Symbolic Programming Examples
Mathematica vs. FORM

Thomas Hahn

Max-Planck-Institut für Physik
München

Mathematica vs. FORM

Mathematica

- Much built-in knowledge,
- Big and slow (especially on large problems),
- Very general,
- GUI, add-on packages…

FORM

- Limited mathematical knowledge,
- Small and fast (also on large problems),
- Optimized for certain classes of problems,
- Batch program (edit-run cycle).

Mathematica

Expert Systems

In technical terms, Mathematica is an Expert System.
Knowledge is added in form of Transformation Rules.
An expression is transformed until no more rules apply.

Example:

```
myAbs[x_] := x /; NonNegative[x]
myAbs[x_] := -x /; Negative[x]
```

We get:

```
myAbs[3] ☞ 3
myAbs[-5] ☞ 5
myAbs[2 + 3 I] ☞ myAbs[2 + 3 I]
```

— no rule for complex arguments so far

```
myAbs[x] ☞ myAbs[x]
```

— no match either

Immediate and Delayed Assignment

Transformations can either be

- added "permanently" in form of Definitions,

```
norm[vec_] := Sqrt[vec . vec]
norm[{1, 0, 2}] ☞ Sqrt[5]
```

- applied once using Rules:

```
a + b + c /. a -> 2 c ☞ b + 3 c
```

Transformations can be Immediate or Delayed. Consider:

```
{r, r} /. r -> Random[] ☞ {0.823919, 0.823919}
{r, r} /. r :> Random[] ☞ {0.356028, 0.100983}
```

Mathematica is one of those programs, like TEX, where you wish you'd gotten a US keyboard for all those braces and brackets.

Almost everything is a List

All Mathematica objects are either **Atomic**, e.g.

```
Head[133] ☞ Integer
Head[a] ☞ Symbol
```

or (generalized) **Lists** with a **Head** and **Elements**:

```
expr = a + b
FullForm[expr] ☞ Plus[a, b]
Head[expr] ☞ Plus
expr[[0]] ☞ Plus        — same as Head[expr]
expr[[1]] ☞ a
expr[[2]] ☞ b
```

List-oriented Programming

Using Mathematica's list-oriented commands is almost always of advantage in both speed and elegance.

Consider:

```
array = Table[Random[], {10^7}];

test1 := Block[ {sum = 0},
  Do[ sum += array[[i]], {i, Length[array]} ];
  sum ]

test2 := Apply[Plus, array]
```

Here are the timings:

```
Timing[test1][[1]] ☞ 31.63 Second
Timing[test2][[1]] ☞ 3.04 Second
```

Map, Apply, and Pure Functions

Map applies a function to all elements of a list:

```
Map[f, {a, b, c}] ☞ {f[a], f[b], f[c]}
f /@ {a, b, c} ☞ {f[a], f[b], f[c]}
```
— short form

Apply exchanges the head of a list:

```
Apply[Plus, {a, b, c}] ☞ a + b + c
Plus @@ {a, b, c} ☞ a + b + c
```
— short form

Pure Functions are a concept from formal logic. A pure function is defined ‘on the fly’:

```
(# + 1)& /@ {4, 8} ☞ {5, 9}
```

The `#` (same as `#1`) represents the first argument, and the `&` defines everything to its left as the pure function.

List Operations

Flatten removes all sub-lists:

```
Flatten[f[x, f[y], f[f[z]]]] ☞ f[x, y, z]
```

Sort and **Union** sort a list. **Union** also removes duplicates:

```
Sort[{3, 10, 1, 8}] ☞ {1, 3, 8, 10}
Union[{c, c, a, b, a}] ☞ {a, b, c}
```

Prepend and **Append** add elements at the front or back:

```
Prepend[r[a, b], c] ☞ r[c, a, b]
Append[r[a, b], c] ☞ r[a, b, c]
```

Insert and **Delete** insert and delete elements:

```
Insert[h[a, b, c], x, {2}] ☞ h[a, x, b, c]
Delete[h[a, b, c], {2}] ☞ h[a, c]
```

Patterns

One of the most useful features is Pattern Matching:

`_`	— matches one object
`__`	— matches one or more objects
`___`	— matches zero or more objects
`x_`	— named pattern (for use on the r.h.s.)
`x_h`	— pattern with head `h`
`x_:1`	— default value
`x_?NumberQ`	— conditional pattern
`x_ /; x > 0`	— conditional pattern

Patterns take function overloading to the limit, i.e. functions behave differently depending on *details* of their arguments:

```
Attributes[Pair] = {Orderless}
Pair[p_Plus, j_] := Pair[#, j]& /@ p
Pair[n_?NumberQ i_, j_] := n Pair[i, j]
```

Mathematical Functions

Mathematica is equipped with a large set of mathematical functions, both for symbolic and numeric operations.

Some examples:

`Integrate[x^2, {x,3,5}]`	— integral
`D[f[x], x]`	— derivative
`Sum[i, {i,50}]`	— sum
`Series[Sin[x], {x,1,5}]`	— series expansion
`Simplify[(x^2 - x y)/x]`	— simplify
`Together[1/x + 1/y]`	— put on common denominator
`Inverse[mat]`	— matrix inverse
`Eigenvalues[mat]`	— eigenvalues
`PolyLog[2, 1/3]`	— polylogarithm
`LegendreP[11, x]`	— Legendre polynomial
`Gamma[.567]`	— Gamma function

Graphics

Mathematica has formidable graphics capabilities:

```
Plot[ArcTan[x], {x, 0, 2.5}]
ParametricPlot[{Sin[x], 2 Cos[x]}, {x, 0, 2 Pi}]
Plot3D[1/(x^2 + y^2), {x, -1, 1}, {y, -1, 1}]
ContourPlot[x y, {x, 0, 10}, {y, 0, 10}]
```

Output can be saved to a file with `Export`**:**

```
plot = Plot[Abs[Zeta[1/2 + x I]], {x, 0, 50}]
Export["zeta.eps", plot, "EPS"]
```

Mathematica Summary

- Mathematica makes it wonderfully easy, even for fairly unskilled users, to manipulate expressions.
- Most functions you will ever need are already built in.
- When using its capabilities (in particular list-oriented programming and pattern matching) right, Mathematica can be very efficient.
  Wrong: `FullSimplify[veryLongExpression]`.
- Mathematica is a general-purpose system, i.e. convenient to use, but not ideal for everything.
  For example, in numerical functions, Mathematica usually selects the algorithm automatically, which may or may not be a good thing.

FORM

FORM Essentials

- A FORM program is divided into **Modules.**
  Simplification happens only at the end of a module.
- FORM is **strongly typed** –
  all variables have to be declared:
  Symbols, Vectors, Indices, (N)Tensors, (C)Functions.
- FORM works **on one term at a time:**
  Can do "`Expand[(a + b)^2]`" (**local** operation) but
  not "`Factor[a^2 + 2 a b + b^2]`" (**global** operation).
- FORM is mainly strong on **polynomial expressions.**
- FORM program + documentation + course available from
  http://nikhef.nl/∼form.

A Simple Example in FORM

```
Symbols a, b, c, d;
Local expr = (a + b)^2;
id b = c - d;
print;
.end
```

Running this program gives:

```
FORM by J.Vermaseren,version 3.2(Mar 1 2007) Run at: Tue May  8 10:14:12 2007
    Symbols a, b, c, d;
    Local expr = (a + b)^2;
    id b = c - d;
    print;
    .end

Time =       0.00 sec    Generated terms =          6
             expr        Terms in output =          6
                         Bytes used      =        104

   expr =
      d^2 - 2*c*d + c^2 - 2*a*d + 2*a*c + a^2;

  0.00 sec out of 0.00 sec
```

Module Structure

A FORM program consists of Modules. A Module is terminated by a "dot" statement (`.sort`, `.store`, `.end`, ...)

- Generation Phase ("normal" statements)
  During the execution of "normal" statement terms are only generated. This is a purely local operation – only one term at a time needs to be looked at.
- Sorting Phase ("dot" statements):
  At the end of the module all terms are inspected and similar terms collected. This is the only 'global' operation which requires FORM to look at all terms 'simultaneously.'

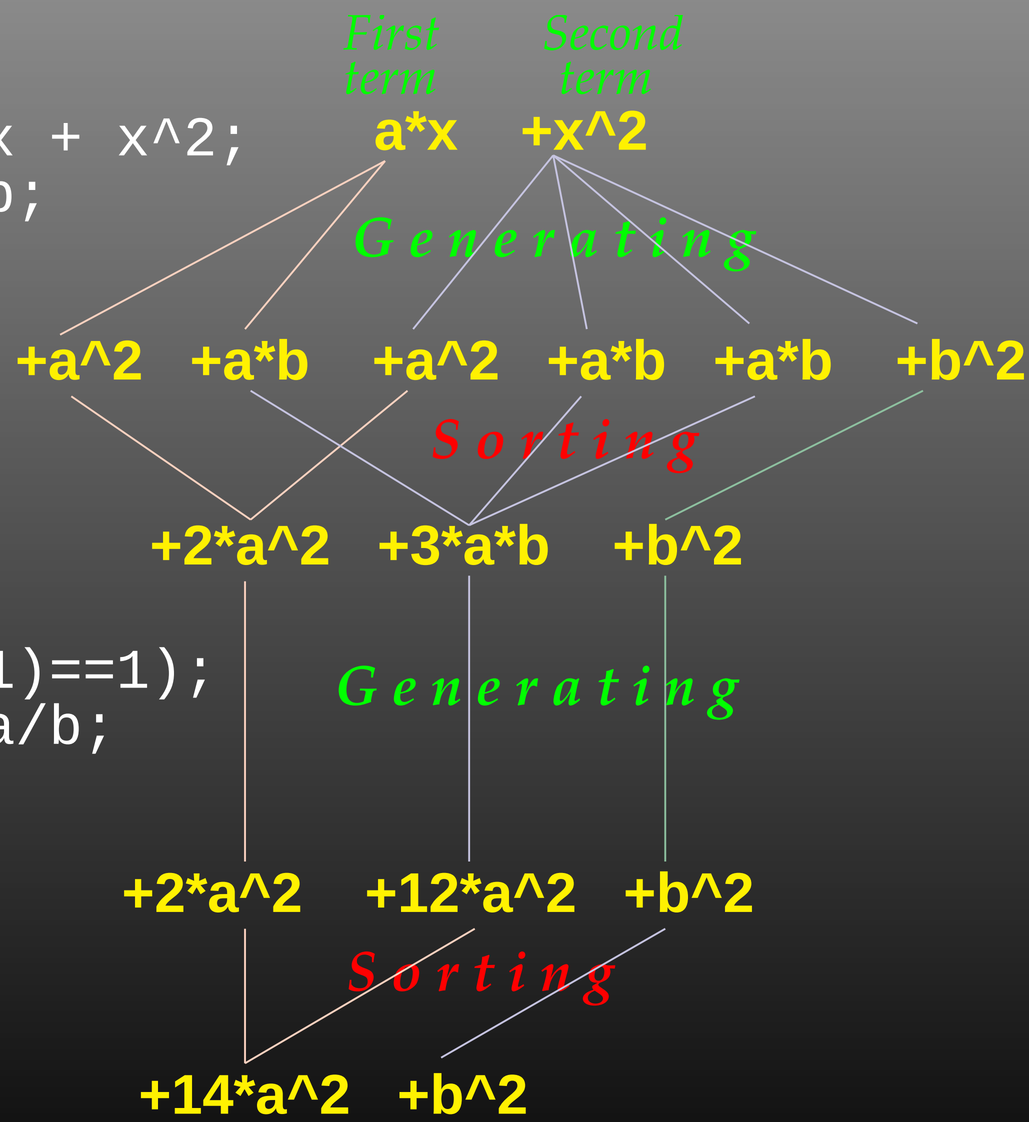
Sorting and Generating

First module

```
l expr = a*x + x^2;
id x = a + b;

.sort
```

Second module

```
if(count(b,1)==1);
multiply 4*a/b;
endif;

print;
.end
```



Id-Statement

The central statement in FORM is the `id`-Statement:

```
a^3*b^2*c
id a*b = d;     ☞  a*c*d^2        — multiple match
once a*b = d;   ☞  a^2*b*c*d      — single match
only a*b = d;   ☞  a^3*b^2*c      — no exact match possible
```

`id` **does not, by default, match negative powers:**

```
x + 1/x
id x = y;        ☞  x^-1 + y
id x^n? = y^n;   ☞  y^-1 + y      — wildcard exponent
```

Patterns

Patterns are possible (but not as powerful as in Mathematica):

```
f(a, b, c) + f(1, 2, 3)
id f(a, b, c) = 1; ☞ 1 + f(1, 2, 3)
```
— explicit match

```
id f(a?, b?, c?) = 1; ☞ 2
```
— wildcard match

```
id f(?a) = g(?a); ☞ g(a, b, c) + g(1, 2, 3)
```
— group-wildcard match

```
id f(a?int_, ?a) = a; ☞ 1 + f(a, b, c)
```
— constrained wildcard

```
id f(a?{a,b}, ?a) = a; ☞ a + f(1, 2, 3)
```
— alternatives

Preprocessor

FORM has a Preprocessor which operates before the compiler.

Many constructs are familiar from C, but the FORM preprocessor can do more:

- `#define`, `#undefine`, `#redefine`,
- `#if{,def,ndef}` … `#else` … `#endif`,
- `#switch` … `#endswitch`,
- `#procedure` … `#endprocedure`, `#call`,
- `#do` … `#enddo`,
- `#write`, `#message`, `#system`.

The preprocessor works across modules, e.g. a `do`-loop can contain a `.sort` statement.

Special Commands for High-Energy Physics

- **Gamma matrices:** `g_`, `g5_`, `g6_`, `g7_`.
- **Fermion traces:** `trace4`, `tracen`, `chisholm`.
- **Levi-Civita tensors:** `e_`, `contract`.
- **Index properties:** `{,anti,cycle}symmetrize`.
- **Dummy indices:** `sum`, `replaceloop`.
  (e.g. $\sum_i a_i b_i + \sum_j a_j b_j = 2 \sum_i a_i b_i$)

FORM Summary

- FORM is a freely available Computer Algebra System with (some) specialization on High Energy Physics.
- Programming in FORM takes more 'getting used to' than in Mathematica. Also, FORM has no GUI or other programming aids.
- FORM programs are module-oriented with global (= costly) operations occurring only at the end of module. A strategic choice of these points optimizes performance.
- FORM is typically much faster than Mathematica on polynomial expressions and can handle in particular huge (GB) expressions.

Examples

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, practically impossible in FORM.
- **Simplification of Colour Structures**
  Different approaches.
- **Calculation of a Fermion Trace**
  Built-in in FORM, complicated in Mathematica.

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`
(or better: `Simplify[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.**

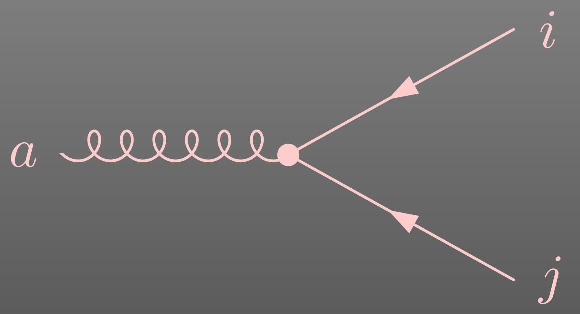
Abbreviationing in Mathematica

```
$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[]
```

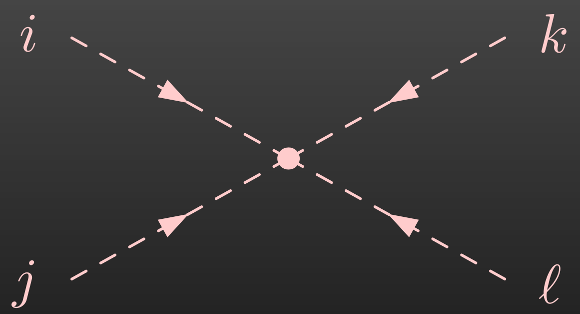
Colour Structures

In Feynman diagrams four type of Colour structures appear:

Natural Representation

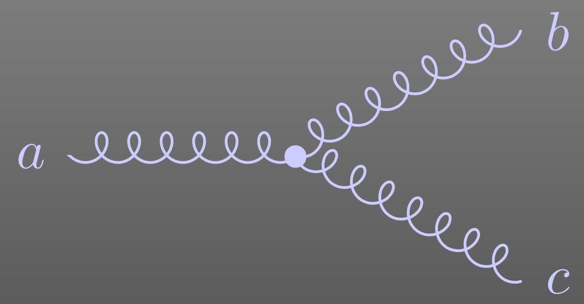


$\sim T^a_{ij} = \texttt{SUNT}[a,i,j]$

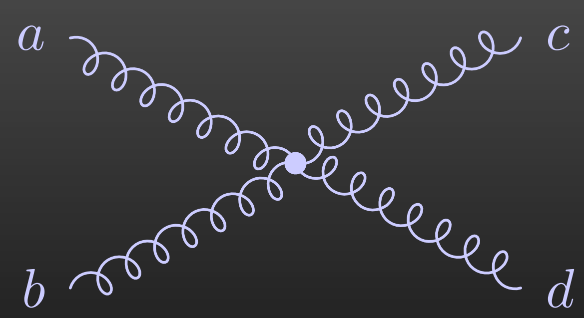


$\sim T^a_{ij} T^a_{k\ell} = \texttt{SUNTSum}[i,j,k,\ell]$

Adjoint Representation



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



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

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`:

- `SUNT[`i`,`j`]` $= \delta_{ij}$
- `SUNT[`a`,`b`, . . . ,`i`,`j`]` $= (T^a T^b \cdots)_{ij}$
- `SUNT[`a`,`b`, . . . ,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
```