Introduction to FORM

Ben Ruijl

July 17 - 19, 2023

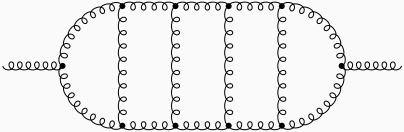
Ruijl Research

Computer algebra and particle physics

- Developments in particle physics have gone hand in hand with those in computer algebra
- In 1963 Veltman created SCHOONSCHIP
- It computed ``a monstrous expression involving in the order of 50 000 terms in intermediate stages'' and had to be stored on tape
- In 1984 Vermaseren started work on FORM
- As computing power and algorithms improved, so did the ambition for precision
- In the 1960s an order of magnitude agreement with experiment was good
- Nowadays, the goal is to achieve $< 1\%$ error

Computational blow-up

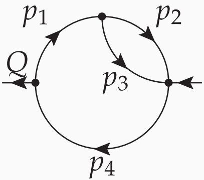
- One of 6000 diagrams of the five-loop gluon propagator:



- Appyling Feynman rules yields 12 029 521 terms!
- Can be reduced to 23 master integrals with algebraic identities
- This reduction requires a terabyte of disk space and is time-consuming
- Blow-up of rational coefficients

- An integral F can be rewritten in terms of simpler ones using Integration by Parts (IBP) identities:

$$\frac{\partial}{\partial p_i} p_j \circ F = 0$$



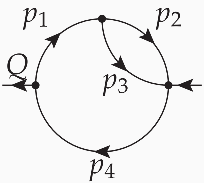
$$= \int d^D p_1 d^D p_2 \frac{1}{(p_1^2)^{n_1}(p_2^2)^{n_2}(p_1 - p_2)^{2n_3}(Q + p_1)^{2n_4}}$$

to which we apply the IBP identity $Q^\mu \frac{\partial}{\partial p_2^\mu}$:

$$\left[-2n_2 \frac{Q \cdot p_2}{p_2^2} + 2n_3 \frac{Q \cdot p_2}{p_3^2} - n_3 \frac{p_4^2}{p_3^2} + n_3 \frac{p_1^2}{p_3^2} + n_3 \frac{Q^2}{p_3^2}\right] \circ F = 0$$

- An integral F can be rewritten in terms of simpler ones using Integration by Parts (IBP) identities:

$$\frac{\partial}{\partial p_i} p_j \circ F = 0$$



$$= \int d^D p_1 d^D p_2 \frac{1}{(p_1^2)^{n_1}(p_2^2)^{n_2}(p_1-p_2)^{2n_3}(Q+p_1)^{2n_4}}$$

to which we apply the IBP identity $Q^\mu \frac{\partial}{\partial p_2^\mu}$:

$$\left[-2n_2\frac{Q\cdot p_2}{p_2^2} + 2n_3\frac{Q\cdot p_2}{p_3^2} - n_3\frac{p_4^2}{p_3^2} + n_3\frac{p_1^2}{p_3^2} + n_3\frac{Q^2}{p_3^2}\right]\circ F = 0$$

Integration by Parts identities

$$\left[-2n_2\frac{Q\cdot p_2}{p_2^2}+2n_3\frac{Q\cdot p_2}{p_3^2}-n_3\frac{p_4^2}{p_3^2}+n_3\frac{p_1^2}{p_3^2}+n_3\frac{Q^2}{p_3^2}\right]\circ F=0$$

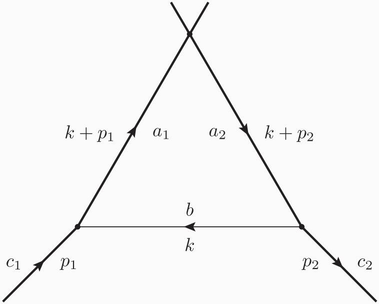
Express IBP identity in basis $\{p_1^2, p_2^2, p_3^2, p_4^2, 2Q\cdot p_2\}$ with raising operators:

$$\left[-n_2\mathsf{N}_5^+\mathsf{N}_2^+ + n_3\left(\mathsf{N}_5^+\mathsf{N}_3^+ - \mathsf{N}_4^-\mathsf{N}_3^+ + \mathsf{N}_1^-\mathsf{N}_3^+ + Q^2\mathsf{N}_3^+\right)\right]F=0$$

- If we can combine rules such that every term has a N_i^-, we can repeat the rule and remove a propagator

Example: the triangle rule

$$\int d^D k \frac{k^{\mu_1} \dots k^{\mu_N}}{\left[(k+p_1)^2+m_1^2\right]^{a_1}\left[(k+p_2)^2+m_2^2\right]^{a_2}(k^2)^b(p_1^2+m_1^2)^{c_1}(p_2^2+m_2^2)^{c_2}}$$

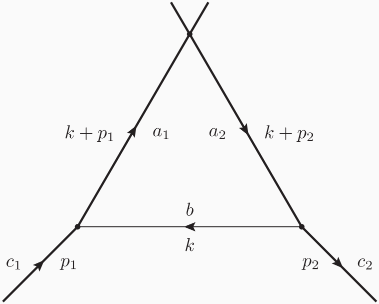


Apply $\frac{\partial}{\partial k_\mu} k_\mu \circ F = 0$:

$$1 = \frac{1}{D+N-a_1-a_2-2b}\Big[a_1 \mathsf{A}_1^+(\mathsf{B}^- - \mathsf{C}_1^-) + a_2 \mathsf{A}_2^+(\mathsf{B}^- - \mathsf{C}_2^-)\Big]$$

Example: the triangle rule

$$\int d^D k \frac{k^{\mu_1}\dots k^{\mu_N}}{\left[(k+p_1)^2+m_1^2\right]^{a_1}\left[(k+p_2)^2+m_2^2\right]^{a_2}(k^2)^b(p_1^2+m_1^2)^{c_1}(p_2^2+m_2^2)^{c_2}}$$



Apply $\frac{\partial}{\partial k_\mu} k_\mu \circ F = 0$:

$$1 = \frac{1}{D+N-a_1-a_2-2b}\Big[a_1 \mathsf{A}_1^+(\mathsf{B}^- - \mathsf{C}_1^-) + a_2 \mathsf{A}_2^+(\mathsf{B}^- - \mathsf{C}_2^-)\Big]$$

Rules are not always easy

```
id Z(n1?pos_,n2?pos_,n3?pos_,n4?pos_,n5?pos_,n6?pos_,n7?pos_,
    n8?pos_,n9?pos_,n10?neg0_,n11?neg0_,n12?neg0_,n13?neg0_,n14?neg_)
       = -rat(1,-2*ep-2*n1-n3-n6-n12-n14+4)*(
+Z(-1+n1,-1+n2,n3,n4,1+n5,n6,n7,n8,n9,n10,n11,n12,n13,1+n14)*rat(-n5,1)
+Z(-1+n1,1+n2,n3,n4,-1+n5,n6,n7,n8,n9,n10,n11,n12,n13,1+n14)*rat(n2,1)
+Z(-1+n1,1+n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,n12,n13,1+n14)*rat(-n2,1)
+Z(-1+n1,n2,1+n3,n4,n5,n6,n7,n8,-1+n9,n10,n11,n12,n13,1+n14)*rat(-n3,1)
+Z(-1+n1,n2,n3,n4,-1+n5,n6,n7,n8,n9,n10,n11,1+n12,n13,1+n14)*rat(-n12,1)
+Z(-1+n1,n2,n3,n4,1+n5,n6,n7,-1+n8,n9,n10,n11,n12,n13,1+n14)*rat(n5,1)
+Z(-1+n1,n2,n3,n4,n5,n6,-1+n7,n8,n9,n10,n11,n12,1+n13,1+n14)*rat(-n13,1)
+Z(-1+n1,n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,1+n12,n13,1+n14)*rat(2*n12,1)
+Z(-1+n1,n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,n12,1+n13,1+n14)*rat(n13,1)
+Z(-1+n1,n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,n12,n13,2+n14)*rat(2*n14+2,1)
+Z(-1+n1,n2,n3,n4,n5,n6,n7,n8,-1+n9,n10,n11,n12,1+n13,1+n14)*rat(n13,1)
+Z(-1+n1,n2,n3,n4,n5,n6,n7,n8,n9,1+n10,-1+n11,n12,n13,1+n14)*rat(-n10,1)
+Z(-1+n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,1+n12,n13,n14)*rat(-n12,1)
+Z(-1+n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,n12,n13,1+n14)*rat(-n2+n5,1)
+Z(n1,-1+n2,-1+n3,n4,n5,n6,n7,n8,n9,n10,n11,1+n12,n13,1+n14)*rat(n12,1)
+Z(n1,-1+n2,-1+n3,n4,n5,n6,n7,n8,n9,n10,n11,n12,n13,2+n14)*rat(1+n14,1)
+Z(n1,-1+n2,1+n3,n4,n5,n6,n7,n8,-1+n9,n10,n11,n12,n13,1+n14)*rat(n3,1)
```

```
+Z(n1,-1+n2,n3,-1+n4,n5,n6,n7,n8,n9,n10,1+n11,n12,n13,1+n14)*rat(n11,1)
+Z(n1,-1+n2,n3,n4,-1+n5,n6,n7,n8,n9,n10,n11,1+n12,n13,1+n14)*rat(n12,1)
+Z(n1,-1+n2,n3,n4,n5,-1+n6,n7,n8,n9,n10,n11,n12,n13,2+n14)*rat(-n14-1,1)
+Z(n1,-1+n2,n3,n4,n5,n6,-1+n7,1+n8,n9,n10,n11,n12,n13,1+n14)*rat(2*n8,1)
+Z(n1,-1+n2,n3,n4,n5,n6,-1+n7,n8,n9,n10,1+n11,n12,n13,1+n14)*rat(-n11,1)
+Z(n1,-1+n2,n3,n4,n5,n6,-1+n7,n8,n9,n10,n11,n12,1+n13,1+n14)*rat(-n13,1)
+Z(n1,-1+n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,1+n12,n13,1+n14)*rat(-2*n12,1)
+Z(n1,-1+n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,n12,1+n13,1+n14)*rat(n13,1)
+Z(n1,-1+n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,n12,n13,2+n14)*rat(-2*n14-2,1)
+Z(n1,-1+n2,n3,n4,n5,n6,n7,1+n8,-1+n9,n10,n11,n12,n13,1+n14)*rat(-n8,1)
+Z(n1,-1+n2,n3,n4,n5,n6,n7,n8,-1+n9,n10,n11,1+n12,n13,1+n14)*rat(-2*n12,1)
+Z(n1,-1+n2,n3,n4,n5,n6,n7,n8,n9,1+n10,-1+n11,n12,n13,1+n14)*rat(n10,1)
+Z(n1,-1+n2,n3,n4,n5,n6,n7,n8,n9,1+n10,n11,n12,n13,1+n14)*rat(-2*n10,1)
+Z(n1,-1+n2,n3,n4,n5,n6,n7,n8,n9,n10,1+n11,n12,n13,1+n14)*rat(-n11,1)
+Z(n1,-1+n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,1+n12,n13,n14)*rat(-n12,1)
+Z(n1,1+n2,n3,n4,-1+n5,n6,n7,n8,n9,-1+n10,n11,n12,n13,1+n14)*rat(-n2,1)
+Z(n1,1+n2,n3,n4,-1+n5,n6,n7,n8,n9,n10,n11,n12,n13,1+n14)*rat(-n2,1)
+Z(n1,1+n2,n3,n4,n5,n6,n7,-1+n8,n9,-1+n10,n11,n12,n13,1+n14)*rat(2*n2,1)
+Z(n1,1+n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,n12,n13,1+n14)*rat(n2,1)
+Z(n1,n2,-1+n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,1+n12,n13,1+n14)*rat(-n12,1)
+Z(n1,n2,-1+n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,n12,n13,2+n14)*rat(-n14-1,1)
+Z(n1,n2,1+n3,n4,n5,n6,n7,n8,-1+n9,-1+n10,n11,n12,n13,1+n14)*rat(n3,1)
```

```
+Z(n1,n2,n3,n4,-1+n5,n6,-1+n7,n8,n9,n10,n11,n12,1+n13,1+n14)*rat(n13,1)
+Z(n1,n2,n3,n4,-1+n5,n6,n7,n8,-1+n9,n10,n11,n12,1+n13,1+n14)*rat(-n13,1)
+Z(n1,n2,n3,n4,-1+n5,n6,n7,n8,-1+n9,n10,n11,n12,n13,2+n14)*rat(1+n14,1)
+Z(n1,n2,n3,n4,-1+n5,n6,n7,n8,n9,-1+n10,n11,1+n12,n13,1+n14)*rat(n12,1)
+Z(n1,n2,n3,n4,-1+n5,n6,n7,n8,n9,n10,-1+n11,n12,1+n13,1+n14)*rat(n13,1)
+Z(n1,n2,n3,n4,-1+n5,n6,n7,n8,n9,n10,n11,1+n12,n13,1+n14)*rat(n12,1)
+Z(n1,n2,n3,n4,-1+n5,n6,n7,n8,n9,n10,n11,n12,1+n13,1+n14)*rat(n13,1)
+Z(n1,n2,n3,n4,-1+n5,n6,n7,n8,n9,n10,n11,n12,n13,1+n14)*rat(-n2+n8-n13,1)
+Z(n1,n2,n3,n4,n5,-1+n6,n7,-1+n8,n9,n10,n11,n12,n13,2+n14)*rat(1+n14,1)
+Z(n1,n2,n3,n4,n5,n6,-1+n7,-1+n8,n9,n10,1+n11,n12,n13,1+n14)*rat(n11,1)
+Z(n1,n2,n3,n4,n5,n6,-1+n7,-1+n8,n9,n10,n11,n12,1+n13,1+n14)*rat(-2*n13,1)
+Z(n1,n2,n3,n4,n5,n6,-1+n7,n8,n9,-1+n10,1+n11,n12,n13,1+n14)*rat(n11,1)
+Z(n1,n2,n3,n4,n5,n6,-1+n7,n8,n9,n10,n11,-1+n12,n13,2+n14)*rat(1+n14,1)
+Z(n1,n2,n3,n4,n5,n6,-1+n7,n8,n9,n10,n11,n12,1+n13,1+n14)*rat(n13,1)
+Z(n1,n2,n3,n4,n5,n6,-1+n7,n8,n9,n10,n11,n12,n13,1+n14)*rat(-2*ep-2*n4-1,1)
+Z(n1,n2,n3,n4,n5,n6,1+n7,-1+n8,n9,n10,n11,n12,n13,1+n14)*rat(-n7,1)
+Z(n1,n2,n3,n4,n5,n6,n7,-1+n8,-1+n9,n10,n11,n12,1+n13,1+n14)*rat(n13,1)
+Z(n1,n2,n3,n4,n5,n6,n7,-1+n8,-1+n9,n10,n11,n12,n13,2+n14)*rat(-2*n14-2,1)
+Z(n1,n2,n3,n4,n5,n6,n7,-1+n8,n9,-1+n10,1+n11,n12,n13,1+n14)*rat(-n11,1)
+Z(n1,n2,n3,n4,n5,n6,n7,-1+n8,n9,-1+n10,n11,1+n12,n13,1+n14)*rat(-2*n12,1)
+Z(n1,n2,n3,n4,n5,n6,n7,-1+n8,n9,-1+n10,n11,n12,n13,2+n14)*rat(-2*n14-2,1)
+Z(n1,n2,n3,n4,n5,n6,n7,-1+n8,n9,1+n10,n11,n12,n13,1+n14)*rat(2*n10,1)
```

```
+Z(n1,n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,-1+n11,n12,1+n13,1+n14)*rat(-2*n13,1)
+Z(n1,n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,1+n11,n12,n13,1+n14)*rat(n11,1)
+Z(n1,n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,-1+n12,n13,2+n14)*rat(-n14-1,1)
+Z(n1,n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,1+n12,n13,1+n14)*rat(-2*n12,1)
+Z(n1,n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,n12,1+n13,1+n14)*rat(-3*n13,1)
+Z(n1,n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,n12,n13,1+n14)*rat(10*ep+2*n1,1)
+Z(n1,n2,n3,n4,n5,n6,n7,-1+n8,n9,n10,n11,n12,n13,2+n14)*rat(-2*n14-2,1)
+Z(n1,n2,n3,n4,n5,n6,n7,n8,-1+n9,-1+n10,n11,1+n12,n13,1+n14)*rat(-n12,1)
+Z(n1,n2,n3,n4,n5,n6,n7,n8,-1+n9,-1+n10,n11,n12,1+n13,1+n14)*rat(-n13,1)
+Z(n1,n2,n3,n4,n5,n6,n7,n8,-1+n9,n10,-1+n11,n12,n13,2+n14)*rat(-n14-1,1)
+Z(n1,n2,n3,n4,n5,n6,n7,n8,-1+n9,n10,n11,n12,1+n13,1+n14)*rat(-n13,1)
+Z(n1,n2,n3,n4,n5,n6,n7,n8,-1+n9,n10,n11,n12,n13,1+n14)*rat(-4*ep-2*n1-n3,1)
+Z(n1,n2,n3,n4,n5,n6,n7,n8,n9,-1+n10,n11,n12,n13,1+n14)*rat(-n5+1,1)
+Z(n1,n2,n3,n4,n5,n6,n7,n8,n9,1+n10,-1+n11,n12,n13,1+n14)*rat(n10,1)
+Z(n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,-1+n11,n12,n13,1+n14)*rat(2*ep+n5+2*n8
    +n9+n10+n11-5,1)
+Z(n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,1+n12,n13,n14)*rat(n12,1)
+Z(n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,n12,-1+n13,1+n14)*rat(-2*ep-n5-2*n8
    -n9-n11-n14+3,1)
+Z(n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,n12,n13,1+n14)*rat(2*ep+n2+n7+
    2*n8+n9+n11+n14-4,1)
);
```

Computational bottlenecks

- Often IBPs cannot be solved parametrically
- Solve the system through a brute force Gaussian elimination
- Simplification of coefficients with many masses and ϵ is a bottleneck
- Millions of terms that do not fit in memory
- Swapping kills performance

Requires special tools

Mathematica, Maple, etc. cannot process this workload.

Polynomial GCDs

$$\gcd(a,b) = \begin{cases} a, & b = 0 \\ \gcd(b, a\%b) & \text{otherwise} \end{cases}$$

$$a = x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5$$
$$b = 3x^6 + 5x^4 - 4x^2 - 9x + 21$$

Polynomial GCDs

$$\gcd(a, b) = \begin{cases} a, & b = 0 \\ \gcd(b, a\%b) & \text{otherwise} \end{cases}$$

$$a = x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5$$
$$b = 3x^6 + 5x^4 - 4x^2 - 9x + 21$$

$$R_1 = -\frac{5}{9}x^2 + \frac{1}{9}x^2 - \frac{1}{3}$$
$$R_2 = -\frac{117}{25}x^2 - 9x + \frac{411}{25}$$
$$R_3 = -\frac{102500}{6591} + \frac{233150}{19773}x$$
$$R_4 = -\frac{1288744821}{543589225}$$

- From $\mathbb{Z}$ to $\mathbb{Q}$
- Rapid coefficient growth

Polynomial GCDs

An attempt with pseudo-remainders:

$$
\begin{aligned}
R_1 &= -15x^4 + 3x^2 - 9 \\
R_2 &= 15795x^2 + 30375x - 5953 \\
R_3 &= 1254542875143750x - 1654608338437500 \\
R_4 &= 12593338795500743100931141992187500
\end{aligned}
$$

- Coefficients still in $\mathbb{Z}$, but are huge
- Modular algorithm with reconstruction is needed
- Multivariate case is much harder and actively researched

- FORM [Vermaseren '89] is a very popular symbolic manipulation toolkit in the field
- It processes each term one by one, as not to run out of memory
- Processed terms are stored into memory at first
- When memory is full, write and read from disk
- Can handle large scale computations

Obtaining FORM

- Download FORM 4.3.1 from:
  `https://github.com/vermaseren/form/releases`
- Tutorial: `https://www.nikhef.nl/~form/maindir/documentation/tutorial/online/online.html`
- Reference manual: `https://www.nikhef.nl/~form/maindir/documentation/reference/online/online.html`
- Visual Studio Code Syntax highlighting extension
- FORM Cookbook:
  `https://github.com/vermaseren/form/wiki/FORM-Cookbook`
- Compile:

```
./configure
make -j4
make install
```

Example program

Save the following as `prog1.frm`:

```
Symbols a,b;
Local F = (a+b)^2;
Print;
.end
```

and run

```
form prog1
```

Example program

```
Symbols a,b;  * define symbols
Local F = (a+b)^2;  * define expression
Print;  * print the expression
.end;  * end the program (and sort)
```

```
Time =        0.00 sec    Generated terms =          3
F                         Terms in output =          3
                          Bytes used      =        108

F =
  b^2 + 2*a*b + a^2;
```

Example program

```
Symbols a,b;  * define symbols
Local F = (a+b)^2;  * define expression
Print;  * print the expression
.end;  * end the program (and sort)
```

```
Time =        0.00 sec    Generated terms =          3
F                         Terms in output =          3
                          Bytes used      =        108

F =
  b^2 + 2*a*b + a^2;
```

Operations on terms

FORM can replace patterns in terms with `id`:

```
Symbols a,b,c;
Local F = (a+b)^6;
id a^2*b = c;   * replacement
Print;
.end
```

```
F =
  15*c^2 + 15*b^3*c + b^6 + 20*a*b^2*c
  + 6*a*b^5 + 6*a^3*c + a^6;
```

Operations on terms

FORM can replace patterns in terms with `id`:

```
Symbols a,b,c;
Local F = (a+b)^6;
id a^2*b = c;   * replacement
Print;
.end
```

```
F =
  15*c^2 + 15*b^3*c + b^6 + 20*a*b^2*c
  + 6*a*b^5 + 6*a^3*c + a^6;
```

FORM behaviour

- FORM always expands terms
- FORM processes expressions term by term
- This means that *only 1 term* should fit in memory, the rest can be read/written to disk
- Pro: memory is no limitation
- Con: operations on expressions are more difficult

When confused why certain operations don't exist, imagine that every expression is too big to fit in memory

The following FORM program will run (try it):

```
Auto Symbols x;  * all starting with x is a symbol
Local F = (x1+x2+x3+x4+x5+x6)^100;
.end
```

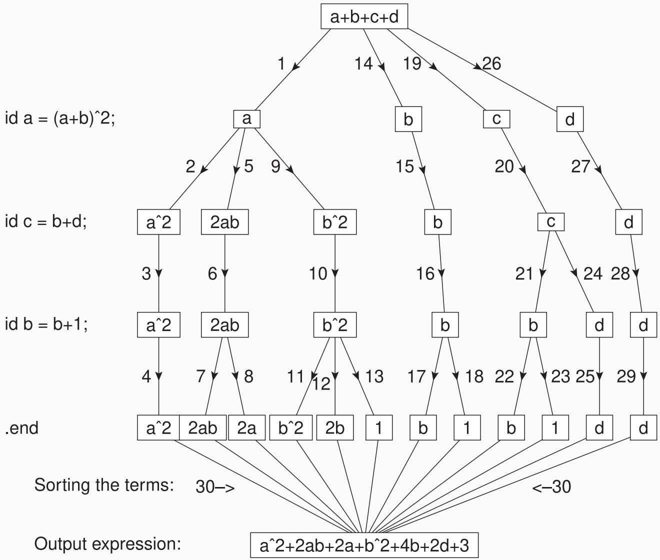
```
Time =         0.45 sec    Generated terms =     100000
                F        1 Terms left      =     100000
                           Bytes used      =    6106364

Time =         1.00 sec    Generated terms =     200000
                F        1 Terms left      =     200000
                           Bytes used      =   12519468
....
```

The following FORM program will run (try it):

```
Auto Symbols x;  * all starting with x is a symbol
Local F = (x1+x2+x3+x4+x5+x6)^100;
.end
```

```
Time =        0.45 sec    Generated terms =     100000
                F        1 Terms left      =     100000
                           Bytes used      =    6106364

Time =        1.00 sec    Generated terms =     200000
                F        1 Terms left      =     200000
                           Bytes used      =   12519468
....
```

Term flow



a+b+c+d
1
14
19
26
id a = (a+b)ˆ2;
a
b
c
d
2
5
9
15
20
27
id c = b+d;
aˆ2
2ab
bˆ2
b
c
d
3
6
10
16
21
24
28
id b = b+1;
aˆ2
2ab
bˆ2
b
b
d
d
4
7
8
11
12
13
17
18
22
23
25
29
.end
aˆ2
2ab
2a
bˆ2
2b
1
b
1
b
1
d
d
Sorting the terms:
30–>
<–30
Output expression:
aˆ2+2ab+2a+bˆ2+4b+2d+3

Sorting

- At the end of a module, terms should be sorted to see if terms will merge
- A module is ended with `.sort` (or `.end`)
- Where to place the sort is up to the user

Sorting: good vs bad

```
Symbols a,b,c,d;
Local F = (a+b+c+1)^6;
id a = -c+d+1;
id b = -d+1;
Print;
.end
```

Generates 924 terms and has 1 in the output...

Sorting: good vs bad

```
Symbols a,b,c,d;
Local F = (a+b+c+1)^6;
id a = -c+d+1;
.sort
id b = -d+1;
.end
```

First sort:

```
Generated terms = 462
Terms in output = 28
```

Second sort:

```
Generated terms = 84
Terms in output = 1
```

Identity statements I

- Patterns for id statements can be any terms
- Use wildcards `var?` to match any object of the same type

```
S x,y;  * short for Symbol
L F = x^2 + y;
id x? = 5;  * match any symbol with x?
Print;
.end
```

yields

```
F = 30
```

Identity statements II

Patterns can be more complicated:

```
S x,y,n;
L F = x^2 + y;
id x?^n? = x^(n + 1);
Print;
.end
```

yields

```
F = y^2 + x^3
```

Identity statements III

- Restrictions can be placed by a set `{1,..}` or a number range `{>5}`
- A statement can be repeated with `repeat`

```
S x,n;
L F = x^10;
repeat id x^n?{>1} = x^(n-1) + x^(n-2);
Print;
.end
```

yields

```
F = 34 + 55*x;
```

Functions

- `Functions` for non-commutative functions
- `CFunctions` for commutative functions

```
S a,b,c;
CF f;  * short for CFunctions
Local F = f(1,2,c);
id f(1,2,b?) = f(1,2,b?+1);
Print;
.end
```

yields:

```
F = f(1,2,c+1);
```

Symmetric functions

```
CF f(s);
L F = f(3,2,1);
Print;
.end
```

yields:

```
F = f(1,2,3);
```

Ranged wildcards

A wildcard starting with a `?` indicates a range:

```
S x;
L F = f(1,2,x,3,4);
id f(?a,x,?b) = f(?b,?a);
Print;
.end
```

yields

```
F = f(3,4,1,2);
```

Applying statements to arguments

id-statements are only applied at ground-level:

```
S x,y;
L F = f(x*y);
id x = 5;  * does not match
argument f;
  id x = 6;
endargument;
Print;
.end
```

yields

```
F = f(6*y)
```

If statements

```
S x,y;
L F = f(2) + f(5);
if (match(f(x?{>4})));
  id f(x?) = f(x + 1);
else;
  id f(x?) = f(x - 1);
endif;
Print;
.end
```

yields

```
F = f(1) + f(6)
```

Bracketing I

- Powers of variables can be extracted
- The terms are not nested for real, but information about brackets can be used in the next module

```
S x,y,z;
L F = x*y + x^2*y + x^2*z + 2;
Bracket x;   * extract powers of x
Print;
.end
```

```
F = + x * ( y )
    + x^2 * ( z + y )
    + 2;
```

Bracketing II

- Brackets can be indexed in the next module

```
S x,y,z;
L F = x*y + x^2*y + x^2*z + 2;
Bracket x;
.sort
L G = F[x^2];
Print G;   * only print G
.end
```

```
G = z + y
```

- Bracketed content can be collected in a function if it fits in memory

```
S x,y,z;
L F = x*y + x^2*y + x^2*z + 2;
Bracket x;
.sort
CF f;
Collect f;
Print;
.end
```

```
F = f(z + y)*x^2 + f(y)*x + f(2)
```

Tools for physicists

Vectors and indices

Contraction and Einstein summation:

```
Index i1,i2,i3;
Vector p1,p2,p3;
Local F = p1(i1)*(p2(i1)+p3(i3))*(p1(i2)+p2(i3));
Print;
.end
```

yields:

```
F = p1(i1)*p1(i2)*p3(i3) + p1(i1)*p2.p3
    + p1(i2)*p1.p2 + p2(i3)*p1.p2;
```

Make sure an index does not appear more than twice in a term!

Traces and gamma matrices

```
S D;
Index i1=D,i2=D;  * D-dimensional indices
Vector p1,p2;
Local F1 = g_(1, i1, i1);  * gamma matrices
Local F2 = g_(1, p1, i2);
Local F3 = g_(1, p1, p2);
tracen 1;  * n-dimensional trace of spin line 1
Print;
.end
```

```
F1 = 4*D;
F2 = 4*p1(i2);
F3 = 4*p1.p2;
```

Feynman rule application

```
S vhhg, gh, gl;  * ghost-gluon vertex, ghost, gluon
I i1,i2;
V Q,p1,p2,p3,p4;
CF vx,prop;
L F = vx(Q,p1,p2,i1,vhhg)
      *vx(-p1,-Q,-p2,i2,vhhg)
      *prop(p1,i1,i2,gl)*prop(p2,gh);

id prop(p1?,i1?,i2?,gl) = d_(i1,i2)/p1.p1;
id prop(p1?,gh) = 1/p1.p1;
id vx(p1?,p2?,p3?,i1?,vhhg) = -i_*vx(p1,p2,p3)*p1(i1);
```

```
F = vx(-p1,-Q,-p2)*vx(Q,p1,p2)*Q.p1*p1.p1^-1*p2.p2^-1;
```

Rational polynomials

Create rational polynomials using **polyratfun**:

```
S x, y;
CF rat;
polyratfun rat;   * enable polyratfun

L F = rat(y,x) + rat(x,1)*rat(1,y+1);
Print;
.end
```

```
F = rat(x^2 + y^2 + y,x*y + x)
```

IBP reduction of one-loop massive vacuum bubble

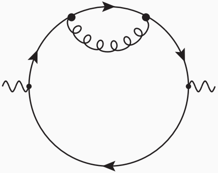
```
* rewrite k.k in the numerator
repeat id k1?.k1?*prop(k1?,n1?) =
  prop(k1, n1-1) + m^2 * prop(k1, n1);
id prop(k1?,n?) = prop(n);

* 1-loop IBP
id prop(n1?{<1}) = 0;
repeat id prop(n1?{>1}) = prop(-1 + n1)*
  rat((2 + (4-2*ep) - 2*n1), 2* (-1 + n1)) / m^2;
* master integral expanded in ep
id prop(1) = 1/ep + 1;
```

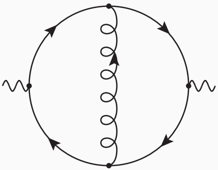
Exercises I

1. Get the maximum function argument, e.g. `f(1,4,2,5,3)` $\rightarrow$ `f(5)`
2. Represent a graph using vertex functions, e.g. `vx(Q,p1,p2)*vx(-Q,-p1,-p2)` and:
    2.1 Write an algorithm that checks if a graph is connected by shrinking edges
    2.2 Determine the number of loops, using $L = E - V + 1$

Large exercise preparation

- Obtain `Qgraf` from `http://cfif.ist.utl.pt/~paulo/qgraf.html`
- Download FORM configuration from `https://gist.github.com/benruijl/c37a8029a2b47a618dd1d2d46c631249`
- Compile with `gfortran`
- Generate ϕ^3 and QED input file





- Apply Feynman rules for two-loop photon self-energy graphs in QED
- Do the same for two-loop photon self-energy with QCD corrections
- Apply triangle reduction rule
- Apply one-loop master formula (next slide)

One-loop reduction

$$\int \frac{d^D k}{(2\pi)^D} \frac{\mathcal{P}_n(k)}{k^{2\alpha}(k-Q)^{2\beta}} =$$

$$\frac{1}{(4\pi)^2}(Q^2)^{D/2-\alpha-\beta} \sum_{\sigma\geq 0}^{[n/2]} G(\alpha,\beta,n,\sigma) Q^{2\sigma} \left\{ \frac{1}{\sigma!} \left(\frac{\Box}{4} \right)^\sigma \mathcal{P}_n(k) \right\}_{k=Q}$$

where

$$\mathcal{P}_n(k) = k_{\mu_1} k_{\mu_2} \cdots k_{\mu_n}, \qquad \Box = \partial^2/\partial k_\mu \partial k_\mu \ ,$$

$$G(\alpha,\beta,n,\sigma) = (4\pi)^\epsilon \frac{\Gamma(\alpha+\beta-\sigma-D/2)\Gamma(D/2-\alpha+n-\sigma)\Gamma(D/2-\beta+\sigma)}{\Gamma(\alpha)\Gamma(\beta)\Gamma(D-\alpha-\beta+n)}$$

- Implement d'Alambertian using `distrib_` and `dd_`

Hints

- Use sets to map numerical indices to Lorentz/Dirac indices
- Represent dirac algebra as `gamma(i,p,j)` and chain the functions

Good luck!