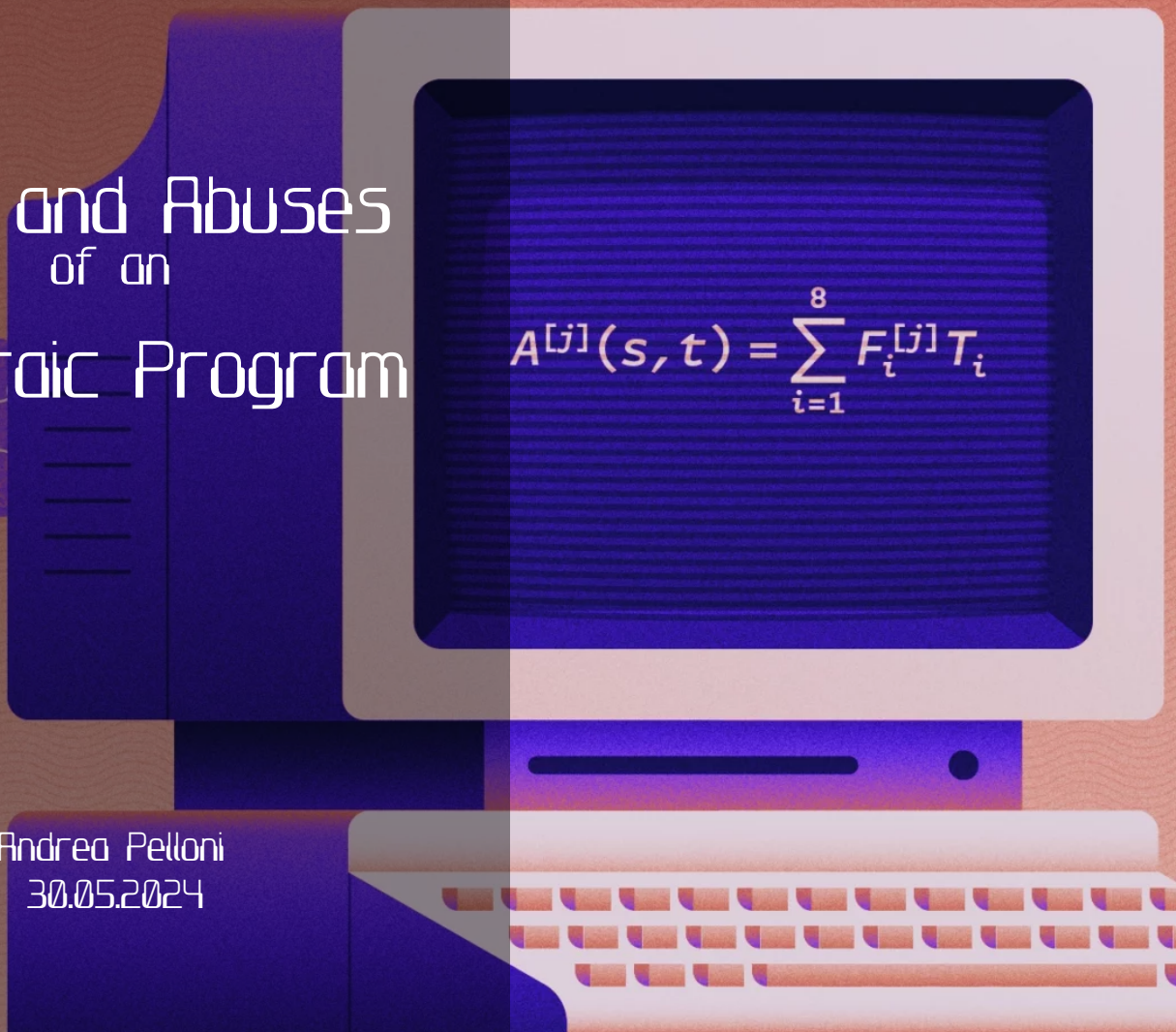
FORM and Symbolica developers meeting

Uses and Abuses of an Algebraic Program



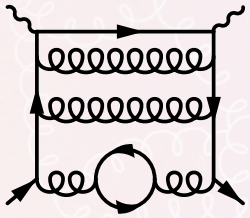
Andrea Pelloni
30.05.2024

ETH zürich

Profile of a computation

Before anything let's start by defining what is the **problem** we are facing. We can split the whole computation in few key **steps** :

Generation → **Simplification** → **Computation** → **Assamble**



Define the problem :

- Model we want to study
- Level of precision (e.g. # of loops, # of final states)
- Use Feynman rules (UFO: link)

Generation of all diagrams :

- FeynArt (link)
- QGraf (link)
- MadGraph (link)
- many more...

Cast into topologies :

- Contract external indices to obtain scalar integrals
- Tensor reduction with projectors
- Define container topologies of linearly idependent propagators

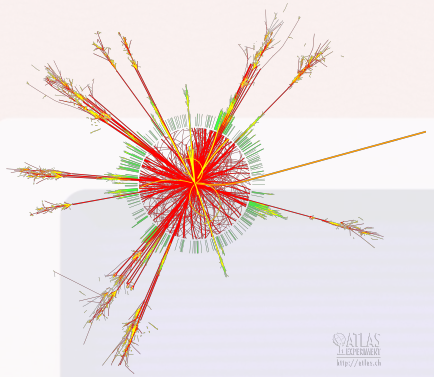
Reduce number of integrals :

- Collect by symmetry/algebra factors to resolve cancellation early on
- Use Integration by Part (**IBP**) identities
- Reduce to **master integrals** (e.g. FIRE, Kira, ...)

$$G(a_1,\ldots,a_n;x) = \int_0^x \frac{\mathrm{d}t}{t-a_1} G(a_2,\ldots,a_n;t)$$

Compute the master integrals

- Identify known expressions
- Create system of DEs
- Obtain expressions for boundary conditions
- Solve analytically/numerically



$$A^{[j]}(s,t) = \sum_{i=1}^{8}$$

Combine all the final expressions together in order to produce prediction for physical observables

Defining a Topology

A general **Feynman integral** doesn't need to be scalar or have linearly independent propagators

$$\int \left[\prod_{i}^{\#\text{ of loops}} \frac{\mathrm{d}^D k_i}{(2\pi)^D}\right] \frac{[\text{Color}]^{a,b} \cdot [\text{Spinor}]^{\mu\nu}}{\prod_{i}^{\#\text{ of propagators}} \left(\left(\sum_{j}^{\#\text{ of loops}} a_{ij} k_i + \sum_{j}^{\#\text{ of externals}} b_{ij} p_i\right)^2 + m_j^2\right)}$$

To set our machinery into motion we need **scalar integrals**

- Cross section with average/sum over external polarization will result in fully contracted indices
- We can isolate color factors outside of the integrand
- Open Lorentz indices can either be contracted with a **projection** or use **tensor reduction** to write the tensor structure using only loop-independent elements.

A **topology** is defined as

$$I_{\nu_1,\ldots,\nu_n} = \int \mathrm{d}^D k_i \frac{1}{D_1^{\nu_1} \ldots D_n^{\nu_n}}$$

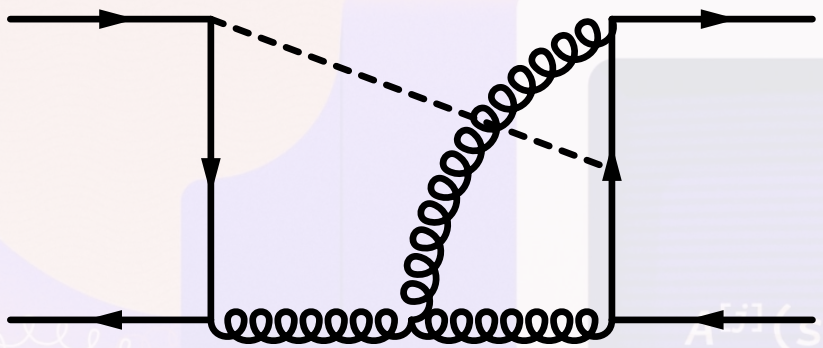
- $\{D_i\}$ are a set of linearly idependent propagators
- D_i can be any linear function of $k_i \cdot k_j$ and $k_i \cdot p_j$
- The topology is complete in the sense that any scalar product involving loop momenta can be written as

$$k_i \cdot v = \sum_j c_i D_i + \text{(scalars)}$$

Feynman Rules

```
*--#[ diag :
L F = +1/4*
 vx(-2,21,2,p2,-k1-p1-p2,k1+p1,2,10,6)*
 vx(-2,21,2,-p1,k1-k2+p1,-k1+k2,3,14,12)*
 vx(-2,21,2,-k2,k2+p2,-p2,17,18,1)*
 vx(-2,25,2,-k1-p1,k1,p1,7,8,4)*
 vx(-2,25,2,k1-k2,-k1,k2,13,9,16)*
 vx(21,21,21,k1+p1+p2,-k1+k2-p1,-k2-p2,11,15,19)*
 prop(2,in,p1,4)*
 prop(-2,in,p2,2)*
 prop(2,out,p1,3)*
 prop(-2,out,p2,1)*
 prop(2,virtual,k1+p1,7,6)*
 prop(2,virtual,-k1+k2,13,12)*
 prop(2,virtual,k2,17,16)*
 prop(21,virtual,-k1-p1-p2,11,10)*
 prop(21,virtual,k1-k2+p1,15,14)*
 prop(21,virtual,k2+p2,19,18)
 prop(25,virtual,k1,9,8)*
;
*--#] diag :
```

We evaluate this integral by means of the residue theorem.



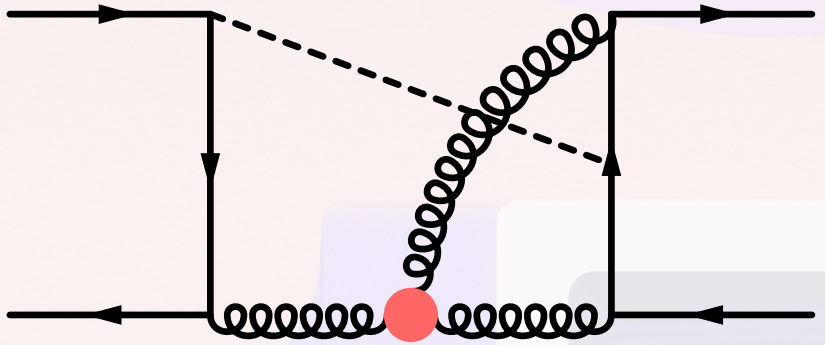
Feynman Rules

```
*--#[ diag :
L F = +1/4*
 vx(-2,21,2,p2,-k1-p1-p2,k1+p1,2,10,6)*
 vx(-2,21,2,-p1,k1-k2+p1,-k1+k2,3,14,12)*
 vx(-2,21,2,-k2,k2+p2,-p2,17,18,1)*
 vx(-2,25,2,-k1-p1,k1,p1,7,8,4)*
 vx(-2,25,2,k1-k2,-k1,k2,13,9,16)*
```

```
 vx(21,21,21,k1+p1+p2,-k1+k2-p1,-k2-p2,11,15,19)*
```

```
 prop(2,in,p1,4)*
 prop(-2,in,p2,2)*
 prop(2,out,p1,3)*
 prop(-2,out,p2,1)*
 prop(2,virtual,k1+p1,7,6)*
 prop(2,virtual,-k1+k2,13,12)*
 prop(2,virtual,k2,17,16)*
 prop(21,virtual,-k1-p1-p2,11,10)*
 prop(21,virtual,k1-k2+p1,15,14)*
 prop(21,virtual,k2+p2,19,18)
 prop(25,virtual,k1,9,8)*
;
*--#] diag :
```

We evaluate this integral by means of the residue theorem.



```
* Triple gluon vertex
#do i=1,1
    id once ifnomatch->skip vx(`GLU', `GLU', `GLU', p1?, p2?, p3?, idx1?, idx2?, idx3?) =
        i_ * gs * cOlf(colA[idx1], colA[idx2], colA[idx3]) *(
            - d_(lorentz[idx1], lorentz[idx3]) * p1(lorentz[idx2])
            + d_(lorentz[idx1], lorentz[idx2]) * p1(lorentz[idx3])
            + d_(lorentz[idx2], lorentz[idx3]) * p2(lorentz[idx1])
            - d_(lorentz[idx1], lorentz[idx2]) * p2(lorentz[idx3])
            - d_(lorentz[idx2], lorentz[idx3]) * p3(lorentz[idx1])
            + d_(lorentz[idx1], lorentz[idx3]) * p3(lorentz[idx2])
        );
    redefine i "0";
    label skip;

    B+ vx;
    .sort:3g;
    Keep brackets;
#enddo
.sort:3-gluon-vertices;
```

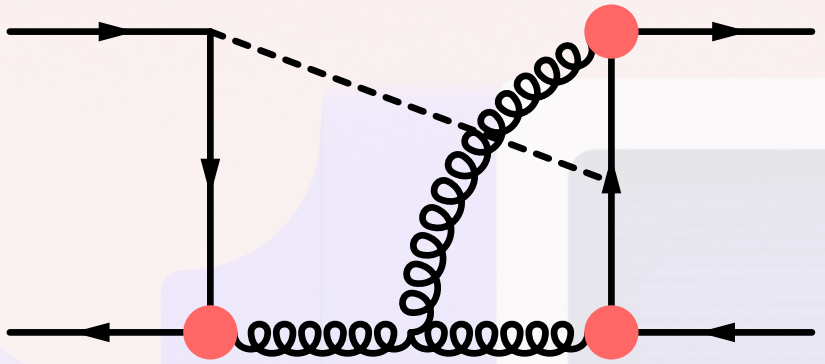
Feynman Rules

```
*--#[ diag :
L F = +1/4*
```

```
 vx(-2,21,2,p2,-k1-p1-p2,k1+p1,2,10,6)*
 vx(-2,21,2,-p1,k1-k2+p1,-k1+k2,3,14,12)*
 vx(-2,21,2,-k2,k2+p2,-p2,17,18,1)*
```

```
 vx(-2,25,2,-k1-p1,k1,p1,7,8,4)*
 vx(-2,25,2,k1-k2,-k1,k2,13,9,16)*
 vx(21,21,21,k1+p1+p2,-k1+k2-p1,-k2-p2,11,15,19)*
 prop(2,in,p1,4)*
 prop(-2,in,p2,2)*
 prop(2,out,p1,3)*
 prop(-2,out,p2,1)*
 prop(2,virtual,k1+p1,7,6)*
 prop(2,virtual,-k1+k2,13,12)*
 prop(2,virtual,k2,17,16)*
 prop(21,virtual,-k1-p1-p2,11,10)*
 prop(21,virtual,k1-k2+p1,15,14)*
 prop(21,virtual,k2+p2,19,18)
 prop(25,virtual,k1,9,8)*
;
*--#] diag :
```

We evaluate this integral by means of the residue theorem.



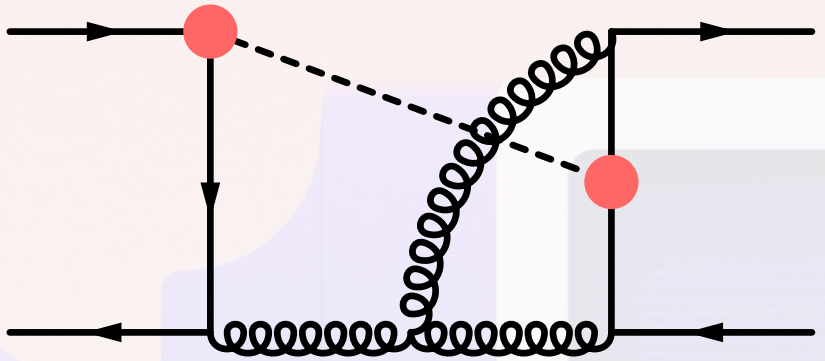
$$A^{[j]}(s,t) = \sum_{i=1}^{8}$$

```
* vertices
id vx(x1?{`QBAR'}, `GLU', x2?{`Q'}, p1?, p2?, p3?, idx1?, idx2?, idx3?) =
    -gs * gamma(dirac[idx1], lorentz[idx2], dirac[idx3]) * T(colF[idx1], colA[idx2], colF
    [idx3])
    ;
id vx(x1?{`QBAR'}, `H', x2?{`Q'}, p1?, p2?, p3?, idx1?, idx2?, idx3?) =
    -gyq(x1) * i_ * d_(dirac[idx1], dirac[idx3]) * d_(colF[idx1], colF[idx3])
    ;
```

Feynman Rules

```
*--#[ diag :
L F = +1/4*
 vx(-2,21,2,p2,-k1-p1-p2,k1+p1,2,10,6)*
 vx(-2,21,2,-p1,k1-k2+p1,-k1+k2,3,14,12)*
 vx(-2,21,2,-k2,k2+p2,-p2,17,18,1)*
 vx(-2,25,2,-k1-p1,k1,p1,7,8,4)*
 vx(-2,25,2,k1-k2,-k1,k2,13,9,16)*
 vx(21,21,21,k1+p1+p2,-k1+k2-p1,-k2-p2,11,15,19)*
 prop(2,in,p1,4)*
 prop(-2,in,p2,2)*
 prop(2,out,p1,3)*
 prop(-2,out,p2,1)*
 prop(2,virtual,k1+p1,7,6)*
 prop(2,virtual,-k1+k2,13,12)*
 prop(2,virtual,k2,17,16)*
 prop(21,virtual,-k1-p1-p2,11,10)*
 prop(21,virtual,k1-k2+p1,15,14)*
 prop(21,virtual,k2+p2,19,18)
 prop(25,virtual,k1,9,8)*
;
*--#] diag :
```

We evaluate this integral by means of the residue theorem.



$$A^{[1]}(s,t) = \sum_{i=1}^{8}$$

```
* vertices
id vx(x1?{`QBAR'}, `GLU', x2?{`Q'}, p1?, p2?, p3?, idx1?, idx2?, idx3?) =
    -gs * gamma(dirac[idx1], lorentz[idx2], dirac[idx3]) * T(colF[idx1], colA[idx2], colF
    [idx3])
    ;
id vx(x1?{`QBAR'}, `H', x2?{`Q'}, p1?, p2?, p3?, idx1?, idx2?, idx3?) =
    -gyq(x1) * i_ * d_(dirac[idx1], dirac[idx3]) * d_(colF[idx1], colF[idx3])
    ;
```

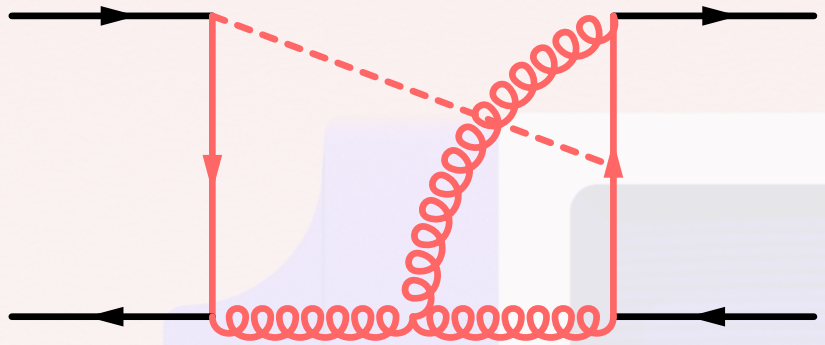
Feynman Rules

```
*--#[ diag :
L F = +1/4*
 vx(-2,21,2,p2,-k1-p1-p2,k1+p1,2,10,6)*
 vx(-2,21,2,-p1,k1-k2+p1,-k1+k2,3,14,12)*
 vx(-2,21,2,-k2,k2+p2,-p2,17,18,1)*
 vx(-2,25,2,-k1-p1,k1,p1,7,8,4)*
 vx(-2,25,2,k1-k2,-k1,k2,13,9,16)*
 vx(21,21,21,k1+p1+p2,-k1+k2-p1,-k2-p2,11,15,19)*
 prop(2,in,p1,4)*
 prop(-2,in,p2,2)*
 prop(2,out,p1,3)*
 prop(-2,out,p2,1)*
```

```
 prop(2,virtual,k1+p1,7,6)*
 prop(2,virtual,-k1+k2,13,12)*
 prop(2,virtual,k2,17,16)*
 prop(21,virtual,-k1-p1-p2,11,10)*
 prop(21,virtual,k1-k2+p1,15,14)*
 prop(21,virtual,k2+p2,19,18)
 prop(25,virtual,k1,9,8)*
```

```
;
*--#] diag :
```

We evaluate this integral by means of the residue theorem.

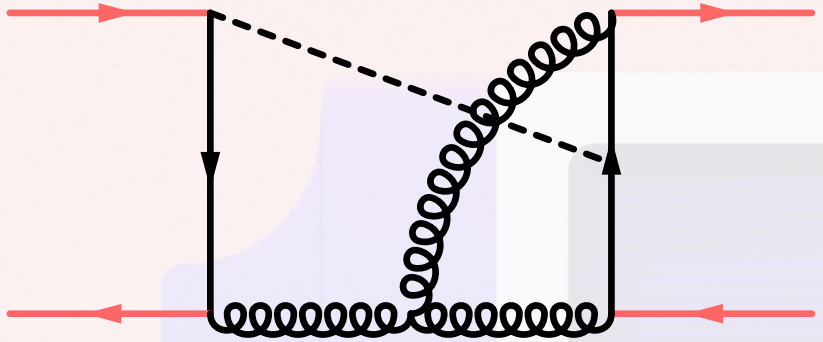


```
* virtual edges
id prop(`GLU', virtual, p?, idx1?, idx2?) =
    - i_ * d_(lorentz[idx1], lorentz[idx2]) * d_(colA[idx1], colA[idx2])
    ;
id prop(x?{`Q'}, virtual, p?, idx1?, idx2?) = i_ * d_(colF[idx2], colF[idx1]) * (
    gamma(dirac[idx2], p, dirac[idx1])
    + masses(x) * gamma(dirac[idx2], dirac[idx1])
    );
id prop(x?{`QBAR'}, virtual, p?, idx1?, idx2?) = - i_  * d_(colF[idx1], colF[idx2]) * (
    gamma(dirac[idx1], p, dirac[idx2]) + masses(x) * gamma(dirac[idx1], dirac[idx2])
    );
id prop(`H', virtual, p?, idx1?, idx2?) = -i_;
```

Feynman Rules

```
*--#[ diag :
L F = +1/4*
 vx(-2,21,2,p2,-k1-p1-p2,k1+p1,2,10,6)*
 vx(-2,21,2,-p1,k1-k2+p1,-k1+k2,3,14,12)*
 vx(-2,21,2,-k2,k2+p2,-p2,17,18,1)*
 vx(-2,25,2,-k1-p1,k1,p1,7,8,4)*
 vx(-2,25,2,k1-k2,-k1,k2,13,9,16)*
 vx(21,21,21,k1+p1+p2,-k1+k2-p1,-k2-p2,11,15,19)*
 prop(2,in,p1,4)*
 prop(-2,in,p2,2)*
 prop(2,out,p1,3)*
 prop(-2,out,p2,1)*
 prop(2,virtual,k1+p1,7,6)*
 prop(2,virtual,-k1+k2,13,12)*
 prop(2,virtual,k2,17,16)*
 prop(21,virtual,-k1-p1-p2,11,10)*
 prop(21,virtual,k1-k2+p1,15,14)*
 prop(21,virtual,k2+p2,19,18)
 prop(25,virtual,k1,9,8)*
;
*--#] diag :
```

We evaluate this integral by means of the residue theorem.



```
* do the spin sum external particles
repeat id prop(x?{`Q'}, in, p?, idx1?)*prop(x?{`Q'}, out, p?, idx2?) =
    d_(colF[idx1], colF[idx2])*(gamma(dirac[idx1], p, dirac[idx2])
    + masses(x)*gamma(dirac[idx1], dirac[idx2]))
    ;
repeat id prop(x?{`QBAR'}, out, p?, idx1?)*prop(x?{`QBAR'}, in, p?, idx2?) =
    d_(colF[idx1], colF[idx2])*(gamma(dirac[idx1], p, dirac[idx2])
    - masses(x)*gamma(dirac[idx1], dirac[idx2]))
    ;
```

Scalar Integrals

FORM allows us to efficiently dress the **Feynman Rules** to the graph representation and to quickly perform the algebraic operations

- **Dirac Algebra** with the use of `g_`
- **Color Algebra** implemented for $SU(N)$ with
  - Fierz Identity
  $$(T^a)_{i_1 j_1}(T^a)_{i_2 j_2} = \delta_{i_1 j_2}\delta_{i_2 j_1} - \frac{1}{N}\delta_{i_1 j_1}\delta_{i_2 j_2}$$
  - structure functions definition
  $$f^{abc} = \frac{1}{T_R}\,\mathrm{tr}\left(T^a[T^b, T^c]\right)$$
  - Casimir Operators

The final step towards proper topologies can be done with:

- projectors acting on the external particles (helicity, tensor structure, ...)
- match statement to map into a topology with a well defined set ot propagators D_i

We are now in the situation of having to deal with only **scalar integrals** with more or less complicated numerators. There are different approaches from this point onwards which we can oversimplify into two:

- direct computation of the integrals at hand via numerical methods
- further simplification of the system by means of Integration by Part Identities (**IBP**) into a master integral basis

Loop-Tree Duality

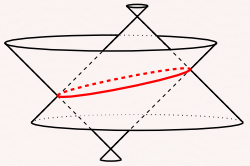
We are now dealing with integrals of the form:

$$\int \left[\prod_{i}^{\#\text{ of loops}} \frac{\mathrm{d}^D k_i}{(2\pi)^D} \right] \frac{\mathcal{N}(\{k_i\}, \{p_j\})}{D_1^{\nu_1} \dots D_n^{\nu_n}}$$

One way to approach the direct integration is by means of Loop-Tree Duality (**LTD**)

Loop-Tree duality key aspects:

- Performing the integration over the energy analytically
- Introduces cuts similar to **unitarity cuts**
- Space of integration goes from 4-dimensional **Minkowski space** to a 3-dimensional **Euclidean space**
- Bounded threshold singularities in Euclidean space



Loop-Tree Duality

Each squared propagator contains poles in the energy in both the upper and lower complex plane

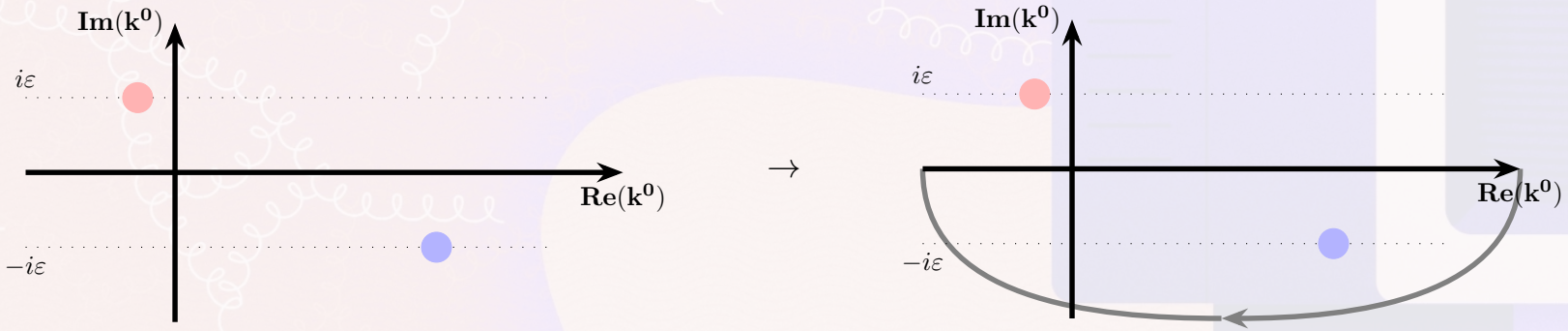
Propagator $\frac{1}{D_i}$:

$$D_i = (k + q_i)^2 - m^2 + i\epsilon = \underbrace{((k^0 + q_i^0) - E_i)}_{\text{Positive Energy}} \underbrace{((k^0 + q_i^0) + E_i)}_{\text{Negative Energy}}$$

On-shell energy:

$$E_i = \sqrt{(\vec{k} + \vec{q}_i)^2 + m^2 - i\epsilon}$$

We can close the contour of integration when condering the integral w.r.t. the energy component of the loop momentum



Loop-Tree Duality

Each squared propagator contains poles in the energy in both the upper and lower complex plane

Propagator $\frac{1}{D_i}$:

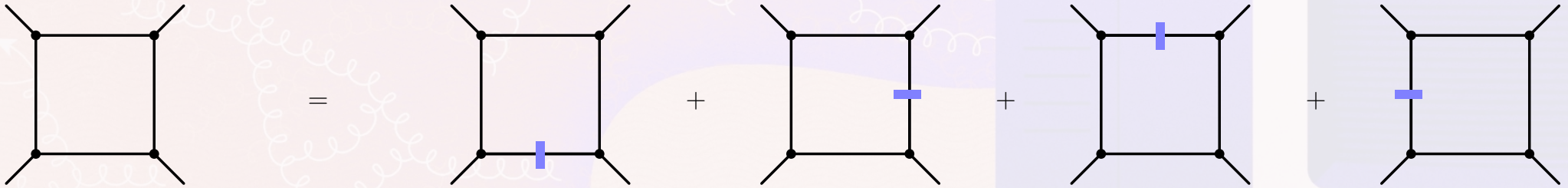
$$D_i = (k+q_i)^2 - m^2 + i\epsilon = \underbrace{((k^0+q_i^0)-E_i)}_{\text{Positive Energy}} \underbrace{((k^0+q_i^0)+E_i)}_{\text{Negative Energy}}$$

On-shell energy:

$$E_i = \sqrt{(\vec{k}+\vec{q}_i)^2 + m^2 - i\epsilon}$$

- Taking the residue over the **positive** energy solution is equivalent to cutting the propagator:

$$\delta_+((k+q_i)^2 - m^2 + i\epsilon^2)$$



$$A^{[j]}(s,t) = \sum_{i=1}^{8}$$

Loop-Tree Duality

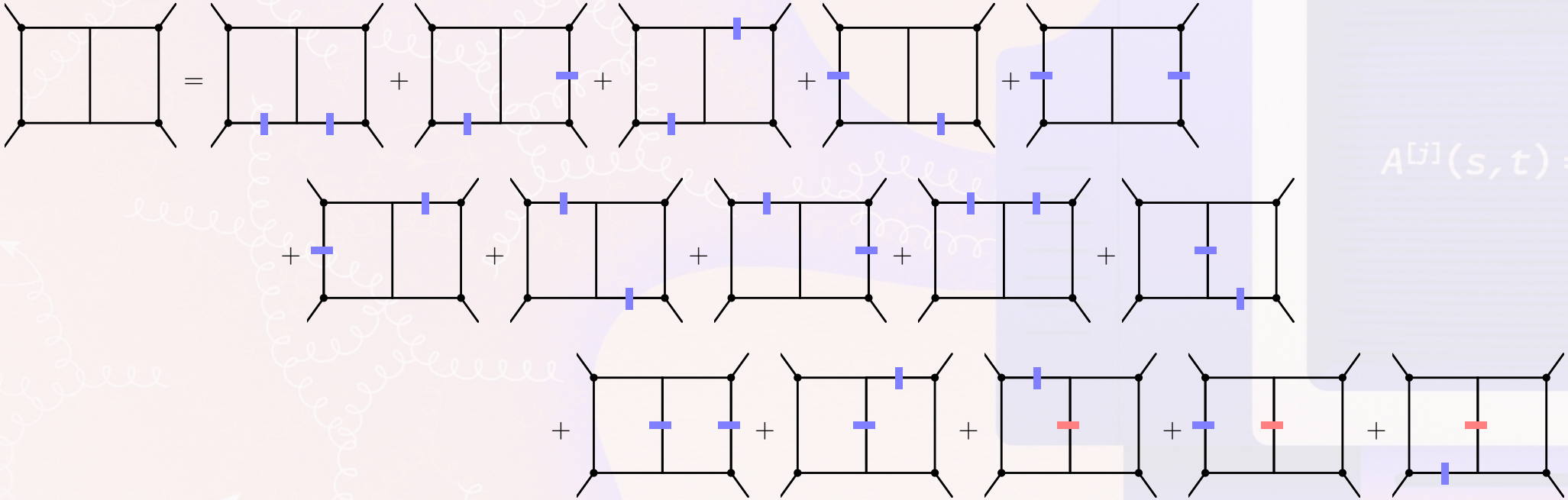
Each squared propagator contains poles in the energy in both the upper and lower complex plane

Propagator $\frac{1}{D_i}$:

$$D_i = (k + q_i)^2 - m^2 + i\epsilon = \underbrace{\left((k^0 + q_i^0) - E_i\right)}_{\text{Positive Energy}} \underbrace{\left((k^0 + q_i^0) + E_i\right)}_{\text{Negative Energy}}$$

On-shell energy:

$$E_i = \sqrt{(\vec{k} + \vec{q}_i)^2 + m^2 - i\epsilon}$$



Pole structure

Singularities can be divided in **two categories** :

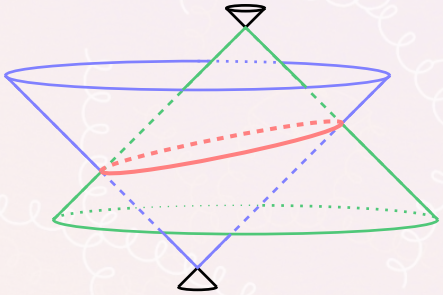
$$D_1 = (k^0 + q_1^0)^2 - E_1^2$$

$$D_2 = (k^0 + q_2^0)^2 - E_2^2$$

$$\xrightarrow{\text{Cut } D_1} \quad \frac{\delta_+(D_1)}{D_2} = \frac{\delta_+(D_1)}{E_{12}\,H_{12}}$$

Ellipsoid

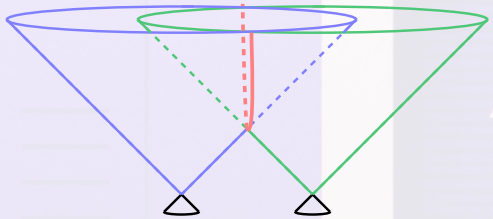
$$E_{12} := \left((q_1^0 - q_2^0) - E_1 - E_2\right)$$



- Need **deformation** $k \to k - i\kappa(k)$
- For single ellipsoids κ can be chosen to the **gradient** of E_{12}

Hyperboloid

$$H_{12} := \left((q_1^0 - q_2^0) - E_1 + E_2\right)$$



- **Spurious** singularities
- Source of numerical instabilities

CODE

```
*   Compute the LTD expression for arbitrary topologies
#procedure LTD(LOOPS);
multiply norm(1)*num();

#message "LOOPS: `LOOPS'"
#do i=0,{`LOOPS'-1}
* Normalise by the contour orientation
    id norm(n?) = norm(-n);

* Identify poles in the loop momentum energy
    splitarg (k`i') prop;
    id prop(k`i',E?) = prop(0,k`i',E);
    id all prop(p?,k?,E?) = x(p,k,E);
    factarg (-1) x 2;
    id x(p?,k?,y?{-1,1},E?) = x(y*p,0,E)*x(y*p,E,0);

* Split the poles by their position in the complex plane
    splitarg x;
    repeat id x(?y1,+E?energies,?y2,E1?,E2?) = x(?y1,?y2,E1+E,E2);
    repeat id x(?y1,-E?energies,?y2,E1?,E2?) = x(?y1,?y2,E1,E2+E);

* Collect arguments
    transform, x, addargs(1,last-2);
    transform, prop, addargs(1,last-1);
    .sort

* take residue
    id x(p?,0,E?)*x(p?,E1?,E2?)*norm(n?)*num(?y) = replace_(k`i',E-p)*norm(n*(
    E+E1-E2))*num(?y,E-p);
    if ( count(x,1) ) discard;
    .sort
print +s;
.sort
```

$$D_i = (k+q_i)^2 - m^2 = \underbrace{\left((k^0+q_i^0)-E_i\right)}_{\text{Positive Energy}} \underbrace{\left((k^0+q_i^0)+E_i\right)}_{\text{Negative Energy}}$$

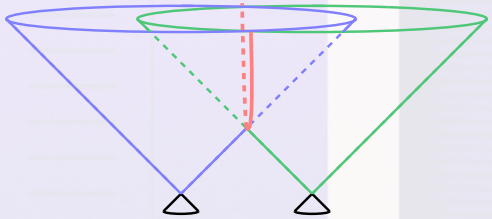
- identify propagators that depend on the loop momentum energy

$$\frac{1}{D_1 D_2 \cdots D_n} \to \sum_i \frac{\delta_+(D_i)}{\prod_{j\neq j} D_j}$$

-

Hyperboloid

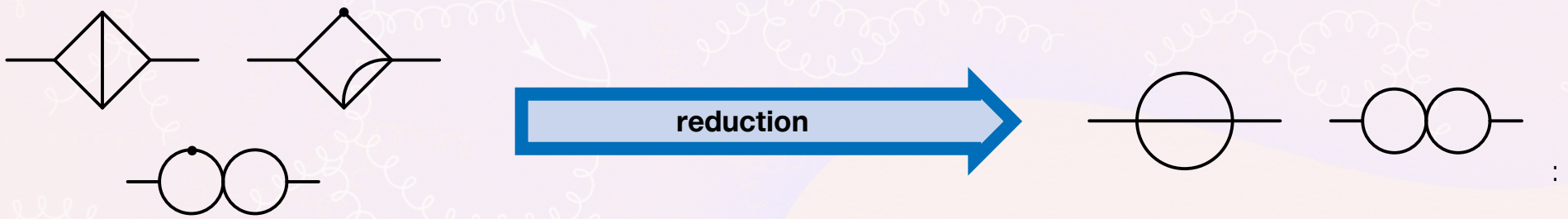
$$H_{12} := \left((q_1^0 - q_2^0) - E_1 + E_2\right)$$



$$A^{[j]}(s,t) = \sum_{i=1}^{8}$$

- Higher propagators will require derivatives
- In **FORM** we need to isolate the information termwise compared with a `Module` in `Mathematica`

Reduction to Master Integrals



- One of the standard tools to be used to resize the magnitude of the problem is to identify, by means of **Integration By Part** (IBP) identities, a set of Master integrals to span the space of the scalar integrals that appear in the computation:
- Within each **topology** we perform a reduction to **master integrals** :

$$\underbrace{I^{(n)}(s_{ij},\epsilon)}_{\text{Topology Integrals}} = \sum_i \underbrace{c_i(s_{ij},\epsilon)}_{\text{Coefficients}} \cdot \underbrace{M_i^{(n)}(s_{ij},\epsilon)}_{\text{Masters}}$$

  - c_i: Coefficient that depends on the external kinematics together with the dimensional regulator ϵ.

    Rational Functions

  - M_i: Master integrals (i.e. scalar integrals) that depend on the external kinematics together with the dimensional regulator.

    Special Functions (Multiple Polylogarithms, Elliptic Functions,...)

There are many publicly available programs that perform reductions to master integrals (AIR, FIRE, Reduze, Kira, LiteRed) each one with their strengths and weaknesses.

Problem with parsing

When dealing with the reductions to master integrals with obtain coefficients that can encode a good deal of complexity. **FORM** doesn't like the **factorized** form of this expression and is better to store them in tables with the expanded form.

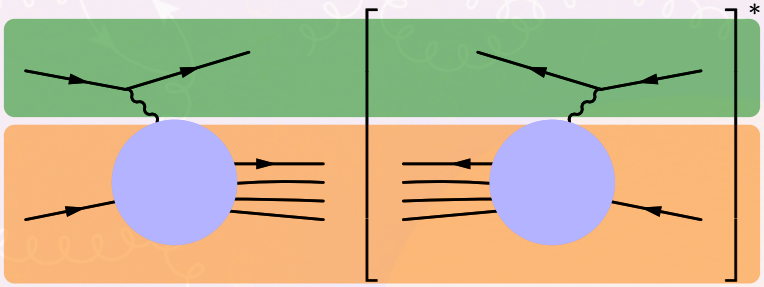
This is an example of a **denominator** that would take ∼3 minutes to evaluate to the final expression.

```
#--
Symbol x,w,D;

L F = 360*(-4+D)^5*
        (-3+D)^2*
        (-2+D)^2*
        (-1+D)*
        (-7+2*D)^2*
        (-10+3*D)^2*
        (-8+3*D)*
        (-13+4*D)*
        (-18+5*D)*
        (-16+5*D)^2*
        (-14+5*D)*
        (77-54*D+9*D^2)*
        (-1+w)^2*
        w^3*
        (1+w)^3*
        (2+w)^2*
        (-3+2*w)*
        (4-D-10*w+3*D*w)*
        (138-63*D+7*D^2-372*w+174*D*w-20*D^2*w+192*w^2-96*D*w^2+12*D^2*w^2);
print +s;
.end
```

```
   #--

Time =     206.59 sec    Generated terms =  362797056
               F         Terms in output =        312
                         Bytes used      =      12180
```

Deep Inelastic Scattering

Probing the **hadron** structure by mean of high energetic **leptons** :



- Factorize the **leptonic** from the **hadronic** part in the cross-section

$$\frac{\sigma_{DIS}}{dxdy} = \frac{2\pi\alpha_{em}^2}{Q^2} L^{\mu\nu} W_{\mu\nu}$$

- Identify DIS **coefficient functions**

$$W_{\mu\nu} = \left(P^\mu - \frac{(P\cdot q)q_\mu}{q^2}\right)\left(P^\nu - \frac{(P\cdot q)q_\nu}{q^2}\right)\frac{F_1(x,Q^2)}{P\cdot q} + \left(-g_{\mu\nu} + \frac{q^\mu q^\nu}{q^2}\right)F_2(x,Q^2) + i\epsilon_{\mu\nu\rho\sigma}\frac{P^\rho q^\sigma}{2P\cdot q}F_3(x,Q^2)$$

- Where the **hadronic** and **partonic** quantities are related via the **PDF**:

$$F_a(x,Q^2) = \sum_i \left[f_i(\xi) \otimes \hat{F}_{a,i}(\xi,Q^2)\right](x)$$

- The problem can be simplified by using the optical theorem for extracting the **Mellin moments** of the process.

Mellin Moments

Objective:

- Compute the hadronic cross-section $\hat{W}_{\mu\nu}$ using the forward scatting $\hat{T}_{\mu\nu}$

How:

- Compute the **Mellin moments** of the structure functions:

$$F_a(x,Q^2) = \sum_i \left[f_i(\xi) \otimes \hat{F}_{a,i}(\xi,Q^2) \right](x)$$

with the **Mellin transform** defined by

$$M[f(x)](N) = \int_0^1 \mathrm{d}x\, x^{N-1} f(x)$$

- The Mellin moments of **cross-section** correspond to the expansion coefficients around $\omega = \frac{1}{x} = 0$ of the **Forward Scattering Amplitude** :

$$M[\hat{W}_{\mu\nu}](N) = \frac{1}{N!} \left[\frac{\mathrm{d}^N T_{\mu\nu}}{\omega^N} \right|_{\omega=0}$$

Expansion

We can reduce the problem to the task of expanding the **Master integrals** around $\omega = 0$. In order to find the expressions that describes them, and by extension all of our integrals, we can set up a **system of differential equations** (DE).

$$\frac{\partial}{\partial t}\ \vec{M}(t,\epsilon) = \underbrace{A(t,\epsilon)}_{\text{Differential Matrix}} \cdot \vec{M}(t,\epsilon)$$

The master formula for the expansion is given by

$$M(z,\ldots;\epsilon) = \sum_{a,n} z^{n-a\epsilon}\, m_{a,n}(z,\ldots;\epsilon)$$

- **n**: index of the Laurent series with bounded lower index
- **a**: sector of the expansion. Do not mix under DEs!

Only simple poles in z

$$A = \frac{A_{-1}}{\omega} + \sum_{k=0}^{\infty} A_k\, z^k \qquad \vec{M} = \sum_{k=0}^{\infty} \vec{m}_k\, z^k$$

$$\underbrace{((k+1)\mathbb{1} - A_{-1})}_{:=B_k} \cdot \vec{m}_{k+1} = \sum_{j=0}^{k} A_j\, \vec{m}_{k-j}$$

$\det(B_k) \neq 0$

Recursive Expression:

$$\vec{m}_{k+1} = B_k^{-1} \cdot \left(\sum_{j=0}^{k} A_j\, \vec{m}_{k-j}\right)$$

$\det(B_k) = 0$

Gaussian Elimination:

Required by a **finite number** of k

Expansion

$$\vec{m}_{k+1} = B_k^{-1} \cdot \left(\sum_{j=0}^{k} A_j \, \vec{m}_{k-j} \right)$$

- The computation of the inverse matrix B_k^{-1} is done in `Maple` and stored for generic values of k in a table.
- The expansion is split into three blocks take full advantage of the workers in **tFORM**

```
...
Fill Binv(1,16,12) = rat(2*(-1+2*ep)^2,(-1+3*ep)*(ep+Kord)*(2*ep+Kord));
Fill Binv(1,16,16) = rat(1,2*ep+Kord);
...
```

```
** Process Table elements for Binv
Local F= sum_(k,`$mink',`expORDER',
  sum_(i1,1,`$masterSIZE',
  sum_(i2,1,`$masterSIZE',
    C(i1,i2,k)*BB(`topoN',i1,i2,k)
  )));

id BB(i1?,i2?,i3?,i4?$y)=Binv(i1,i2,i3);

argument rat;
mu replace_(Kord,$y);
endargument;

B C;
.sort:collect-C;
FillExpression invB = F(C);
*printtable invB;
.sort:init-invB;
Drop;

.sort
```

```
* Solve order by order in w
#do k=`$mink',`expORDER'
    #message == Order k = `k';

    Local F= sum_(k1,1,`k', Zz(k1));
    .sort:sum-1;
    id Zz(k1?) =
       sum_(i2,1,`$masterSIZE',
          sum_(i1,1,`$masterSIZE',
             Aexp(i1,i2,k1)
             *m(`topoN',i2,`k'-k1)
             *C(`topoN', i1,`k')
          ));
    .sort:sum-2;
    id C(x1?, i1?, k?) = sum_(i3,1,`$masterSIZE', C(x1,i3,k)*
    invB(i3,i1,`k'));
    .sort:sum-3;
...
```

Rational Function expansion

Since we want to compute a Taylor Series it is natural that one has to deal with many expansions. This is a problem that also shows up in other cases (e.g. threshold expansion) and is generalized to **Laurent series**.

How can we deal with this in **FORM** ?

- We can use built-in function with `PolyRatFun`
- The expansion is defined by the number of terms not the order
- There are limination with the expansion working only with uni-variate expressions

One main drawback of using master integrals is that they do not need to reflect any of the precess properties.Simple solver
One clear example is the presence of un-physical poles such as:

$$\frac{1}{\epsilon-\omega} = \frac{1}{\epsilon}\sum_{n=0}^{\infty}\left(\frac{\omega}{\epsilon}\right)^n$$

The issue is that the two **limits do not commute** and we cannot expand "safely" in both ϵ and ω:

$$r = \lim_{n\to\infty}\frac{c_n}{c_{n+1}} = \epsilon$$

However we know that this type of pole cancel in the final expression so we should be able to **safely** truncate the series

$$\frac{1}{\epsilon-\omega} - \left(\frac{\omega}{\epsilon}\right)^m \frac{1}{\epsilon-\omega} = \frac{1}{\epsilon}\sum_{n=0}^{m}\left(\frac{\omega}{\epsilon}\right)^n$$

$$\left(\frac{\omega}{\epsilon}\right)^m\left(\frac{1}{\epsilon}\sum_{n=0}^{N-1}\left(\frac{\omega}{\epsilon}\right)^n - \left(\frac{1}{\epsilon}\sum_{n=0}^{N-1}\left(\frac{\omega}{\epsilon}\right)^n + \mathcal{O}(\omega^N)\right) + \mathcal{O}(\omega^N)\right) = \frac{1}{\epsilon}\sum_{n=0}^{m}\left(\frac{\omega}{\epsilon}\right)^n + \mathcal{O}(\omega^N)$$

Wishlist

Features :

- Select the order of expansion insted of the number of coefficients

```
Symbol ep;
CF rat;

L F = rat(ep^2,1-ep);
.sort

PolyRatFun rat(expand,ep,3);
print +s;
.sort:series-expand;
```

```
   F =
       + rat(ep^2 + ep^3 + ep^4 + ep^5)
      ;
```

```
Symbol ep;
CF rat;

L F = rat(ep^2,1-ep);
.sort

PolyRatFun rat(truncate,ep,3);
print +s;
.end:series-expand;
```

```
   F=
       + rat(ep^2 + ep^3)
      ;
```

Wishlist

Features :

- Select the order of expansion insted of the number of coefficients
- PolyRatFun allows for multivariate expansion (with proper flags)

```
S ep,w;
CF rat,den;

#include rat_expand.frm;
#define ExpansionVariablesSize "2";
Fill ExpansionVariables(1) = ep; Fill ExpansionOrders(1) = 2;
Fill ExpansionVariables(2) = w;  Fill ExpansionOrders(2) = 4;


* Define Function
L F = rat(1,1+w+ep);
.sort

* Expand in w end ep
#call ratExpand(2);
B ALARM;
print +s;
.end
```

```
   F=
      +ALARM(ep,0,2)*ALARM(w,0,4)*(
         +w*rat(-3*ep^2+2*ep-1,1)
         +w^2*rat(6*ep^2-3*ep+1,1)
         +w^3*rat(-10*ep^2+4*ep-1,1)
         +w^4*rat(15*ep^2-5*ep+1,1)
         +rat(ep^2-ep+1,1)
         );
```

Wishlist

Features :

- Select the order of expansion insted of the number of coefficients
- PolyRatFun allows for multivariate expansion (with proper flags)

```
S ep,w;
CF rat,den;
PolyRatFun rat;

#include rat_expand.frm;
#define ExpansionVariablesSize "2";
* Expand ep and w
Fill ExpansionVariables(1) = w;  Fill ExpansionOrders(1) = 4;
Fill ExpansionVariables(2) = ep; Fill ExpansionOrders(2) = 2;
* Define Function
L F = rat(1,w+ep);
.sort
* Expand
#call ratExpand(2);
print +s;
.sort
* Check
mu (w+ep);
id ep^x? = rat(ep^x,1);
print +s;
.end
```

```
   F=
      +ALARM(ep,-5,2)*ALARM(w,0,4)*(
         +w*rat(-1,ep^2)
         +w^2*rat(1,ep^3)
         +w^3*rat(-1,ep^4)
         +w^4*rat(1,ep^5)
         +rat(1,ep)
         );

   F=
      +ALARM(ep,-5,2)*ALARM(w,0,4)*w^5*rat(1,ep^5)
      +ALARM(ep,-5,2)*ALARM(w,0,4)*rat(1,1)
      ;
```

```
S ep,w;
CF rat,den;
PolyRatFun rat;

#include rat_expand.frm;
#define ExpansionVariablesSize "2";
* Expand ep and w
Fill ExpansionVariables(1) = ep; Fill ExpansionOrders(1) = 2;
Fill ExpansionVariables(2) = w;  Fill ExpansionOrders(2) = 4;
* Define Function
L F = rat(1,w+ep);
.sort
* Expand
#call ratExpand(2);
print +s;
 .sort
* Check
PolyRatFun rat;
mu (w+ep);
id ep^x? = rat(ep^x,1);
print +s;
.end
```

```
   F=
      +ALARM(ep,0,2)*ALARM(w,-3,4)*(
         +w^-3*rat(ep^2,1)
         +w^-2*rat(-ep,1)
         +w^-1*rat(1,1)
         );

   F=
      +ALARM(ep,0,2)*ALARM(w,-3,4)*w^-3*rat(ep^3,1)
      +ALARM(ep,0,2)*ALARM(w,-3,4)*rat(1,1)
      ;
```

Wishlist

Features :

- Select the order of expansion insted of the number of coefficients
- PolyRatFun allows for multivariate expansion (with proper flags)
- Matrix manipulation
  - RowReduction
  - LU factorization
  - ...

  This could go towards linking to the `FLINT` library also for improved rational function arithmetic
- Being able to simplify/manipulate tables directly

Functionalities

- preprocessor `#while` loop
- Simple solver
- Extending pattern matching for `f(a*b*x)` for `id f(a??*x) = a??*f(x)`

Accessibility

- Improved documentation (migration to markdown based website)

Wishlist

Features :

- Select the order of expansion insted of the number of coefficients
- PolyRatFun allows for multivariate expansion (with proper flags)
- Matrix manipulation
  - RowReduction
  - LU factorization
  - ...

  This could go towards linking to the `FLINT` library also for improved rationa
- Being able to simplify/manipulate tables directly

Functionalities

- preprocessor `#while` loop
- **Simple solver**
- Extending pattern matching for `f(a*b*x)` for `id f(a??*x) = a??*f(x)`

Accessibility

- Improved documentation (migration to markdown based website)

```
Symbol x,y;
Auto Symbol a;
CF expr, sol;

L F = expr( a0 + a1 * x + a2 * y + a3 * x * y^2);
.sort

* Set up expression to be solved
SplitArg ((x)), expr;
Transform expr, addargs(2,last);
.sort

* Check if solution is possible
argument expr,2;
    if ( count(x,1) > 1 );
        print "solveERROR: %t";
        exit "solveERROR: solver argument must be linear in x";
    endif;
endargument;
.sort

* Solve
PolyRatFun sol;
id expr(a1?,a2?) = sol(a1,a2/x);
id expr(a1?) = sol(0);
print;
.end
```

```
   F =
      sol(y*a2 + a0,y^2*a3 + a1);
```

Wishlist

Features :

- Select the order of expansion insted of the number of coefficients
- PolyRatFun allows for multivariate expansion (with proper flags)
- Matrix manipulation
  - RowReduction
  - LU factorization
  - ...

  This could go towards linking to the `FLINT` library also for improved rational function arithmetic
- Being able to simplify/manipulate tables directly

Functionalities

- preprocessor `#while` loop
- Simple solver
- Extending pattern matching for `f(a*b*x)` for `id f(a??*x) = a??*f(x)`

Accessibility

- Improved documentation (migration to markdown based website)

And remember...