Symbolic Programming Examples

Thomas Hahn

Max-Planck-Institut für Physik
München

List of Examples

- **Antisymmetric Tensor**
  Built-in in FORM, easy in Mathematica.
- **Application of Momentum Conservation**
  Easy in Mathematica, complicated in FORM.
- **Abbreviationing**
  Easy in Mathematica, new in FORM.
- **Simplification of Colour Structures**
  Different approaches.
- **Calculation of a Fermion Trace**
  Built-in in FORM, complicated in Mathematica.
- **Tensor Reduction**

Reference Books, Formula Collections

- V.I. Borodulin et al.
  CORE (Compendium of Relations)
  hep-ph/9507456.
- Herbert Pietschmann
  Formulae and Results in Weak Interactions
  Springer (Austria) 2nd ed., 1983.
- Andrei Grozin
  Using REDUCE in High-Energy Physics
  Cambridge University Press, 1997.

Antisymmetric Tensor

The **Antisymmetric Tensor in n dimensions** is denoted by $\varepsilon_{i_1 i_2 \dots i_n}$. You can think of it as a matrix-like object which has either -1, 0, or 1 at each position.

For example, the **Determinant** of a matrix, being a **completely antisymmetric** object, can be written with the ε-tensor:

$$\det A = \sum_{i_1,\dots,i_n=1}^{n} \varepsilon_{i_1 i_2 \dots i_n} A_{i_1 1} A_{i_2 2} \cdots A_{i_n n}$$

In practice, the ε-tensor is usually contracted, e.g. with vectors. We will adopt the following notation to avoid dummy indices:

$$\varepsilon_{\mu\nu\rho\sigma} p^\mu q^\nu r^\rho s^\sigma = \varepsilon(p, q, r, s)\,.$$

Antisymmetric Tensor in Mathematica

```
   (* implement linearity: *)

Eps[a___, p_Plus, b___] := Eps[a, #, b]&/@ p

Eps[a___, n_?NumberQ r_, b___] := n Eps[a, r, b]

   (* otherwise sort the arguments into canonical order: *)

Eps[args__] := Signature[{args}] Eps@@ Sort[{args}] /;
  !OrderedQ[{args}]
```

Momentum Conservation

Problem: Proliferation of terms in expressions such as

$$d = \frac{1}{(p_1 + p_2 - p_3)^2 + m^2}$$
$$= \frac{1}{p_1^2 + p_2^2 + p_3^2 + 2p_1p_2 - 2p_2p_3 - 2p_1p_3 + m^2},$$

whereas if $p_1 + p_2 = p_3 + p_4$ we could have instead

$$d = \frac{1}{p_4^2 + m^2}.$$

In Mathematica: just do `d /. p1 + p2 - p3 -> p4`.
Problem: FORM cannot replace sums.

Momentum Conservation in FORM

Idea: for each expression x, add and subtract a zero, i.e. form

$\{x, y = x + \sigma, z = x - \sigma\}$, where e.g. $\sigma = p_1 + p_2 - p_3 - p_4$,

then select the shortest expression. But: how to select the shortest expression (in FORM)?

Solution: add the number of terms of each argument, i.e.

$$\{x, y, z\} \to \{\overset{1}{x}, \overset{2}{y}, \overset{3}{z}, \overset{4}{n_x}, \overset{5}{n_y}, \overset{6}{n_z}\}\,.$$

Then sort n_x, n_y, n_z, but when exchanging n_a and n_b, exchange also a and b:

```
symm 'foo' (4,1) (5,2) (6,3);
```

This unconventional sort statement is rather typical for FORM.

Momentum Conservation in FORM

```
#procedure Shortest(foo)

id `foo'([x]?) = `foo'([x], [x] + `MomSum', [x] - `MomSum');

* add number-of-terms arguments
id `foo'([x]?, [y]?, [z]?) = `foo'([x], [y], [z],
  nterms_([x]), nterms_([y]), nterms_([z]) );

* order according to the nterms
symm `foo' (4,1) (5,2) (6,3);

* choose shortest argument
id `foo'([x]?, ?a) = `foo'([x]);

#endprocedure
```

Abbreviationing

One of the most powerful tricks to both **reduce the size** of an expression and **reveal its structure** is to substitute subexpressions by new variables.

The essential function here is `Unique` with which new symbols are introduced. For example,

```
Unique["test"]
```

generates e.g. the symbol `test1`, which is **guaranteed not to be in use so far.**

The `Module` function which implements lexical scoping in fact uses `Unique` to rename the symbols internally because Mathematica can really do dynamical scoping only.

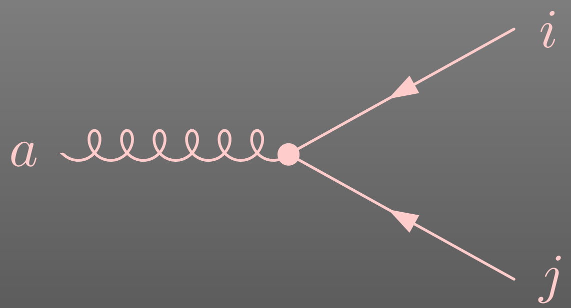
Abbreviationing in Mathematica

```
$AbbrPrefix = "c"

abbr[expr_] := abbr[expr] = Unique[$AbbrPrefix]

   (* abbreviate function *)
Structure[expr_, x_] := Collect[expr, x, abbr]

   (* get list of abbreviations *)
AbbrList[] := Cases[DownValues[abbr],
  _[_[_[f_]], s_Symbol] -> s -> f]

   (* restore full expression *)
Restore[expr_] := expr /. AbbrList[]
```

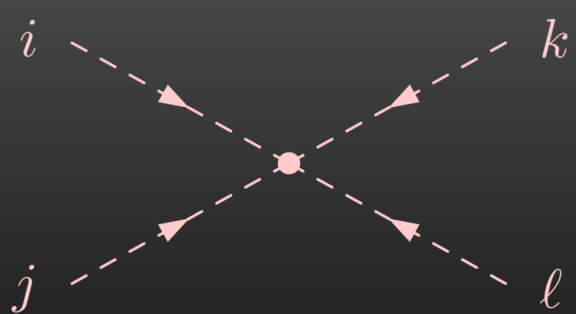
Colour Structures

In Feynman diagrams four type of Colour structures appear:

Natural Representation

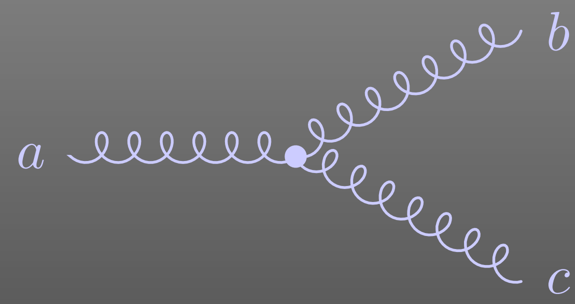


$\sim T^a_{ij}$ = `SUNT[`a`,`i`,`j`]`

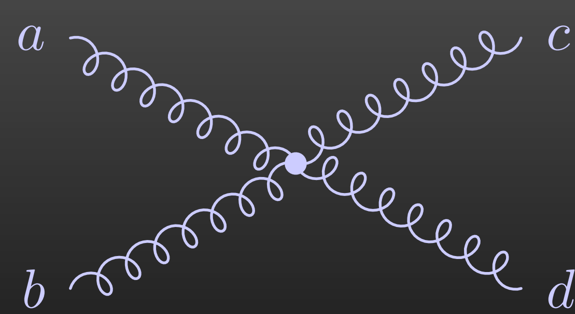


$\sim T^a_{ij} T^a_{k\ell}$ = `SUNTSum[`i`,`j`,`k`,`ℓ`]`

Adjoint Representation



$\sim f^{abc}$ = `SUNF[`a`,`b`,`c`]`



$\sim f^{abx} f^{xcd}$ = `SUNF[`a`,`b`,`c`,`d`]`

Unified Notation

The `SUNF`'s can be converted to `SUNT`'s via

$$f^{abc} = 2\mathrm{i}\big[\mathrm{Tr}(T^c T^b T^a) - \mathrm{Tr}(T^a T^b T^c)\big]\,.$$

We can now represent all colour objects by just `SUNT`:

- $\texttt{SUNT}[i,j] = \delta_{ij}$
- $\texttt{SUNT}[a,b,\ldots,i,j] = (T^a T^b \cdots)_{ij}$
- $\texttt{SUNT}[a,b,\ldots,0,0] = \mathrm{Tr}(T^a T^b \cdots)$

This notation again avoids unnecessary dummy indices. (Mainly namespace problem.)

For purposes such as the "large-N_c limit" people like to use SU(N) rather than an explicit SU(3).

Fierz Identities

The **Fierz Identities** relate expressions with **different orderings of external particles.** The Fierz identities essentially express completeness of the underlying matrix space.

They were originally found by Markus Fierz in the context of Dirac spinors, but can be generalized to any finite-dimensional matrix space [hep-ph/0412245].

For SU(N) (colour) reordering, we need

$$T^a_{ij} T^a_{k\ell} = \frac{1}{2}\left(\delta_{i\ell}\delta_{kj} - \frac{1}{N}\delta_{ij}\delta_{k\ell}\right).$$

Cvitanovich Algorithm

For an **Amplitude:**

- convert all colour structures to (generalized) `SUNT` objects,
- simplify as much as possible, i.e. use the Fierz identity on all internal gluon lines.

For a **Squared Amplitude:**

- use the Fierz identity for SU(N) to get rid of all `SUNT` objects.

For “hand” calculations, a pictorial version of this algorithm exists in the literature.

Colour Simplify in FORM

```
* introduce dummy indices for the traces
repeat;
  once SUNT(?a, 0, 0) = SUNT(?a, DUMMY, DUMMY);
  sum DUMMY;
endrepeat;

* take apart SUNTs with more than one T
repeat;
  once SUNT(?a, [a]?, [b]?, [i]?, [j]?) =
    SUNT(?a, [a], [i], DUMMY) * SUNT([b], DUMMY, [j]);
  sum DUMMY;
endrepeat;

* apply the Fierz identity
id SUNT([a]?, [i]?, [j]?) * SUNT([a]?, [k]?, [l]?) =
  1/2 * SUNT([i], [l]) * SUNT([j], [k]) -
  1/2/(`SUNN') * SUNT([i], [j]) * SUNT([k], [l]);
```

Translation to Colour-Chain Notation

In colour-chain notation we can distinguish two cases:

a) Contraction of different chains:

$$\langle A|\, T^a \,|B\rangle \, \langle C|\, T^a \,|D\rangle = \frac{1}{2} \left(\langle A|D\rangle \, \langle C|B\rangle - \frac{1}{N} \, \langle A|B\rangle \, \langle C|D\rangle \right),$$

b) Contraction on the same chain:

$$\langle A|\, T^a \,|B|\, T^a \,|C\rangle = \frac{1}{2} \left(\langle A|C\rangle \, \mathrm{Tr}\, B - \frac{1}{N} \, \langle A|\, B \,|C\rangle \right).$$

Colour Simplify in Mathematica

```
   (* same-chain version *)
sunT[t1___, a_Symbol, t2___, a_, t3___, i_, j_] :=
  (sunT[t1, t3, i, j] sunTrace[t2] -
    sunT[t1, t2, t3, i, j]/SUNN)/2

   (* different-chain version *)
sunT[t1___, a_Symbol, t2___, i_, j_] *
sunT[t3___, a_, t4___, k_, l_] ^:=
  (sunT[t1, t4, i, l] sunT[t3, t2, k, j] -
    sunT[t1, t2, i, j] sunT[t3, t4, k, l]/SUNN)/2

   (* introduce dummy indices for the traces *)
sunTrace[a__] := sunT[a, #, #]&[ Unique["col"] ]
```

Fermion Trace

Leaving apart problems due to γ_5 in d dimensions, we have as the main algorithm for the 4d case:

$$\begin{aligned}\mathrm{Tr}\,\gamma_\mu\gamma_\nu\gamma_\rho\gamma_\sigma\cdots = &+ g_{\mu\nu}\,\mathrm{Tr}\,\gamma_\rho\gamma_\sigma\cdots\\ &- g_{\mu\rho}\,\mathrm{Tr}\,\gamma_\nu\gamma_\sigma\cdots\\ &+ g_{\mu\sigma}\,\mathrm{Tr}\,\gamma_\nu\gamma_\rho\cdots\end{aligned}$$

This algorithm is recursive in nature, and we are ultimately left with

$$\mathrm{Tr}\,\mathbb{1} = 4\,.$$

(Note that this 4 is not the space-time dimension, but the dimension of spinor space.)

Fermion Trace in Mathematica

```
Trace4[mu_, g__] :=
Block[ {Trace4, s = -1},
  Plus@@ MapIndexed[
    ((s = -s) Pair[mu, #1] Drop[Trace4[g], #2])&,
    {g} ]
]

Trace4[] = 4
```

Tensor Reduction

The loop integrals corresponding to closed loops in a Feynman integral in general have a **tensor structure** due to **integration momenta in the numerator.** For example,

$$B_{\mu\nu}(p) = \int \mathrm{d}^d q \, \frac{q_\mu q_\nu}{\left(q^2 - m_1^2\right)\left((q-p)^2 - m_2^2\right)} \, .$$

Such tensorial integrals are rather unwieldy in practice, therefore they are reduced to linear combinations of Lorentz-covariant tensors, e.g.

$$B_{\mu\nu}(p) = B_{00}(p) \, g_{\mu\nu} + B_{11}(p) \, p_\mu p_\nu \, .$$

It is the **coefficient functions** B_{00} and B_{11} which are implemented in a library like LoopTools.

Tensor Reduction Algorithm

The first step is to convert the integration momenta in the numerator to an actual tensor, e.g. $q_\mu q_\nu \to N_{\mu\nu}$. FORM has the special command `totensor` for this:

```
totensor q1, NUM;
```

The next step is to take out $g_{\mu\nu}$'s in all possible ways. We do this in form of a sum:

$$N_{\mu_1\dots\mu_n} = \sum_{i=0,2,4,\dots}^{n} \pi(0)^i \sum_{\substack{\text{all } \{\nu_1,\dots,\nu_i\} \\ \in\{\mu_1,\dots,\mu_n\}}} g_{\nu_1\nu_2}\cdots g_{\nu_{i-1}\nu_i} N_{\mu_1\dots\mu_n\setminus\nu_1\dots\nu_i}$$

The $\pi(0)^i$ keeps track of the indices of the tensor coefficients, i.e. it later provides the two zeros for every $g_{\mu\nu}$ in the index, as in D_{0012}.

Tensor Reduction Algorithm

To fill in the remaining $\pi(i)$'s, we start off by **tagging the arguments** of the loop function, which are just the momenta. For example:

$$C(p_1, p_2, \ldots) \to \tau\big(\pi(1)p_1 + \pi(2)p_2\big)\, C(p_1, p_2, \ldots)$$

The temporary function τ keeps its argument, the 'tagged' momentum p, separate from the rest of the amplitude.

Now **add the indices** of $N_{\mu_1 \ldots \mu_n}$ to the momentum in τ:

$$\tau(p)\, N_{\mu_i \ldots \mu_n} = p_{\mu_i} \cdots p_{\mu_n} \,.$$

Finally, collect all π's into the tensor-coefficient index.

Tensor Reduction in FORM

```
totensor q1, NUM;

* take out 0, 2, 4... indices for g_{mu nu}
id NUM(?b) = sum_(DUMMY, 0, nargs_(?b), 2,
  pave(0)^DUMMY * distrib_(1, DUMMY, dd_, NUM, ?b));

* construct tagged momentum in TMP
id C0i([p1]?, [p2]?, ?a) = TMP(pave(1)*[p1] + pave(2)*[p2]) *
  C0i(MOM([p1]), MOM([p2] - [p1]), MOM([p2]), ?a);

* expand momentum
repeat id TMP([p1]?) * NUM([mu]?, ?a) =
  d_([p1], [mu]) * NUM(?a) * TMP([p1]);

* collect the indices
chainin pave;
```