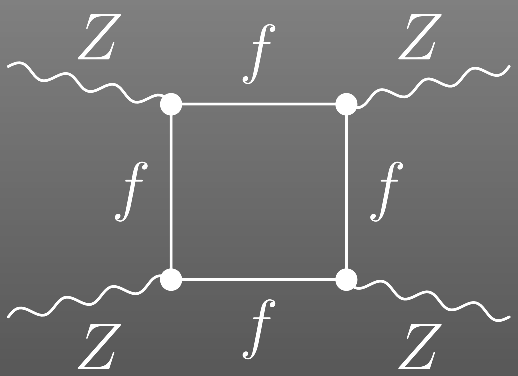
FORM – a user's memoirs

Thomas Hahn

Max-Planck-Institut für Physik
München

The Making of FormCalc

Attempt to compute boxes like this with FeynCalc failed:



Problem was an inefficient implementation of the fermion trace.

Today I might have used

```
Trace4[mu_, g___] := Block[{Trace4, s = -1},
  Plus@@ MapIndexed[((s = -s) Pair[mu, #1] Drop[Trace4[g], #2])&, {g}]];
Trace4[] = 4
```

but in 1995 computation of a trace was a big mystery (to me) and then there was FORM . . .

NoDoorCalc?

The name FormCalc actually derived from the fact that the program was doing the same as FeynCalc, but using FORM (more or less).

After the first release I received mail from Schörghuber, who had registered FormCalc as a trademark (to compute some kind of form factors for special doors). I was able to convince them that there was no chance of mistaking one for the other and they kindly allowed me to continue using the name.

Nowadays, the only FormCalc remaining is the one using FORM.

Factor 2

The interface between Mathematica and FORM had always been a weak point, i.e. it was not difficult to write down Mathematica expressions that FORM would hiccup over, e.g. `f[a][b]` (complex head).

Hence, in 2002, I sat down and tried very hard to write a Mathematica equivalent that was approximately on-par with the FORM code. Trying to generate the fewest numbers terms from the beginning etc.

Result: Mathematica code still factor 2 slower than FORM.

Quantum Leaps in FORM

Abbreviationing: replace subexpressions by symbols.

FORM 2:

(nothing, i.e. keep subexpr in FORM)

Quantum Leaps in FORM

Abbreviationing: replace subexpressions by symbols.

FORM 3:

```
#procedure ToMma(expr)
#toexternal "%E,", `expr'
#endprocedure

#procedure FromMma(expr)
G `expr' =
#fromexternal
  ;
#endprocedure
```

Quantum Leaps in FORM

Abbreviationing: replace subexpressions by symbols.

FORM 4:

```
argument mulM;
toPolynomial;
endargument;
```

Factorization

Ah yes, and there was factorization in FORM 4, too.

Before that:

```
L expr = (a + b)*(c + d);
print;
.sort
     expr = a*c + a*d + b*c + b*d;

bracket a, b;
print;
.sort
     expr = + a * ( c + d )
            + b * ( c + d );

collect f;
bracket f;
print;
.end
     expr = + f(c + d) * ( a + b );
```

FORM Executables

Availability of FORM executables was an issue for some time:

Early days:

- User's responsibility to install FORM.
- FORM 2 commercial, fallback to FORM 1 was possible.

With FORM 3:

- Initially still user responsibility.
- After too many 'bug reports' included binaries.

With FORM 4:

- Sources included (to fix version).
- Pre-compiled executables included.
- Build script included (rarely used).

FORM Syntax

Only language that admits `[x+1]` as variable name!

```
* variables appearing in the CalcFeynAmp input and output
cf SumOver, PowerOf, Den, A0, IGram, List;
...

* variables that make it into Mma but don't appear in the output
cf powM, sunM, intM, tensM, extM, paveM, cutM, numM, qfM, qcM;
...

* patterns
i [a], [b], [c], [d];
...
```

For, if any of the patterns makes it into Mathematica, this is bound to give a failure notice.

FORM Syntax

Canonical ordering can be chosen:

```
v k2, k1, k3, k4;
L eps = e_(k1, k2, k3, k4);
.sort

eps = -e_(k2, k1, k3, k4);
```

FORM Syntax

Lack of scoping of local variables in `#procedure`.

Maybe do like Mathematica:

```
In[1]:= Module[{a}, Print[a]]

Out[1]= a$1099
```

Things I spent time on

On reflection, there are not many things that are ultimately impossible in FORM, but for quite a few one needs to look at the problem in a new way and devise an alternative approach, as the straightforward one cannot be written down directly.

FORM needs some 'getting used to.'

I suspect this is where (some of) its power derives from: forcing the user to re-organize problems in a way more tractable to the workings of FORM/the computer in general.

Still, in terms of overall development time it could be worthwhile to come closer to the naive version.

Things I spent time on

Extracting the n-th argument of a function.

If n is known at compile time, workaround with $\ldots$ operator.

If n is known only at run time?

```
L test = int(Den(...), Den(...), ...);

$n = 0;
repeat;
  $n = $n + 1;
  once int(Den(?a), ?b) = Den($n,?a)*int(?b);
endrepeat;
```

How complicated could something like `arg_(n,?a)` be?

Things I spent time on

Getting the fewest number of terms.
- This is in general a hugely complex endeavor.

FormCalc currently brute-forces momentum conservation (by default):

```
#define k1 "-k2 + k3 + k4"
#define k2 "-k1 + k3 + k4"
#define k3 "k1 + k2 - k4"
#define k4 "k1 + k2 - k3"
#define MomRange "1,2,3,4"

#do i = {`MomRange'}
id k`i' = `k`i'';
#call ChainOrder
#enddo
```

Reduce Number of Terms

On a smaller scale: apply e.g. momentum conservation,

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

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

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

Something like `tryreplace` but for number-of-terms would be very helpful.

Time to Adoption

Another one of these Quantum Leaps (I have not used so far):

```
format O4;
```

The reason is that I need a certain structure of the result to correctly identify over which terms a `SumOver` extends.

Add possibility to exclude certain functions from optimization.

Mathematica vs. FORM – that emblematic comparison

Mathematica

- Much built-in knowledge,
- ‘Big and slow’ (esp. 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).

FORM + Other Tools

FORM has a lot unique features not found in other programs. Can we make it easier connecting FORM with other tools?

FORM already has pipes for external communication: `#external`, `#toexternal`, `#fromexternal`. But not straightforward to work with.

FormRun is an attempt to feed 'arbitrary' Mathematica expressions into FORM:

`FormRun[expr, (decl), (cmd)]`

Still, the module structure of FORM input makes it somewhat more complicated than just writing an expression out and getting it back.