# A package of generating Feynman rules in GRACE system.

ACAT 2002
2002/06
KANEKO Toshiaki

---

## Contents

# 1 Introduction

- Problem
  - Many particles and vertices
    55 particles and 3553 vertices in MSSM.
  - Derivation of Feynman rules.
  - Typing of coupling constants for an automatic calculation system.
- Purpose
  - Generation of Feynman rules from a Lagrangian.
  - Breaking the symmetry of Lagrangian.
  - Suitable output for `GRACE` system.
- Method : two levels
  1. New symbolic language and interpreter
     $\Longleftarrow$ general method.
  2. Derivation of Feynman rules
     $\Longleftarrow$ specialized method.
     $\Longrightarrow$ user program written in this language.
- Preceding works
  - `LanHEP` by A. Semenov
  - Part of `FDC` by J.-X. Wang

# 2 Language

## 2.1 Data

- sets, symbols and variables

```
set A;
symbol a, b : A;
var x, y;

x = (a + 2*b(1) + 3*b(2, 3))**2;
y = x;
```

1. declare an abstract (commutative) set $A$.
2. let $a$ and $b$ being elements of set $A$. $(a, b \in A)$.
3. keep an expression $(a + 2b_1 + 3b_{2,3})^2$ in $x$.
4. copy the contents of $x$ to $y$.

- A *symbol* is data corresponding to a mathematical symbol.
- A *symbol* can be used with indices (arguments).
- A *symbol* is an element of *set* (type of *symbol*).
- A *variable* is a holder of data.
- A *variable* has no type.
- A *variable* and *symbols* cannot be mixed.

- non-commutative symbols

```
set P : NonCom;
set Q : NonCom;
symbol p1, p2 : P;
symbol q1, q2 : Q;
var x;

x = p1*q2*2*p2*q1;
    % ==> x = 2*p1*p2*q2*q1
```

  - `p1` and `p2` are not commutable.
  - `q1` and `q2` are not commutable.
  - `p1` and `q1` are commutable.

- array and list

```
var x = newArray(10);
var y, z;
var i;

x[0] = 1;
x[1] = 1;
for(i = 2; i < 10; i = i + 1) {
    x[i] = x[i-2] + x[i-1];
}
y = [1, 2, 3];
z = cons(0, y);    % ==> z = [0, 1, 2, 4]
```

  - An *array* is a set of holders of data
  - A *list* is a set of sequential data used as in LISP.

- record (struct)

```
symbol a;
var x = 2*a;
var y = record {
    var    p, q;

    p = 3*a*x;
    q = 4*p;
};
var u, v, w;

u = y.p / x;        % ==> u = 3*a

w = "q";
v = y.[w] / x;      % ==> v = 12*a
```

  - A *record* is a set of *fields* (named variables).
  - A *fields* can be specified by a character string (one can specify *fields* dynamically).

- local symbol

```
set A;
symbol sum, x, y;
var a, b;
a = sum( record {
              symbol i : A;
              var value;

              value = x(i)*y(i);
         });
b = (1 + a)*(1 + a);
```

  - A *field* of *record* may be a symbol (*local symbol*).
  - The contents of `a` is $\Sigma_{i \in A} x_i y_i$, where $i$ is local to this expression.
  - evaluation of `b = (1 + a)*(1 + a);`
    1. create two copies (different objects) of `a`.
    2. multiply one `1 + a` and another `1 + a`.
    3. store result to `b`.

## 2.2 Control

- Branch and loop

```
var e, i, s, x;

foreach(e; [1, 3, 21, 4]) {
    x = 0;
    s = 0;
    for(i = 1; i <= e; i = i + 1) {
        s = s + i;
    }
    if(x < s) {
        x = s;
    }
}
print("x = ", x);    % => x = 231
```

  - Statements with keywords `if`, `while` and `for` are similar to ones defined in many programming languages.
  - Statement with keyword `foreach` is similar to one defined in **`perl`** (repeat for each element in a list).

- Functions

```
var sqr = func (a)
{
    var b;
    b = a*a;
    return b;
};

symbol x;
var    f, a, b;

a = sqr(x);       % ==> a = x**2
f = sqr;
b = f(2);         % ==> b = 4
```

  - As `function` is a data, it can be assigned to a *variable*.
  - evaluation : `b = f(2);`
    1. evaluate `f` $\Longrightarrow$ `func(a) { ... }`.
    2. bind `2` to parameter `a`.
    3. calculate the return value $\Longrightarrow$ `4`.
    4. assign `4` to `b`

- symbol substitution

```
symbol x, y;
var    a, b;

a = x + x(1)*x(1) + x(2)*y(2);
b = substSymbol(a, x, y);
      % ==> b = y + y(1)**2 + y(2)**2
```

  – `substSymbol(a, x, y)` means $a\,|_{x=y}$.

### 2.3 actions

- action of a symbol

```
var sqr = func (a)
{
    return a*a;
};

symbol x, y;
var    a, b, c, d, e;

% direct application of function
a = sqr(2);       % ==> a = 4

% by substitution of symbol
b = y(2);         % ==> b = y(2)
c = substSymbol(b, y, sqr);
                  % ==> c = 4
% by action
y.action = sqr;
d = b;            % ==> d = 4;

% clear action
y.action = Null;
e = b;            % ==> e = y(2)
```

- A *symbol* has predefined *fields* (named variable) like a *record*.
- A *field* named `action` controls the behavior of the *symbol*.
- The value of `action` can be *function* data or `Null` which means empty.
- When its value is `function` data and the `symbol` is used with arguments, the `function` is called with the arguments.
- evaluation : `y.action = sqr; d = b;`
  1. assign `func(a) { return a*a; }` to the *field* `action` of *symbol* `y`.
  2. evaluate a copy of expression `b = y(2)`.
  3. call `y.action(2)`.
  4. evaluate `2*2` $\Longrightarrow$ `4`.
  5. assign `4` to variable `d`.

- action of non-commutative product

```
set Grassman  : NonCom;

var grsmprod = func(arg)
{
    ( sort terms in product 'arg' )
    ( if two terms are same, return 0 )
    return (sign of permutation)*
           (sorted product);
}

Grassman.ncprod = grsmprod;

symbol g : Grassman;
symbol p, q : NonCom;
var v;

v = g(2)*p*g(p)*q*g(5)*g(1);
    % ==> v = ((-1)*p*q*g(1)*g(2)*g(5)*g(p))
v = v*g(2);
    % ==> v = 0
```

- A *set*, like *symbol*, has predefined *fields*.
- A *field* named `ncprod` controls the behavior product of elements of this set.
- The value of `ncprod` can be *function* data or `Null`.
- When its value is a *function* data and a product of elements of the *set* appeared in an expression, the *function* is called with the produce as the argument. The original product is replaced by the return value of the *function*.
- evaluation of `v = g(2)*p*g(p)*q*g(5)*g(1);`
  1. reorder product with regard to the non-commutativity.
     $\Longrightarrow$ `(p*q)*(g(2)*g(p)*g(5)*g(1))`
  2. call `Grassman.ncprod(g(2)*g(p)*g(5)*g(1))`.
  3. the function reorders the product with appropriate sign of the permutation.
     $\Longrightarrow$ `-g(1)*g(2)*g(5)*g(p)`
  4. the evaluation finishes by replacing the original product by the reordered one.
     $\Longrightarrow$ `(p*q)*(-g(1)*g(2)*g(5)*g(p))`

## Output of `grassman.gsr`

```
          ...
 0: --- input from file ---
 7< set Grassman  : NonCom;
13< var grsmprod = func(arg)
14< {
15<     var p, q, prod, cl, c, a, n, i, pp, sgn;
16<     var mod = False;
18<     prod = 1;
19<     if(isProduct(arg)) {
20<         pp  = sortList(deProduct(arg));
21<         sgn = first(pp);
22<         if(sgn == 0) {
23<             return 0;
24<         }
25<         foreach(p; pp) {
26<             if(isPower(p)) {
27<                 cl = dePower(p);
28<                 q  = first(cl);
29<                 n  = second(cl);
30<                 if(!isNumber(n) || n < 1) {
31<                     q = p;
32<                 } else {
33<                     return 0;
34<                 }
35<             } else {
36<                 q = p;
37<             }
38<             Grassman.ncprod = Null;
39<             prod = prod * q;
40<             Grassman.ncprod = grsmprod;
41<             if(isNumber(prod) && prod == 0) {
42<                 return 0;
43<             }
44<         }
45<     } else {
46<         return Undef;
47<     }
48<     return prod;
49< };
```

```
  51< Grassman.ncprod = grsmprod;
  53< symbol g : Grassman;
  54< symbol p, q : NonCom;
  55< symbol r;
  57< var v;
  59< v = g(1)*2*g(5)*g(r)*g(2);
  60< print("v = ", v);
v = (2*g(1)*g(2)*g(5)*g(r))
  62< v = g(2)*q*g(5)*p*g(r)*g(1);
  63< print("v = ", v);
v = ((-1)*p*q*g(1)*g(2)*g(5)*g(r))
  65< v = substSymbol(v, r, 2);
  66< print("v = ", v);
v = 0
  68< v = g*g(1)*g(5)*g(r)*g(2);
  69< print("v = ", v);
v = (g*g(1)*g(2)*g(5)*g(r))
  71< v = g(2)*p*g(r)*q*g(5)*g(1);
  72< print("v = ", v);
v = ((-1)*p*q*g(1)*g(2)*g(5)*g(r))
  74< v = v*g(2);
  75< print("v = ", v);
v = 0
  77< v = g(2);
  78< print("v = ", v);
v = g(2)
  80< end;

End === Total 2048, (  32.000 KBytes) 3 + 210(env) used, 1833 free (0.895)
End === id table    :       12 defined,        0 used
End === identifiers :      106 defined,        0 used
End === set         :        3 defined,        0 used,        0 refs
End === records     :        0 defined,        0 used,        0 refs
End === arrays      :        0 defined,        0 used,        0 refs
End === files       :        3 defined,        0 used
End === Time        : 0.023438 sec (total 0.023438 sec)
```

# 3 Generation of Feynman rules

- main control

```
var main = func()
{
    var l0, lsb;
    var i, a, b, j;

    init();

    FieldComp();

    l0 = lag();

    lsb = expand(l0);
    lsb = lagExpand(lsb);

    decomp(lsb);
    evalCF();    % for QCD
    prVert();

    outmdl("out.mdl");
    outfin("out.fin");
};
```

- Definition of fields (particles).
  $\Longrightarrow$ model dependent
- Definition of Lagrangian.
  $\Longrightarrow$ model dependent
- Expand Lagrangian in terms of component fields.
  $\Longrightarrow$ common package
- Decompose Lagrangian to the set of vertices.
  $\Longrightarrow$ common package
- Extract coefficients of fields (differentiate by fields).
  $\Longrightarrow$ common package
- Print the result.

- Special treatment
  - Derivative coupling
    $\Longrightarrow$ *action* of a *symbol*.
  - Summation of dummy indices
    $\Longrightarrow$ *record* with *local symbol*.
  - Separation of coefficients of fields in vertices.
    $\Longrightarrow$ differentiation in terms of fields.

- Models

  We can generate Feynman rules for

  - QED
    Tribial internal symmetry
    $U(1)$
  - Electro-weak theory
    Broken internal symmetry
    $SU(2) \times U(1) \Rightarrow U(1)$
  - QCD
    Unbroken internal symmetry
    $SU(3)$

## First part of `qcd.gsr`

```
%============================================================
% summation
%
include "testl/sum.gsr";
include "testl/lag.gsr";

%************************************************************
% Model dependent part
%------------------------------------------------------------
% procedures to be defined
%     group      = func() { ... };
%     coupling   = func() { lOrders = [...]; lCoupls = [...] };
%     fields     = func() { lParticles = [...] };
%     lagrangian = func() { ... };
%============================================================
% group
%------------------------------------------------------------

group = func()
{
};

%------------------------------------------------------------
% flavor
set FlavorIndex : MatrixIndex;
const  NFMax    = 6;
const  NFTotal  = 2;

%============================================================
% color group
symbol cF;    % structure constant.
symbol cT;    % fundamental representation
symbol CF;    % structure constant.

%============================================================
% Coupling constants
symbol QCD;
```

```
symbol gc;

symbol lambda;
symbol amlp;

%-------------------------------------------------------------
coupling = func()
{
    lOrders = [QCD];
    lCoupls = [gc];

    gc.value = record { var order = [ 1]; };
};

%=============================================================
% Fields
%-------------------------------------------------------------
% Vectors
% symbol Gl : (LorentzIndex) --> BField;
symbol Gl : (LorentzIndex, Color8) --> BField;

%-------------------------------------------------------------
% Scalar

%-------------------------------------------------------------
% Dirac fermion
symbol Uq, UqBar : FField;
symbol Dq, DqBar : FField;

var    Psi, PsiBar;

%-------------------------------------------------------------
% Ghosts
symbol CG, CGBar : (Color8) --> GField;

%-------------------------------------------------------------
% Array of fermions.
var FieldComp = func()
{
    var q;
```

```
    Psi    = newArray(NFTotal);
    PsiBar = newArray(NFTotal);

    Psi[0] = Uq;
    Psi[1] = Dq;

    PsiBar[0] = UqBar;
    PsiBar[1] = DqBar;
};

%------------------------------------------------------------
% Fields

fields = func()
{
    var f, i;

    % table of particles and anti-particles
    lPandA = [
         UqBar,    DqBar,
         Uq,       Dq,
         Gl,
         CG,       CGBar
    ];

    % vectors
    Gl.value = record {
        var particle  = Gl;
        var antip     = Gl;
        var pname     = "gluon";
        var sname     = "g";
        var gname     = "g";

        var gnameg    = "g";
        var mom       = P;
        var ptype     = NVector;
        var charge    = 0;
        var color     = "8";
        var massless  = True;
```

```
    var mass      = "amg";
    var width     = "0";
    var pcode     = "1";
    var kfcode    = "21";
    var gauge     = "gl";
    var furry     = False;
    var symmetry  = cF;

    var pselect   = Null;
};

% leptons

Uq.value = record {
    var particle  = Uq;
    var antip     = UqBar;
    var pname     = "u";
    var sname     = "u";
    var gname     = "u";

    var gnameg    = "\\bar{u}";
    var mom       = P;
    var ptype     = NDirac;
    var charge    = 2/3;
    var color     = "3";
    var mass      = "amuq(1/3)";
    var width     = "aguq(1/3)";
    var massless  = False;
    var pcode     = "61";
    var kfcode    = "2";
    var gauge     = Null;
    var furry     = False;
  % var symmetry  = cT;
    var symmetry  = Null;

    var pselect   = Null;
};
UqBar.value = record {
    var particle  = Uq;
    var antip     = UqBar;
```

```
        var pname     = "u-bar";
        var sname     = "~u";
        var gname     = "\\bar{u}";
    };

...
```

## A part of `qcd.gsr`

```
        ...

%-------------------------------------------------------------
% F_{\mu\nu}
%
var F = func(a, m, n)
{
    symbol b, c;
    var ans;

    ans = Ds(m, Gl(n, a)) - Ds(n, Gl(m, a))
          - gc*sum([b, c], cF(a,b,c)*Gl(m, b)*Gl(n, c));
    return ans;
};

var DD;

DD = func(m, p)
{
%   ans = Ds(m, p(fl)) - I*sum([k],q*gc*p(fl)*Gl(m, a)*cT(a));

    symbol a;
    var ans, k, q;

    if(!isSymbol(p)) {
        print("*** DDl : not symbol");
        exit(1);
    } else if(!isRecord(p.value)) {
        print("*** DDl : ", p, ".value is not record");
        exit(1);
    } else if(!inRecord(p.value, "charge")) {
        print("*** DDl : no ", p, ".value.charge");
        exit(1);
    }

    ans = Ds(m, p) + I*gc*sum([a], Gl(m, a)*cT(a))*p;
    return ans;
};
```

```
%-----------------------------------------------------------
% Lagrangian

% Vector
var lagG = func()
{
    symbol a;
    var Lg;

    Lg = - 1/4 * sum([a], F(a, mc, nc)*F(a, m, n));
    return Lg;
};

% Fermion -- Vector

var lagF = func(nf)
{
    var Lf, f, i;

    Lf = 0;
    for(f = 0; f < nf; f = f + 1) {
        Lf = Lf + I*PsiBar[f]*gamma(m)*DD(m, Psi[f])
                - amlp(f)*PsiBar[f]*Psi[f];
    }
    return Lf;
};

% Gauge fixing term

var lagGF = func()
{
    symbol a;
    var Lgf;

    Lgf = - 1/(2*lambda)*sum([a], DsBar(m, Gl(m, a))*Ds(n, Gl(n, a)));
    return Lgf;
};

var lagFP = func()
```

```
{
    symbol a, b, c;
    var Lfp;

    Lfp = - sum([a], CGBar(a)*Ds2(m, CG(a)))
          + gc*sum([a, b, c], CGBar(a)*cF(a, b, c)*Ds(m, Gl(m, b)*CG(c)));

    return Lfp;
};


% sum

var lag = func()
{
    var L;
    var NF;

    NF = NFTotal;

    L = 0;
    L = L + lagG();
    L = L + lagF(NF);
    L = L + lagGF();
    L = L + lagFP();

    return L;
};
```

# Generated Feynman rules for electro-weak theory

```
%%% prVert : nVertex = 98
%-----------------------------------------------------
fcode = 0[]

ord   = [0];
coupl = (
  (1/2)/e^2*amw^2*amh^2
  +(-1/2)/e^2*amw^4/amz^2*amh^2
);
%-----------------------------------------------------
fcode = 7[ Scalar Scalar]

ord   = [0, chiPlus, chiMinus];
coupl = (
  P(m, chiMinus)^2
  +(-1)*amw^2*alphW
);

ord   = [0, chi3, chi3];

      ...

%-----------------------------------------------------
fcode = 14[ Fermion Fermion]

ord   = [0, NuElBar, NuEl];
coupl = (
  (-1)*amnu(1)
  +P(m, NuEl)*gamma(m)
);

      ...

%-----------------------------------------------------
fcode = 28[ Ghost Ghost]

ord   = [0, CPlusBar, CPlus];
coupl = (
```

```
  P(m, CPlus)^2
  +(-1)*amw^2*alphW
);

      ...

%---------------------------------------------------
fcode = 35[ Vector Vector]

ord   = [0, WPlus, WMinus];
coupl = (
  (-1)*Delta(m0, m1)*P(m, WMinus)^2
  +P(m0, WMinus)*P(m1, WMinus)
  +amw^2*Delta(m0, m1)
  +(-1)/alphW*P(m0, WMinus)*P(m1, WMinus)
);

      ...

%---------------------------------------------------
fcode = 43[ Scalar Scalar Scalar]

ord   = [1, phi, chiPlus, chiMinus];
coupl = ((-1/2)*e/amw*amz/amzw*amh^2);

      ...

%---------------------------------------------------
fcode = 85[ Scalar Fermion Fermion]

ord   = [1, chi3, ElBar, El];
coupl = ARef(Array[2]={
  ((-1/2)*I*e/amw*amz/amzw*amlp(1)),
  ((1/2)*I*e/amw*amz/amzw*amlp(1))
});

      ...

%---------------------------------------------------
fcode = 89[ Vector Fermion Fermion]
```

```
ord   = [1, Z, NuElBar, NuEl];
coupl = ARef(Array[2]={
  (
    (1/2)*e*amw/amzw
    +(1/2)*e/amw*amzw
  ),
  0
});

      ...

%-----------------------------------------------------
fcode = 169[ Scalar Ghost Ghost]

ord   = [1, chi3, CPlusBar, CPlus];
coupl = ((1/2)*I*e*amw*amz/amzw*alphW);

      ...

%-----------------------------------------------------
fcode = 173[ Vector Ghost Ghost]

ord   = [1, WPlus, CPlusBar, CZ];
coupl = (e*amw/amzw);

      ...

%-----------------------------------------------------
fcode = 187[ Scalar Scalar Vector]

ord   = [1, phi, chi3, Z];
coupl = (
  (-1/2)*I*e*amw/amzw
  +(-1/2)*I*e/amw*amzw
);

      ...

%-----------------------------------------------------
```

```
fcode = 211[ Scalar Vector Vector]

ord   = [1, chiPlus, WMinus, A];
coupl = (I*e*amw);

      ...

%-----------------------------------------------------
fcode = 215[ Vector Vector Vector]

ord   = [1, WPlus, WMinus, Z];
coupl = ((-1)*e*amw/amzw);

ord   = [1, WPlus, WMinus, A];
coupl = ((-1)*e);
%-----------------------------------------------------
fcode = 259[ Scalar Scalar Scalar Scalar]

ord   = [2, chiPlus, chiMinus, chi3, chi3];
coupl = ((-1/4)*e^2/amw^2*amz^2/amzw^2*amh^2);

      ...

%-----------------------------------------------------
fcode = 1267[ Scalar Scalar Vector Vector]

ord   = [2, phi, chiPlus, WMinus, Z];
coupl = ((-1/2)*I*e^2/amw*amz);

      ...

%-----------------------------------------------------
fcode = 1295[ Vector Vector Vector Vector]

ord   = [2, WPlus, WMinus, A, Z];
coupl = (e^2*amw/amzw);

ord   = [2, WPlus, WMinus, A, A];
coupl = (e^2);
```

```
ord   = [2, WPlus, WMinus, Z, Z];
coupl = (e^2*amw^2/amzw^2);

ord   = [2, WPlus, WPlus, WMinus, WMinus];
coupl = ((-1)*e^2*amz^2/amzw^2);
%-----------------------------------------------------
+++ written in out.mdl +++
1964< end;
1964: --- input closed ---
1964: --- input closed ---

End === Total 75776, (   1.156 MBytes) 3 + 612(env) used, 75087 free (0.991)
End === id table    :      894 defined,        0 used
End === identifiers :     2833 defined,        0 used
End === set         :       69 defined,        0 used,        0 refs
End === records     :      169 defined,        0 used,        0 refs
End === arrays      :      737 defined,        0 used,        0 refs
End === files       :        5 defined,        0 used
End === Time        : 28.109375 sec (total 28.109375 sec)

       28.17 real        28.09 user         0.01 sys
```

## Generated Feynman rules for QCD

```
%%% prVert : nVertex = 9
%-----------------------------------------------------
fcode = 14[ Fermion Fermion]

ord   = [0, UqBar, Uq];
coupl = (
  (-1)*amlp(0) +P(m, Uq)*gamma(m)
);

ord   = [0, DqBar, Dq];
coupl = (
  (-1)*amlp(1) +P(m, Dq)*gamma(m)
);
%-----------------------------------------------------
fcode = 28[ Ghost Ghost]

ord   = [0, CGBar, CG];
coupl = (Delta(cl0, cl1)*P(m, CG(cl0))^2);
%-----------------------------------------------------
fcode = 35[ Vector Vector]

ord   = [0, Gl, Gl];
coupl = (
  Delta(m0, m1)*Delta(cl0, cl1)*P(m, Gl(m0, cl0))^2
  +(-1)*Delta(cl0, cl1)*P(m0, Gl(m1, cl0))*P(m1, Gl(m0, cl0))
  +(-1)/lambda*Delta(cl0, cl1)*P(m0, Gl(m0, cl0))*P(m1, Gl(m1, cl0))
);
%-----------------------------------------------------
fcode = 89[ Vector Fermion Fermion]

ord   = [1, Gl, UqBar, Uq];
coupl = ARef(Array[2]={
  ((-1)*gc*cT(cl0)),
  ((-1)*gc*cT(cl0))
});

ord   = [1, Gl, DqBar, Dq];
coupl = ARef(Array[2]={
```

```
  ((-1)*gc*cT(cl0)),
  ((-1)*gc*cT(cl0))
});
%-----------------------------------------------------
fcode = 173[ Vector Ghost Ghost]

ord   = [1, Gl, CGBar, CG];
coupl = (
  I*gc*P(m0, Gl(m0, cl0))*CF(cl0, cl1, cl2)
  +I*gc*P(m0, CG(cl2))*CF(cl0, cl1, cl2)
);
%-----------------------------------------------------
fcode = 215[ Vector Vector Vector]

ord   = [1, Gl, Gl, Gl];
coupl = (
  I*gc*Delta(m0, m1)*P(m2, Gl(m0, cl0))*CF(cl0, cl1, cl2)
  +(-1)*I*gc*Delta(m0, m1)*P(m2, Gl(m0, cl1))*CF(cl0, cl1, cl2)
  +(-1)*I*gc*Delta(m0, m2)*P(m1, Gl(m0, cl0))*CF(cl0, cl1, cl2)
  +I*gc*Delta(m0, m2)*P(m1, Gl(m0, cl2))*CF(cl0, cl1, cl2)
  +I*gc*Delta(m1, m2)*P(m0, Gl(m1, cl1))*CF(cl0, cl1, cl2)
  +(-1)*I*gc*Delta(m1, m2)*P(m0, Gl(m1, cl2))*CF(cl0, cl1, cl2)
);
%-----------------------------------------------------
fcode = 1295[ Vector Vector Vector Vector]

ord   = [2, Gl, Gl, Gl, Gl];
coupl = (
  (-1)*gc^2*Delta(m0, m1)*Delta(m2, m3)*sum([a;], (CF(cl0, cl3, a)*CF(
  cl1, cl2, a));)
  +(-1)*gc^2*Delta(m0, m1)*Delta(m2, m3)*sum([a;], (CF(cl0, cl2, a)*CF(
  cl1, cl3, a));)
  +gc^2*Delta(m0, m2)*Delta(m1, m3)*sum([a;], (CF(cl0, cl3, a)*CF(cl1,
  cl2, a));)
  +(-1)*gc^2*Delta(m0, m2)*Delta(m1, m3)*sum([a;], (CF(cl0, cl1, a)*CF(
  cl2, cl3, a));)
  +gc^2*Delta(m0, m3)*Delta(m1, m2)*sum([a;], (CF(cl0, cl2, a)*CF(cl1,
  cl3, a));)
  +gc^2*Delta(m0, m3)*Delta(m1, m2)*sum([a;], (CF(cl0, cl1, a)*CF(cl2,
  cl3, a));)
```

```
);
%-------------------------------------------------
+++ written in out.mdl +++
 717< end;
 717: --- input closed ---
 717: --- input closed ---

End === Total 320512, (   4.891 MBytes) 3 + 612(env) used, 319584 free (0.997)
End === id table    :    18084 defined,        0 used
End === identifiers :    42234 defined,        0 used
End === set         :       31 defined,        0 used,        0 refs
End === records     :    10989 defined,        0 used,        0 refs
End === arrays      :       69 defined,        0 used,        0 refs
End === files       :        5 defined,        0 used
End === Time        : 9.718750 sec (total 9.718750 sec)

        9.75 real         9.70 user         0.02 sys
```

# 4 Summary

- Summary
  - symbolic language and interpreter are developed.
    - Flexible data structures.
    - Standard control statements.
    - Substitution of sub-expression with predefined fields of symbols and sets.
  - Feynman rules are generated for
    - QED
    - Electro-weak
    - QCD
- Problems
  - improve the language
  - introduce BRST symmetry
  - generate Faddeev-Popov ghost term
  - generate Feynaman rules for SUSY
  - start from super-fields
  - ...