a isto o nome POTÊNCIAS CONTÁVEIS. No *FORM*, algo

5.1 POTÊNCIAS CONTÁVEIS

pode ser conseguido com a função COUNT que pode ser usada nas declarações IF e WHILE. A sintaxe básica desta função é possuir pares de argumentos. O primeiro elemento do par é um objeto e o segundo é seu valor (a ser adicionado). Estes objetos podem ser de diferentes tipos. Os tipos permitidos são:

- **Símbolo**

O objeto é o nome de um símbolo. Quando a contagem é realizada, o valor é multiplicado pela potência deste símbolo e adicionado à contagem.

- **Função**

O objeto é o nome de uma função. Para cada ocorrência desta função, os valores dos argumentos fora da função são adicionados à contagem.

- **Produto Interno**

O objeto é um produto interno. Estes produtos internos são manipulados da mesma maneira que os símbolos.

- **Vetor**

O objeto é o nome de um vetor. Neste caso, existe possíveis ambigüidades. Também existe a necessidade de ter algum controle sobre aquelas ocorrências do vetor. Existem novamente quatro casos:

1. Vetor: Vetores livres com índices.

2. Produto Interno: Vetores dentro de produtos internos.

3. Função: Vetores dentrode tensores e funções especiais, como LEVI-CIVITA e MATRIZES GAMMA.

4. Conjuntos: Vetores que são os argumentos de uma função a qual é um membro de um conjunto especificado. É assumido que a função é linear no vetor.

CAPíTULO 5 CONTROLE DE OTIMIZAÇÃO

No momento, temos controle sobre vários casos onde existe a possibilidade de adicionar algumas opções depois do nome do vetor. Isto é feito digitando-se um + depois do nome do vetor, seguido de um ou mais caracteres. V, para vetores; F, para funções; D, para produtos internos; e ?SETNAME para opções de conjunto. Estas são sempre a última opção e somente um conjunto pode ser especificado. Quando não existe + seguido de um nome de um vetor, o resultado é idêntico ao nome do vetor +VFD.

No princípio, pode haver alguma interferência entre a contagem de vetores e produtos internos. A regra é que se um objeto ocorre mais de uma vez na lista, ela é contada mais de uma vez, portanto:

```
count(p,1,p.q,-1)
```

fornece para o produto interno P.Q um valor de contagem zero e para o produto interno P.K uma valor de contagem +1(assumindo que P,Q,K são vetores). O produto interno $p.k^5$ forneceria uma contagem de cinco e $p.p$ fornece uma contagem dois. O termo $p(alfa) * p.q$ daria uma contagem de um.

Adicionalmente a função COUNT, existe uma declaração COUNT. Ela é originada da compatibilidade com Schoonship. Porque a versão

```
if (count(x,1,y,2) < 4) discard;
```

é muito mais legível que

```
count 4,x,1,y,2;
```

a última versão eventualmente será removida da sintaxe.

5.2 Goto e Labels

Como qualquer linguagem de computação decente, o *FORM* é equipado com uma declaração não-famosa GOTO . Para esta declaração trabalhar satisfatoriamente, existe a declaração LABEL. Esta declaração, vem da palavra 'label ', um espaço em branco ou uma vírgula e então um número de 0 a

5.2 GOTO E LABELS

20. A sintaxe da declaração GOTO é parecida : a palavra goto seguida de um espaço em branco ou vírgula depois que existir o número da label que é definido no mesmo módulo e então um número um ponto e vírgula .As labels existem somente durante a execução do módulo no qual elas são definidas. Isto significa que o número de uma label pode ser utilizado novamente em outros módulos.

As principais utilidades de labels no *FORM* são para as construções de loops e simplificações de construções difíceis, no qual teríamos um pedaço de código ocorrendo mais de uma vez. Eles podem ser úteis para pular os limites de muitos loops armazenados.

Não é permitido utilizar uma declaração GOTO dentro de uma declaração REPEAT/ENDREPEAT .

Capítulo 6

Matrizes Gamma

Pelo seu uso em Física de Altas Energias, o *FORM* é equipado com uma classe dessas funções construídas. Estas são genericamente denotadas por g_ .As matrizes Gamma satisfazewm as relações:

```
{g_(j1,mu),g_(j1,nu)} = 2 * d_(mu,nu)
[g_(j1,mu),g_(j2,nu)] = 0      j1 nao e igual a j2
```

O primeiro argumento é chamado índice de linha de spin. Quando matrizes gamma possuem a mesma linha de spin, elas pertencem à mesma álgebra de Dirac e comutam com as matrizes de outras álgebras de Dirac. Os índices μ e ν estão sobre o espaço-tempo e, portanto, variam de 1 até 4(ou de 0 a 3 na métrica de Bjorken & Drell). O produto totalmente anti-simétrico e_ (m_1,m_2,...,m_n)g_ (j,m_1) é definido sendo gamma5 ou g5_(j) . A notação 5 encontra suas raízes em 4 espaço-tempo é denotada por g_{i5}(j) . Em quatro dimensões , uma base da álgebra de Dirac pode ser dada por:

```
gi_(j)
g_(j,mu)
[g_(j,mu),g_(j,nu)]/2
g5_(j)*g_(j,mu)
g5_(j)
```

CAPíTULO 6 MATRIZES GAMMA

Em um número diferente de dimensões, esta base é correspondente a seguinte notação por conveniência:

```
g6_(j) = gi(j) + g5_(j)       (de Schoonschip)
g7_(j) = gi_(j) - g5_(j)
g7_(j,mu,nu) = g_(j,mu)*g_(j,nu)     (de Reduce)
g_(j,mu,nu,.....,ro,si) = g_(j,mu,nu,.....,ro)*g_(j,si)
g_(j,5_) = g5_(j)
g_(j,6_) = g6_(j)
g_(j,7_) = g7_(j)
```

A operação comum sobre as matrizes gamma é obter o traço desta .Isto é feito com a declaração: TRACE4, J

Tirar o traço em 4 dimensões de combinação de todas as matrizes com a linha de spin j no tempo corrente. Qualquer objeto não-comutativo que pode estar entre alguma destas matrizes é ignorado. É responsabilidade do usuário emitir a declaração somente depois de todas as funções das matrizes relevantes estarem resolvidas. O 4 refere-se à truques especiais que podem ser aplicados em 4 dimensões. Isto permite, relativamente, expressões compactas.

TRACEN, J

Tira o traço em um número não-especificado de dimensões. O número de dimensões é considerado ser o mesmo. Os traços são avaliados somente pelo uso de propriedades de anticomutação de matrizes. Como o número de dimensões não é especificado, a ocorrência de $g^5 - (j)$é um erro fatal. Em geral, as expressões que são geradas desta maneira estão longe de expressões 4-dimensionais.

É possível alterar o valor do traço da matriz unitária. O seu valor padrão é quatro, mas pelo uso da declaração

```
unittrace value;
```

isto pode ser alterado. O valor pode ser qualquer pequeno número positivo ou um único símbolo com exceção do símbolo i_.

Existem muitas opções para traços quadri-dimensionais. Estas opções encontram sua origem em uma relação que é válida em 4-dimensões, mas não em um número geral de dimensões. Esta relação pode ser encontrada na literatura. É dada por:

$$\gamma_\mu Tr\left[\gamma_\mu S\right] = 2\left(S + S^R\right)$$

no qual S é um conjunto de matrizes gamma com um número opcional de matrizes (γ_5 conta com um mesmo número de matrizes). S^R é a matriz inversa. Esta relação pode ser usada para combinar traços com índices comuns. Na vesão corrente do *FORM*, o uso desta relação é o padrão para o comando TRACE4. Se for necessário faremos esta mudança, devemos adicionar o parâmetro extra 'nocontract':

```
trace4, nocontract,j;
```

O parâmetro 'contract' é o padrão, mas isso pode ser usado para aumentar a legilibilidade do programa. O segundo parâmetro que refere-se à esta relação é o parâmetro 'symmetrize' (ou com o parâmetro 'nonsymmetrize') o primeiro destes índices é retirado e a relação é aplicada à ela. Com o parâmetro 'symmetrize' todas as possibilidades sobre o termo são retiradas. Isto significa, é claro, que na existência de dois índices comuns, a quantidade de trabalho é dobrada. Em alguns traços que envolvem o uso de γ_5, o uso de algoritmos automáticos resultam, às vezes, em uma avalanche de termos com um único tensor de Levi-Civita, enquanto argumentos simétricos podem mostrar que estes termos deverão resultar em zero.

O *FORM*, às vezes, é capaz de eliminar todos ou o mais próximo de todos os tensores de Levi-Civita pelo trabalho dos traços em sua forma simétrica. Normalmente, uma eliminação é complicada. Isto envolve relações que, de longe, desafiam a própria implementação, mesmo porque pessoas tem procurado estes algoritmos de longa data. Conseqüentemente, o uso da simetria desde o começo até o momento, a melhor escolha.

Os traços n dimensionais podem utilizar uma forma especial, quando a declaração dos índices envolvidos o permitirem. Quando um índice tiver sido declarado como n dimensional e a dimensão é seguida por um segundo símbolo como em

CAPíTULO 6 MATRIZES GAMMA

```
symbols n,nn;
index mu = n:nn;
```

e se o índice mu é contraído em um único traço n-dimensional, então a fórmula para este traço pode ser reduzida utilizando-se nn (um termo) ao invés da quantidade (n-4) (dois termos). Isto pode fazer a retirada dos traços n-dimensionais significativamente mais rápida.

ALGORITMOS:

O *FORM* tem sido equipado com muitas construídas para manter o número de termos gerados ao mínimo, durante a avaliação do traço. Estas regras são:

REGRA 1: Matrizes com um número ímpar de matrizes (gamma5 conta por um mesmo número de matrizes) tem um traço que é zero, quando usando TRACE4 ou TRACEN.

REGRA 2: Um conjunto de matrizes gamma é primeiro examinada por matrizes adjacentes que têm os mesmos contraíveis, ou que são contraídos com o mesmo vetor. Se um par é encontrado, as relações

```
g_(1.mu,mu) = gi_(1)*d_(mu,mu)
g_(1,p1,p1) = gi_(i)*p1.p1
```

são aplicadas.

REGRA 3: Há um exame para pares de mesmos índices contraíveis que possuem um número ímpar de outras matrizes entre elas. Isto é feito somente para quatro dimensões (TRACE4) e a dimensão dos índices deve ser quatro. Se encontrados, a identidade de Chisholm é aplicada:

```
g_(1,mu,m1,m2,...mn,mu) = -2*g_(1,mn,...,m2,m1)
```

REGRA 4: Então, (novamente para TRACE4) há uma procura de pares de matrizes com o mesmo índice quadri-dimensional e um mesmo número de matrizes entre elas. Se encontradas, uma das seguintes variações da identidade de Chisholmm é aplicada:

```
g_(1,mu,m1,m2,mu) = 4*gi_(1)*d_(m1,m2)
g_(1,mu,m1,m2,...mj,mn,mu) = -2*g_(1,mn,m1,m2,...,mj)
                             +2*g_(1,mj,...,m2,m1,mn)
```

REGRA 5: Há um exame de pares de matrizes que possuam o mesmo índice ou que são contraídos com o mesmo vetor. Se encontrado, a identidade

```
g_(1,mu,m1,m2,...,mj,mn,mu) = 2*d_(mu,mn)*g(1,mu,m1,m2,...,mj)
                            - 2*d_(mu,mj)*g(1,mu,m1,m2,...,mn)
                            ...
                            -/+2*d_(mu,m2)*g_(1,mu,m1,...,mj,mn)
                            +/-2*d_(mu,m1)*g_(1,mu,m2,...,mj,mn)
                            -/+2*d_(mu,mu)*g_(1,m1,m2,...,mj,mn)
```

é usada para 'anticomutar' estes objetos idênticos até eles se tornarem adjacentes e poderem ser eliminados com a aplicação da REGRA 2. No caso de um traço n-dimensional e quando mu é um índice (ele deveria ser também um vetor na fórmula acima) para qual a definição da dimensão envolvia dois símbolos, existe uma fórmula menor. Neste caso, os últimos três termos podem ser combinados em dois termos:

```
-/+(n-4)*g_(1,m1,m2,...,mj,mn)
-/+4*d_(m1,m2)*g_(1,m3,m4,...,mj,mn)
```

Deve estar claro agora que esta fórmula é somente superior quando há um símibolo único para representar $(n-4)$. Depois de todas as matrizes gamma que são deixadas possuírem um índice diferente ou que foram contraídos com diferentes vetores. Este são tratados utilizando:

REGRA 6: Traços em quatro dimensões para o qual todas as matrizes gamma possuem um índice diferente, ou que são contraídos com um quadri-vetor diferente ou são avaliados usando a fórmula de redução

```
g_(1,mu,nu,ro) = g_(1,5_,si)*e_(mu,nu,ro,si)
               + d_(mu,nu)*g(1,ro)
               - d_(mu,ro)*g(1,nu)
               + d_(nu,ro)*g(1,mu)
```

Para o comando TRACEN, o algoritmo geral é baseado na geração de todos os possíveis pares de índices/vetores que ocorrem nas matrizes gamma em combinação com o próprio sinal deles. Quando a dimensão não é especificada, não há nenhuma expressão menor.

CAPíTULO 6 MATRIZES GAMMA

Observações:

Quando um índice é declarado tendo dimensão n e o comando TRACE4 é usado, as regras especiais quadri-dimensionais 2 e 3 não são aplicadas para este índice. A aplicação da REGRA 2 ou REGRA 4 darão, então, o resultado correto. O resultado, todavia, estará errado de acordo com a REGRA 5, quando existe no mínimo 10 matrizes gamma deixadas depois da aplicação das primeiras quatro regras, como os dois algoritmos da REGRA 5 dá somente uma diferença, quando existe no mínimo 10 matrizes gamma. Para a contagem de matrizes gamma a γ_5 conta por quatro matrizes, com respeito à essa regra. O resultado é imprevisível quando ambos os índices em quatro dimensões e índices em n dimensões ocorrem no mesmo conjunto de matrizes gamma. Portanto, devemos ter muito cuidado quando usamos o traço quadri-dimensional sob a condição que os resultados necessitam para serem corretos em n dimensões. Isto é, às vezes, necessário, quando uma γ_5 é envolvida.

A declaração TRACEN não permitirá a presença de uma γ_5. Em geral, é melhor simular traços n-dimensionais com γ_5 separadamente. O traço eventual, de uma com todas as matrizes com um índice diferente, pode ser gerado com o uso da função distrib_:

```
I m1,m2,m3,m4,m5,m6,m7,m8;
F G,g1,g2;
L F = G(m1,m2,m3,m4,m5,m6,m7,m8);
id G(??) = distrib_(-1,4,g1,g2,..);
id g1(??) = e_(..);
id g2(??) = g_(1,..);
tracen,1;
.end

Time =       0.11 sec    Generated terms =        210
                F        Terms in output =        210
                         Bytes used      =       4034
```

Este resultado simétrico está em contraste com o resultado quadri-dimensional que é muito menor, mas não é muito simétrico.

```
I m1,m2,m3,m4,m5,m6,m7,m8;
L F = g_(1,5_,m1,m2,m3,m4,m5,m6,m7,m8);
```

```
    trace4,1;
    .end

Time =        0.00 sec    Generated terms =          33
                  F       Terms in output =          33
                          Bytes used      =         730
```

Os trabalhos precisos da função distrib_ pode ser observada no capítulo que descreve 'Funções Especiais'.

Capítulo 7

O Comando Modulus

Às vezes, é necessário usar coeficientes que são inteiros. Isto pode ser feito convenientemente em *FORM*. A declaração:

```
Modulus number;
```

especifica que toda a aritmética deve ser módulo do número 'number'. Deste comando somente o m é obrigatório, os outros caracteres da palavra 'modulus' são opcionais.

Quando o número é menor do que a potência máxima que é permitida para símbolos, as potências de símbolos e produtos internos são reduzidas com a relação

```
x^number = x;
```

Todas as potências, então são positivas ou zero. A situação com argumentos da função é algo mais complicado. Ele retira o módulo do "número", mesmo achando que pode estar incorreta, se um dos argumentos representam uma potência. É responsabilidade do usuário evitar estes problemas.

A aritmética do módulo é extendida à frações. Existem., também, reduções para inteiros, porque o inverso de um número pode ser definido propriamente, ao menos que 'number' não seja um número primo. Se este é o caso, a divisão por zero pode ter resultado.

Quando o número que é especificado na declaração 'módulo' é maior do que a potência máxima que é permitida, não haverá redução de potências. O

CAPíTULO 7 O COMANDO MODULUS

usuário deve trabalhar sobre esta restrição com o uso de síimbolos especiais. Este é o preço a ser pago pela velocidade do *FORM*.

O argumento é uma declaração 'módulo' que deve ser positivo ou zero. Se o argumento é 0 ou 1, a declaração estaria sem sentido. É, então, interpretada como uma indicação que qualquer existência de aritmética modular deve ser desligada e aritméricas fracionais ordinárias deve ser reintegradas.

Às vezes, a redução de potências na aritmética modular pode ser um estorvo. Neste caso, esta redução pode ser desligada pela especificação de um valor negativo em uma declaração modulus. Portanto

```
Modulus,-17;
```

colocaremos todas as aritméticas de coeficientes para módulo 17, mas os expoentes não serão afetados. Note que a vírgula é necessária aqui.

Quando o valor é uma declaração 'modulus' é um número primo, existe uma outra forma que é escrita em minutos. É possível encontrar um número x, tanto que quando a aritmética é módulo p, todos os números maiores do que zero e menores do que p pode ser escrito como x^n para algum n. Um número x é chamado de gerador com a declaração

```
Modulus p:x;
```

O *FORM*, então, construirá uma tabela em que cada número menor do que p é expresso como uma potência do gerador. Isto falhará, quando o gerador diz que não pode ser um gerador real ou quando não existe memória suficiente para uma tabela. Portanto, esta opção é somente útil quando p não é muito grande.

Capítulo 8

Convivendo com Erros

Mesmo sendo um usuário dos mais cuidadosos, ocasionalmente há algumas mensagens de erros misteriosas que aparecem na tela do programa *FORM*. Temos , por exemplo, o seguinte programa:

```
Symbols x,n;
CFunctions g1,g2
Functions f1,f2,f,dx;
Local F = f1(0,x)*f2(0,x);
multiply dx;
repeat;
    id,dx*f?(n?,x) = f(n+1,x)+f(n,x)*dx;
endrepeat;
id dx = 0;
print;
.end

WorkSpace Overflow          4000 bytes is not enough
```

É também possível que este programa não produza uma mensagem de erro, mas, sim que um erro de memória ocorra primeiro, causando uma interrupção, nada elegante, do programa (em muitas implementações não há verificação para erros de memória). O que aconteceu aqui? O programa se parece com muitos outros que fizemos nesta apostila.

CAPíTULO 8 CONVIVENDO COM ERROS

Uma das maneiras na qual o *FORM* pode ajudar-nos é imprimindo como ele interpretou as listas de nomes. Podemos forçar isso com a declaração "write names". Esta declaração liga o comando que diz ao *FORM* para permitir-nos visualizar a lista de nomes cada vez que o *FORM* encontrar uma instrução do tipo fim de módulo. Este comando pode ser desligado com a declaração "Nwrite names".

```
    Symbols x,n;
    CFunctions g1,g2
    Functions f1,f2,f,dx;
    Local F = f1(0,x)*f2(0,x);
    multiply dx;
    repeat;
        id,dx*f?(n?,x) = f(n+1,x)+f(n,x)*dx;
    endrepeat;
    id dx = 0;
    write names;
    print;
    .end

 Symbols
   i_#i x n
 Functions
   g_#i(Tensor) sum_ sump_ reverse_ distrib_
 Commuting Functions
   e_#i(Tensor) fac_ theta_ delta_ replace_ integer_ order_ g1 g2 Function
   f1 f2 f dx
 Expressions
   F(local)
 Expressions to be printed
   F
WorkSpace Overflow        4000 bytes is not enough
```

Agora temos: nossas funções têm sido digitadas na lista de nomes como funções comutativas, mesmo aquelas que nós declaramos como funções não-comutativas. Adicionalmente, há a misteriosa função "functions". Vamos analisá-la. O leitor pode notar que está faltando um ponto-e-vírgula (;) na

declaração de funções comutativas. Portanto, o *FORM* lê a próxima linha como uma continuação da linha anterior.

O que acontece mais freqüentemente é que o usuário esquece o ponto-e-vírgula e depois o compilador manda uma mensagem sobre a sintaxe incorreta de uma declaração que não ser referirá a nenhum ponto-e-vírgula. Apesar disso, uma das regras deve ser que se uma mensagem de erro aparecer sobre algo que parece correto, devemos verificar se todas as declarações ao seu redor estão terminadas propriamente.

Outro erro comum é o erro "write error while sorting" depois que o *FORM* terminar a execução. Isto indica, usualmente, que ou o usuário tentou escrever ao redor de alguma quota ou que o disco está cheio. Analisando as estatísticas devem revelar qual é o caso. Sobre o sistema UNIX, o uso da opção "TEMPDIR" pode resolver este problema tendo arquivos intermediários escritos no diretório /tmp, apesar da experiência mostrar que por misteriosas razões, muitos administradores de sistemas gostam de colocar o diretório tmp em uma partição menor.

Os casos mais freqüentes de terminação anormal do trabalho são loops infinitos.

Em linguagens como FORTRAN, C, ou PASCAL, um loop infinito mantém um programa rodando até alguém decidir terminá-lo. Este não é o caso do *FORM*. Se o buffer estiver exausto (o que usualmente acontece em grandes sistemas) uma mensagem de erro como a do primeiro exemplo será impressa. O usuário pode não resolver isso rapidamente e apenas aumentar o problema usando o arquivo "setup".

Neste caso, o *FORM* reclamará novamente, mas agora, talvez sobre um outro buffer. Neste caso, a falta de memória torna-se o problema e o *FORM* terminará o programa abruptamente. Este será sempre um loop infinito.

Quando vários casos são considerados numa declaração complicada da estrutura IF, pode ser que isso pareça que todas as possibilidades são convertidas. Neste IF alguém gostaria que o programa desse uma mensagem de erro, e com isso, terminar a execução. A mensagem não existe ainda, mas a interrupção pode ser arranjada como se segue:

```
else;
    multiply 1/(1-1);
endif;
```

A divisão por zero só é notada quando a multiplicação é necessária. Isto não é o caso de $\frac{1}{0}$, porque o compilador tenta construir uma fração própria e

CAPíTULO 8 CONVIVENDO COM ERROS

trabalhar com a divisão ilegal.

Capítulo 9

Substituições Polinomiais

Usualmente, polinômios e variáveis comutativas são substituídas usando coeficientes binomiais, quando potências de polinômios estão envolvidas. Um exemplo simples é:

```
s a,b;
L F = (a+b)^4;
print;
.clear

F =
   4*a*b^3 + 6*a^2*b^2 + 4*a^3*b + a^4 + b^4;
```

Neste caso, o *FORM* gera somente os cinco termos na resposta, ao invés dos dezesseis termos. Quando existem objetos não-comutativos (funções, subexpressões) envolvidas, o método de força bruta é usado ao invés da expansão binomial. Na substituição da expansão, o uso de coeficientes binomiais não é suficiente. Uma expressão contendo m-termos elevados a n-potência, gerará $\frac{(n+m-1)!}{n!(m-1)!}$ termos antes do *FORM* poder coletá-los e colocá-los em ordem. Quando m e n são da ordem de 20, o método é claramente impraticável.

O método prático será ilustrado no seguinte exemplo. Tiraremos a expansão de $\ln(1+x)$ acima de 40 termos e substituiremos neste a expansão para $\exp(z) - 1$. O resultado deve ser, é claro, z, depois do término de todos os cálculos. Este resultado trivial permite-nos checar que o algoritmo é correto e que as rotinas aritméticas do *FORM* estão funcionando propriamente. As expansões que precisamos são:

CAPíTULO 9 SUBSTITUIÇÕES POLINOMIAIS

$$\ln(1+x) = \sum_{i=1}^{n} (-1)^{i+1} \frac{x^i}{i}$$

e

$$e^z - 1 = \sum_{i=1}^{n} \frac{z^n}{n!}$$

Esta última expansão tende a ser modificada, pois quando n é muito grande, o resultado pode ser catastrófico. Usaremos a fórmula "telescopic":

$$e^z - 1 = z\left(1 + \frac{z}{2}\left(1 + \frac{z}{3}\left(1 + \frac{z}{4}(1...)\right)\right)\right)$$

Esta fórmula é aplicada pela substituição sucessivas.

```
id x = y*z;
id y = 1 + y*z/2;
id y = 1 + y*z/3;
id y = 1 + y*z/4;
...
id y = 1 + y*z/n;
id y = 1;
```

O uso do pré-processador para fazer loops é o mais conveniente ao programa. Note também que o uso da variável pré-processadora n indica a profundidade da expansão e o corte da potência de z.

```
#define n ''40''
s x,y,z(:'n'),i;
L F = sum_(i,1,'n',-(-x)^i/i);
id x= y*z;
#do i = 2, 'n'
id y = 1 + y*z/'i';
#enddo
id y = 1;
print;
.end
```

Este exemplo fornece resposta z, mas toma muito mais tempo para fazê-lo. O programa torna-se quarenta vezes mais rápido quando é construído sobre a supressão de termos após cada substituição. Isto é feito adicionando-se a instrução .sort dentro do último loop.

```
#do i = 2, 'n '
id y = 1 + y*z/'i';
.sort
#enddo
```

Agradecimentos

Nossa gratidão aos Profs. A. Heck, R.M. Doria e J.A. Helayël-Neto pelas discussões e sugestões; ao CNPq, pelas nossas bolsas de Iniciação Científica.