FeynCalc goes multiloop

Vladyslav Shtabovenko
Karlsruhe Institute of Technology
Institute for Theoretical Particle Physics
in collaboration with R. Mertig and F. Orellana

20th International Workshop on
Advanced Computing and Analysis Techniques
in Physics Research
Daejeon, South Korea
1st of December 2021
Outline

1 Motivation: Why multiloop with \texttt{FeynCalc}?

2 Symanzik polynomials and Pak algorithm

3 \textbf{Using \texttt{FeynCalc} in multiloop calculations}
   - Topology identification
   - Master integrals

4 Summary and Outlook
Motivation: Why multiloop with FeynCalc?
Motivation: Why multiloop with FeynCalc?

FeynCalc offers a toolbox-oriented approach to symbolic Feynman diagram calculations.

Not foolproof: correctness of the results $\propto$ user's understanding of QFT

Very useful for people who know what they want to calculate

FeynCalc has plenty of tools for calculations at tree- and 1-loop level

In multiloop setups the package is in general not so useful

The idea to substantially improve on this matured during my work on QCD Energy-energy correlations [Dixon, Luo, VS, Yang and Zhu 2018; Luo, VS, Yang and Zhu 2019; Gao, VS, Yang 2020] and NNLO QCD corrections to $B$-meson mixing [Gerlach, Nierste, VS, Steinhauser, 2021]

This talk covers two aspects of the ongoing work in this direction

- Identification and mapping of loop integral topologies
- Handling of master integrals

FeynCalc 10 featuring this functionality will be released in 2022
**Motivation: Why multiloop with FeynCalc?**

**FeynCalc** offers a toolbox-oriented approach to symbolic Feynman diagram calculations.

- Not foolproof: correctness of the results $\propto$ user's understanding of QFT
- Very useful for people who know what they want to calculate

**FeynCalc** has plenty of tools for calculations at tree- and 1-loop level.

- In multiloop setups the package is in general not so useful

The idea to substantially improve on this matured during my work on QCD Energy-energy correlations [Dixon, Luo, VS, Yang and Zhu 2018; Luo, VS, Yang and Zhu 2019; Gao, VS, Yang 2020] and NNLO QCD corrections to $B$-meson mixing [Gerlach, Nierste, VS, Steinhauser, 2021].

This talk covers two aspects of the ongoing work in this direction:
- Identification and mapping of loop integral topologies
- Handling of master integrals

**FeynCalc** 10 featuring this functionality will be released in 2022.

---

1991 - **FeynCalc 1.0** [Mertig et al., 1991]
1997 - **TARCER** [Mertig & Scharf, 1998]
2012 - **FeynCalcFormLink** [Feng & Mertig, 2012]
2016 - **FeynCalc 9.0** [VS, Mertig, Orellana, 2016]
2017 - **FeynHelpers** [VS, 2016]
2020 - **FeynCalc 9.3** [VS, Mertig, Orellana, 2020]
2020 - **FeynOnium** [Brambilla, Chung, VS, Vairo 2020]
**Motivation: Why multiloop with FeynCalc?**

- **FeynCalc** offers a toolbox-oriented approach to symbolic Feynman diagram calculations
- Not foolproof: correctness of the results $\propto$ user's understanding of QFT
- Very useful for people who know what they want to calculate
- **FeynCalc** has plenty of tools for calculations at tree- and 1-loop level
- In multiloop setups the package is in general not so useful
- The idea to substantially improve on this matured during my work on QCD Energy-energy correlations [Dixon, Luo, VS, Yang and Zhu 2018; Luo, VS, Yang and Zhu 2019; Gao, VS, Yang 2020] and NNLO QCD corrections to $B$-meson mixing [Gerlach, Nierste, VS, Steinhauser, 2021]
- This talk covers two aspects of the ongoing work in this direction
  - Identification and mapping of loop integral topologies
  - Handling of master integrals
- **FeynCalc** 10 featuring this functionality will be released in 2022

---

**Timeline of FeynCalc versions**

- **1991** • **FeynCalc 1.0** [Mertig et al., 1991]
- **1997** • **TARCER** [Mertig & Scharf, 1998]
- **2012** • **FeynCalcFormLink** [Feng & Mertig, 2012]
- **2016** • **FeynCalc 9.0** [VS, Mertig, Orellana, 2016]
- **2017** • **FeynHelpers** [VS, 2016]
- **2020** • **FeynCalc 9.3** [VS, Mertig, Orellana, 2020]
- **2020** • **FeynOnium** [Brambilla, Chung, VS, Vairo 2020]
Symanzik polynomials and Pak algorithm
Feynman parametric representation of an $L$-loop scalar Minkowskian integral

\[
\left( \frac{e^{\varepsilon \gamma E}}{i\pi^{d/2}} \right)^L \int \frac{\left( \prod_{i=1}^L d^d k_i \right)}{P_1^{m_1} \ldots P_N^{m_N}} = (-1)^N m \left( m \frac{N - Ld}{2} \right) \int_0^\infty \prod_{j=1}^N dx_j x_j^{m_j - 1} \delta \left( 1 - \sum_{i=1}^N x_i \right) \frac{\mathcal{U}^{N_m - \frac{(L+1)d}{2}}}{\mathcal{F}^{N_m - \frac{Ld}{2}}}
\]

with $N$ quadratic/eikonal propagators $P_i$ and $N_m = \sum_{i=1}^N m_i$ with $m_i \geq 0$

- Properties of the integral encoded in the Symanzik polynomials $\mathcal{U}$ and $\mathcal{F}$ (nice summary in [Bogner & Weinzierl, 2010])
- Some combination of $(\mathcal{U}, \mathcal{F})$ and $m_i$ to characterize the given loop integral topology?
- Use this to find mappings between different topologies?
- In principle, yes! But things are not so simple ...
**Feynman parametric representation of an $L$-loop scalar Minkowskian integral**

$$(e^{\gamma E}/i\pi^{d/2})^L \int \frac{\prod_{i=1}^{L} d^dk_i}{P_1^{m_1} \ldots P_N^{m_N}} = (-1)^N m \Gamma(N_m - \frac{Ld}{2}) \int_0^{\infty} \prod_{j=1}^{N} dx_j x_j^{m_j - 1} \delta \left( 1 - \sum_{i=1}^{N} x_i \right) \frac{U^{N_m - \frac{(L+1)d}{2}}}{F^{N_m - \frac{Ld}{2}}}$$

with $N$ quadratic/eikonal propagators $P_i$ and $N_m = \sum_{i=1}^{N} m_i$ with $m_i \geq 0$

**Properties of the integral encoded in the Symanzik polynomials $U$ and $F$** (nice summary in [Bogner & Weinzierl, 2010])

- Some combination of $(U, F)$ and $m_i$ to characterize the given loop integral topology?
- Use this to find mappings between different topologies?
- In principle, yes! But things are not so simple ...

V. Shtabovenko (KIT) / ACAT 2021, 01.12.2021

FeynCalc goes multiloop
Feynman parametric representation of an $L$-loop scalar Minkowskian integral

$$\left(\frac{e^{\varepsilon \gamma E}}{i \pi^{d/2}}\right)^L \int \frac{\left(\prod_{i=1}^{L} d^d k_i\right)}{P_1^{m_1} \ldots P_N^{m_N}} = (-1)^{N_m} \frac{\Gamma(N_m - \frac{L d}{2})}{\prod_{j=1}^{N} \Gamma(m_j)} \int_0^\infty \prod_{j=1}^{N} dx_j x_j^{m_j - 1} \delta \left(1 - \sum_{i=1}^{N} x_i\right) \frac{\mathcal{U}^{N_m - \frac{(L+1)d}{2}}}{\mathcal{F}^{N_m - \frac{L d}{2}}}$$

with $N$ quadratic/eikonal propagators $P_i$ and $N_m = \sum_{i=1}^{N} m_i$ with $m_i \geq 0$

- **Properties of the integral encoded in the Symanzik polynomials $\mathcal{U}$ and $\mathcal{F}$** (nice summary in [Bogner & Weinzierl, 2010])
- Some combination of $(\mathcal{U}, \mathcal{F})$ and $m_i$ to characterize the given loop integral topology?
- Use this to find mappings between different topologies?
- In principle, yes! But things are not so simple ...
Denote \((U, F)\) as the characteristic polynomial \(P\)

Popular choices: \(P = U \times F\) or \(P = U + F\)

\(P\) depends on the Feynman parameters \(x_i\) and is not unique!

A new \(P'\) from \(P\) by permuting \(x_i\) (e.g. \(x_1 \leftrightarrow x_5, x_3 \leftrightarrow x_7\)) still describes the same loop integral

Enumerating all \(x_i\) permutations by brute force highly impractical!

Need to find some canonical ordering of the Feynman parameters \(x_i\) in the given \(P\)

Possible solution: Algorithm invented by Alexey Pak [Pak, 2012]
Denote \((U, F)\) as the characteristic polynomial \(P\)

- Popular choices: \(P = U \times F\) or \(P = U + F\)
- \(P\) depends on the Feynman parameters \(x_i\) and is not unique!
- A new \(P'\) from \(P\) by permuting \(x_i\) (e.g. \(x_1 \leftrightarrow x_5, x_3 \leftrightarrow x_7\)) still describes the same loop integral
- Enumerating all \(x_i\) permutations by brute force highly impractical!
- Need to find some canonical ordering of the Feynman parameters \(x_i\) in the given \(P\)
- Possible solution: Algorithm invented by Alexey Pak [Pak, 2012]
Denote \((U, F)\) as the characteristic polynomial \(P\)

Popular choices: \(P = U \times F\) or \(P = U + F\)

\(P\) depends on the Feynman parameters \(x_i\) and is not unique!

A new \(P'\) from \(P\) by permuting \(x_i\) (e.g., \(x_1 \leftrightarrow x_5, x_3 \leftrightarrow x_7\)) still describes the same loop integral

Enumerating all \(x_i\) permutations by brute force highly impractical!

Need to find some \textit{canonical ordering} of the Feynman parameters \(x_i\) in the given \(P\)

Possible solution: Algorithm invented by Alexey Pak [Pak, 2012]
Denote \((\mathcal{U}, \mathcal{F})\) as the characteristic polynomial \(P\)

- Popular choices: \(P = \mathcal{U} \times \mathcal{F}\) or \(P = \mathcal{U} + \mathcal{F}\)
- \(P\) depends on the Feynman parameters \(x_i\) and is not unique!
- A new \(P'\) from \(P\) by permuting \(x_i\) (e.g. \(x_1 \leftrightarrow x_5, x_3 \leftrightarrow x_7\)) still describes the same loop integral
- Enumerating all \(x_i\) permutations by brute force highly impractical!
- Need to find some *canonical ordering* of the Feynman parameters \(x_i\) in the given \(P\)

Possible solution: Algorithm invented by Alexey Pak \(\text{[Pak, 2012]}\)
Symanzik polynomials and Pak algorithm

- Rough idea: Write $P$ as a matrix, find the canonical form by swapping/sorting rows and columns
- Pak algorithm: canonical ordering of $x_i$ + symmetries between the corresponding lines.
- Very detailed description in the PhD thesis of Jens Hoff [Hoff, 2015]
- Technical implementation in Mathematica straightforward
  - Automatic calculation of $\mathcal{U} + \mathcal{F}$ in UF.m (now part of FIESTA [Smirnov et al., 2021] and FIRE [Smirnov & Chuharev, 2020])
  - Many of Pak’s ideas implemented in TopoID [Hoff, 2016], https://github.com/thejensemann/TopoID
Using **FeynCalc** in multiloop calculations
Using FeynCalc in multiloop calculations

Two new **FeynCalc** symbols for multiloop calculations

- **FCTopology[id,{propagators}, {loop momenta}, {external momenta}, {kinematics}, {}]**
  - denotes a loop integral family id

- **GLI[id,{propagator_powers}]**
  - is a loop integral belonging to the integral family id

Syntax inspired by **FIRE** and **LiteRed** [Lee, 2014], yet there are important differences

- **FCTopology**'s are local, can simultaneously work with multiple topologies and/or modify them on the fly
- No global list of topologies known in the current **Mathematica** session
- Relevant functions usually take 2 arguments: a list of GLI's and a list of FCTopology's

In addition to that, dozens of new functions that work with GLI's and FCTopology's:

- Naming scheme: **FCLoopXYZ**
- Most new functions also work with integrals in the (S)FAD-notation
Using FeynCalc in multiloop calculations

Two new FeynCalc symbols for multiloop calculations

FCTopology[id,{propagators}, {loop momenta}, {external momenta}, {kinematics}, {}] denotes a loop integral family id

GLI[id,{propagator powers}] is a loop integral belonging to the integral family id

Syntax inspired by FIRE and LiteRed [Lee, 2014], yet there are important differences

- FCTopology's are local, can simultaneously work with multiple topologies and/or modify them on the fly
- No global list of topologies known in the current Mathematica session
- Relevant functions usually take 2 arguments: a list of GLI's and a list of FCTopology's

In addition to that, dozens of new functions that work with GLI's and FCTopology's:

- Naming scheme: FCLoopXYZ
- Most new functions also work with integrals in the (S)FAD-notation
Two new **FeynCalc** symbols for multiloop calculations

- **FCTopology[id,{propagators}, {loop momenta}, {external momenta}, {kinematics}, {}]**
  
  denotes a loop integral family id

- **GLI[id,{propagator powers}]**
  
  is a loop integral belonging to the integral family id

Syntax inspired by **FIRE** and **LiteRed** [Lee, 2014], yet there are important differences

- FCTopology's are local, can simultaneously work with multiple topologies and/or modify them on the fly
- No global list of topologies known in the current **Mathematica** session
- Relevant functions usually take 2 arguments: a list of GLI's and a list of FCTopology's

In addition to that, dozens of new functions that work with GLI's and FCTopology's:

- Naming scheme: FCLoopXYZ
- Most new functions also work with integrals in the (S)FAD-notation
Two new **FeynCalc** symbols for multiloop calculations

**FCTopology**[id, {propagators}, {loop momenta}, {external momenta}, {kinematics}, {}]

denotes a loop integral family id

**GLI**[id, {propagator powers}]

is a loop integral belonging to the integral family id

Syntax inspired by **FIRE** and **LiteRed** [Lee, 2014], yet there are important differences

- **FCTopology**'s are local, can simultaneously work with multiple topologies and/or modify them on the fly
- No global list of topologies known in the current **Mathematica** session
- Relevant functions usually take 2 arguments: a list of GLI's and a list of FCTopology's

In addition to that, dozens of new functions that work with GLI's and FCTopology's:

- Naming scheme: FCLoopXYZ
- Most new functions also work with integrals in the (S)FAD-notation
Example: Integral $G(1, 1, 0, 1, 1)$ from the family of fully massive 2-loop on-shell propagators with $q^2 = m_1^2$

First we need to define the topology (call it prop2L)

```math
In[1]:= topo = FCTopology["prop2L", 
      {FAD[{p1, m1}], FAD[{p2, m2}], FAD[{p1 + q + m3}], FAD[{p2 + q + m4}], FAD[{p1 - p2, m5}]},
      {p1, p2}, {q}, {SPD[q] -> m1^2}], {});
```

The integral is just a symbol that doesn't do anything

```math
In[2]:= GLI["prop2L", {1, 1, 0, 1, 1}]
Out[2]= G_{prop2L}(1,1,0,1,1)
```

Simplest manipulation: convert a GLI symbol to the (S)FAD-notation

```math
In[3]:= FCLoopFromGLI[GLI["prop2L", {1, 1, 0, 1, 1}], {topo}]
Out[3]= \frac{1}{(p_1^2-m_1^2)(p_2^2-m_2^2)((p_2+q)^2-m_4^2)((p_1-p_2)^2-m_5^2)}
```
Example: Integral $G(1, 1, 0, 1, 1)$ from the family of fully massive 2-loop on-shell propagators with $q^2 = m_1^2$

First we need to define the topology (call it prop2L)

```math
In[1]:= topo = FCTopology[prop2L, {FAD[{p1, m1}], FAD[{p2, m2}], FAD[{p1 + q, m3}], FAD[{p2 + q, m4}], FAD[{p1 - p2, m5}]}, {p1, p2}, {q}, {SPD[q] -> m1^2}, {}];
```

The integral is just a symbol that doesn't do anything

```math
In[2]:= GLI[prop2L, {1, 1, 0, 1, 1}]
```

```math
Out[2]= G^{prop2L}(1,0,1,1)
```

Simplest manipulation: convert a GLI symbol to the (S)FAD-notation

```math
In[3]:= FCLoopFromGLI[GLI[prop2L, {1, 1, 0, 1, 1}], {topo}]
```

```math
Out[3]= \frac{1}{(p1^2-m1^2) (p2^2-m2^2) ((p2+q)^2-m4^2) ((p1-p2)^2-m5^2)}
```
Topology identification
Coming back to the question of topology identification and canonical ordering ...

How do we get the $U$ and $F$ polynomials of the given integral?

Example: 2-loop massive tadpole (here we use the (S)FAD-notation)

$U$ and $F$: first two entries in the list returned by FCFeynmanPrepare

\[
\text{In}[4] = \text{FCFeynmanPrepare}[\text{SFAD}[\{p1, m1^2\}, \{p2, m2^2\}, \{p1 - p2, m3^2\}], \{p1, p2\}, \text{Names} \rightarrow x][1;; 2] \] // TableForm

\[
\text{Out}[4]//\text{TableForm} = \\
x(1) x(2) + x(3) x(2) + x(1) x(3) \\
(x(1) x(2) + x(3) x(2) + x(1) x(3)) (m1^2 x(1) + m2^2 x(3) + m3^2 x(2))
\]

FCFeynmanPrepare also returns other building blocks

- propagator powers
- $M$ from $U = \det M$
- $J$ and $Q^\mu$ from $F = \det M (QM^{-1}Q - J)$
Coming back to the question of topology identification and canonical ordering ...

How do we get the $\mathcal{U}$ and $\mathcal{F}$ polynomials of the given integral?

Example: 2-loop massive tadpole (here we use the $(S)FAD$-notation)

$\mathcal{U}$ and $\mathcal{F}$: first two entries in the list returned by FCFeynmanPrepare

```
In[5]:= FCFeynmanPrepare[SFAD[{p1, m1^2}, {p2, m2^2}, {p1 - p2, m3^2}], {p1, p2}, Names -> x][[1 ;; 2]] // TableForm
```

```
Out[5]//TableForm=
x(1) x(2)+x(3) x(2)+x(1) x(3)
(x(1) x(2)+x(3) x(2)+x(1) x(3)) (m1^2 x(1)+m2^2 x(3)+m3^2 x(2))
```

FCFeynmanPrepare also returns other building blocks

- propagator powers
- $M$ from $\mathcal{U} = \det M$
- $J$ and $Q^\mu$ from $\mathcal{F} = \det M (QM^{-1}Q - J)$
Coming back to the question of topology identification and canonical ordering ...

How do we get the $U$ and $F$ polynomials of the given integral?

Example: 2-loop massive tadpole (here we use the (S)FAD-notation)

**$U$ and $F$: first two entries in the list returned by FCFeynmanPrepare**

\[
\text{In}[6]= \quad \text{FCFeynmanPrepare}[(\text{SFAD}[\{p_1, m_1^2\}, \{p_2, m_2^2\}, \{p_1 - p_2, m_3^2\}], \{p_1, p_2\}, \text{Names} \rightarrow x][1 ;; 2] \quad // \quad \text{TableForm}
\]

\[
\text{Out}[6]//\text{TableForm}=
\begin{align*}
x(1) x(2)+x(3) x(2)+x(1) x(3) \\
(x(1) x(2)+x(3) x(2)+x(1) x(3)) (m_1^2 x(1)+m_2^2 x(3)+m_3^2 x(2))
\end{align*}
\]

FCFeynmanPrepare also returns other building blocks

- propagator powers
- $M$ from $U = \det M$
- $J$ and $Q^\mu$ from $F = \det (Q M^{-1} Q - J)$
Using FeynCalc in multiloop calculations

Topology identification

Coming back to the question of topology identification and canonical ordering ...

How do we get the $U$ and $F$ polynomials of the given integral?

Example: 2-loop massive tadpole (here we use the (S)FAD-notation)

$U$ and $F$: first two entries in the list returned by FCFeynmanPrepare

```plaintext
In[7]:= FCFeynmanPrepare[SFAD[{{p1, m1^2}, {p2, m2^2}, {p1 - p2, m3^2}}, {p1, p2}, Names -> x]][[1 ;; 2]] // TableForm
```

```
Out[7]//TableForm= x(1) x(2)+x(3) x(2)+x(1) x(3)
(x(1) x(2)+x(3) x(2)+x(1) x(3)) (m1^2 x(1)+m2^2 x(3)+m3^2 x(2))
```

FCFeynmanPrepare also returns other building blocks

- propagator powers
- $M$ from $U = \det M$
- $J$ and $Q^\mu$ from $F = \det M (QM^{-1}Q - J)$
What about the characteristic polynomial $P = U \times F$?

Same specimen, the 2-loop tadpole

$$\text{In}[8] = \text{FCLoopToPakForm}[\text{FAD}[[\{p1, m1\}, \{p2, m2\}, \{p1 - p2, m3\}]], \{p1, p2\}, \text{Names} \rightarrow x][[2]][[1]]$$

$$\text{Out}[8] = m1^2 x(1)^2 x(2)+m1^2 x(1)^2 x(3)+m1^2 x(1) x(2) x(3)+m2^2 x(1)^2 x(2)^2 +m2^2 x(1) x(2) x(3)+m3^2 x(1) x(3)^2 +m3^2 x(2) x(3)^2 +m3^2 x(1) x(2) x(3)+x(1)^2 x(2)+x(1) x(3)+x(2) x(3)$$

The output is already canonically ordered

Each $x_i$ corresponds to one of the propagators (the function keeps track of that)

Want to work canonicalize the given polynomial?

Use FCLoopPakOrder

$$\text{In}[9] = \text{poly} = -1/4*(x(2)^2*x(3)) - (x(1)^2*x(4))/4 - (x(1)^2*x(5))/4 + (x(1)*x(2)*x(5))/2 - (x(2)^2*x(5))/4 + x(3)*x(4)*x(5);$$

$$\text{Out}[9] = -1/4 x(4) x(1)^2 - 1/4 x(5) x(1)^2 + 1/2 x(2) x(5) x(1) - 1/4 x(2)^2 x(3) - 1/4 x(2)^2 x(5) + x(3) x(4) x(5)$$

$$\text{In}[10] = \text{FCLoopPakOrder}[\text{poly}, x, \text{Rename} \rightarrow \text{True}]$$

$$\text{Out}[10] = -1/4 x(3) x(1)^2 - 1/4 x(5) x(1)^2 + 1/2 x(2) x(3) x(1) - 1/4 x(2)^2 x(3) - 1/4 x(2)^2 x(4) + x(3) x(4) x(5)$$
What about the characteristic polynomial $P = U \times F$?

Same specimen, the 2-loop tadpole

```
In[11]= FCLoopToPakForm[FAD[{p1, m1}, {p2, m2}, {p1 - p2, m3}], {p1, p2}, Names -> x][[2]][[1]]
```

```
Out[11]= -m1^2 x(1)^2 x(2) + m1^2 x(1)^2 x(3) + m1^2 x(1) x(2) x(3) + 2 m2^2 x(1) x(2)^2 + m2^2 x(2) x(3) - m3^2 x(1) x(3)^2 + m3^2 x(2) x(3)^2 + m3^2 x(1) x(2) x(3) + x(1) x(2) + x(1) x(3) + x(2) x(3)
```

The output is already canonically ordered

Each $x_i$ corresponds to one of the propagators (the function keeps track of that)

Want to work canonicalize the given polynomial?

Use FCLoopPakOrder

```
```

```
Out[12]= -1/4 x(4) x(1)^2 - 1/4 x(5) x(1)^2 + 1/2 x(2) x(5) x(1) - 1/4 x(2)^2 x(3) - 1/4 x(2)^2 x(5) + x(3) x(4) x(5)
```

```
In[13]= FCLoopPakOrder[poly, x, Rename -> True]
```

```
Out[13]= -1/4 x(3) x(1)^2 - 1/4 x(5) x(1)^2 + 1/2 x(2) x(3) x(1) - 1/4 x(2)^2 x(3) - 1/4 x(2)^2 x(4) + x(3) x(4) x(5)
```
What about the characteristic polynomial $P = U \times F$?

Same specimen, the 2-loop tadpole

\[\text{In}[14] := \ \text{FCLoopToPakForm}[\text{FAD}[\{p1, m1\}, \{p2, m2\}, \{p1 - p2, m3\}], \{p1, p2\}, \text{Names} \rightarrow x][[2]][[1]]\]

\[\text{Out}[14] := m1^2 x(1)^2 x(2) + m2^2 x(1)^2 x(3) + m2^2 x(1) x(2)^2 + m2^2 x(2)^2 x(3) + m3^2 x(1) x(3)^2 + m3^2 x(2) x(3)^2 + m3^2 x(1) x(2) x(3) + x(1) x(2) + x(1) x(3) + x(2) x(3)\]

The output is already canonically ordered

Each $x_i$ corresponds to one of the propagators (the function keeps track of that)

Want to work canonicalize the given polynomial?

Use FCLoopPakOrder


\[\text{Out}[15] := -\frac{1}{4} x(4) x(1)^2 - \frac{1}{4} x(5) x(1)^2 - \frac{1}{2} x(2) x(5) x(1) - \frac{1}{4} x(2)^2 x(3) - \frac{1}{4} x(2)^2 x(5) + x(3) x(4) x(5)\]

\[\text{In}[16] := \ \text{FCLoopPakOrder}[\text{poly}, x, \text{Rename} \rightarrow \text{True}]\]

\[\text{Out}[16] := -\frac{1}{4} x(3) x(1)^2 - \frac{1}{4} x(5) x(1)^2 - \frac{1}{2} x(2) x(3) x(1) - \frac{1}{4} x(2)^2 x(3) - \frac{1}{4} x(2)^2 x(4) + x(3) x(4) x(5)\]
What about the characteristic polynomial $P = U \times F$?

Same specimen, the 2-loop tadpole

```math
In[17]:= FCLoopToPakForm[FAD[{p1, m1}, {p2, m2}, {p1 - p2, m3}], {p1, p2}, Names -> x][[2]][[1]]
```

```math
Out[17]= m1^2 x(1)^2 x(2)+m2^2 x(3)+m^2 x(1) x(2) x(3)+m2^2 x(2)^2 x(3)+m^2 x(1) x(2) x(3)+m3^2 x(1) x(3)^2+m3^2 x(2) x(3)^2+m3^2 x(1) x(2) x(3)+x(1) x(2)+x(1) x(3)+x(2) x(3)
```

The output is already canonically ordered

- Each $x_i$ corresponds to one of the propagators (the function keeps track of that)
- Want to work canonicalize the given polynomial?

Use `FCLoopPakOrder`

```math
```

```math
Out[18]= \frac{1}{4} x(4) x(1)^2 - \frac{1}{4} x(5) x(1)^2 + \frac{1}{2} x(2) x(5) x(1) - \frac{1}{4} x(2)^2 x(3) - \frac{1}{4} x(2)^2 x(5) + x(3) x(4) x(5)
```

```math
In[19]:= FCLoopPakOrder[poly, x, Rename -> True]
```

```math
Out[19]= -\frac{1}{4} x(3) x(1)^2 - \frac{1}{4} x(5) x(1)^2 + \frac{1}{2} x(2) x(3) x(1) - \frac{1}{4} x(2)^2 x(3) - \frac{1}{4} x(2)^2 x(4) + x(3) x(4) x(5)
```
What about the characteristic polynomial $P = \mathcal{U} \times \mathcal{F}$?

Same specimen, the 2-loop tadpole

```math
\begin{align*}
|\text{In[20]}| := & \quad \text{FCLoopToPakForm[FAD[\{p1, m1\}, \{p2, m2\}, \{p1 - p2, m3\}], \{p1, p2\}, Names -> x]][2][1]} \\
|\text{Out[20]}| := & \quad m^2 x(1)^2 x(2) + m^2 x(1)^2 x(3) + m^2 x(1) x(2) x(3) + m^2 x(2)^2 x(3) + m^2 x(1) x(2) x(3) + m^2 x(1) x(2)^2 + m^2 x(1) x(2) + x(1) x(2) + x(1) x(3) + x(2) x(3)
\end{align*}
```

The output is already canonically ordered

Each $x_i$ corresponds to one of the propagators (the function keeps track of that)

Want to work canonicalize the given polynomial?

Use FCLoopPakOrder

```math
\begin{align*}
|\text{In[21]}| := & \quad \text{poly} = -1/4 \times (x[2] \times x[3]) - (x[1] \times 2 \times x[4]) / 4 - (x[1] \times x[2] \times x[5]) / 4 + (x[1] \times x[2] \times x[5]) / 2 - (x[2] \times 2 \times x[5]) / 4 + x[3] \times x[4] \times x[5]; \\
|\text{Out[21]}| := & \quad -\frac{1}{4} x(4) x(1)^2 - \frac{1}{4} x(5) x(1)^2 + \frac{1}{2} x(2) x(5) x(1) - \frac{1}{4} x(2)^2 x(3) - \frac{1}{4} x(2)^2 x(5) + x(3) x(4) x(5)
\end{align*}
```

```math
\begin{align*}
|\text{In[22]}| := & \quad \text{FCLoopPakOrder[poly, x, Rename -> True]} \\
|\text{Out[22]}| := & \quad -\frac{1}{4} x(3) x(1)^2 - \frac{1}{4} x(5) x(1)^2 + \frac{1}{2} x(2) x(3) x(1) - \frac{1}{4} x(2)^2 x(3) - \frac{1}{4} x(2)^2 x(4) + x(3) x(4) x(5)
\end{align*}
```
Suppose that we have a set of source topologies that can be mapped into a set of target topologies.

Working with amplitudes, not graphs: need explicit momentum shifts that describe these mappings.

Example: 3-loop propagator-type massless integrals, 2 source and 1 target topologies:

```
In[23]:= source = {FCTopology[topo1, {FAD[p1], FAD[p2], FAD[p3], FAD[p1 + p3], FAD[p2 + p3], FAD[p2 - Q], FAD[p1 + p3 - Q], FAD[p2 + p3 - Q], FAD[p1 + p2 + p3 - Q]}, {p1, p2, p3}, {Q}, {}, {}],
                FCTopology[topo2, {FAD[p1], FAD[p2], FAD[p3], FAD[p1 + p3], FAD[p2 + p3], FAD[p1 - Q], FAD[p2 - Q],
                                 FAD[p1 + p3 - Q], FAD[p1 + p2 + p3 - Q]}, {p1, p2, p3}, {Q}, {}, {}}}

Out[23]= {FCTopology(topo1,{ 1
                                 p1^2, p2^2, p3^2, (p1+p3)^2, (p2+p3)^2, (p2-Q)^2, (p1+p3-Q)^2, (p1+p2+p3-Q)^2 },{p1,p2,p3},{Q},{},{}),
           FCTopology(topo2,{ 1
                                 p1^2, p2^2, p3^2, (p1+p3)^2, (p2+p3)^2, (p1-Q)^2, (p2-Q)^2, (p1+p3-Q)^2, (p1+p2+p3-Q)^2 },{p1,p2,p3},{Q},{},{}))}
```

```
In[24]:= target = {FCTopology[prop3L, {FAD[p1], FAD[p2], FAD[p3], FAD[p2 + p3], FAD[p1 - Q], FAD[p2 - Q],
                                 FAD[p1 + p3 - Q], FAD[p2 + p3 - Q], FAD[p1 + p2 + p3 - Q]}, {p1, p2, p3}, {Q}, {}, {}}}

Out[24]= {FCTopology(prop3L,{ 1
                                 p1^2, p2^2, p3^2, (p2+p3)^2, (p1-Q)^2, (p2-Q)^2, (p1+p3-Q)^2, (p2+p3-Q)^2, (p1+p2+p3-Q)^2 },{p1,p2,p3},{Q},{},{}))}
```

We can get the mappings to the target topology with just one command:

```
In[25]:= mappings = FCLoopFindTopologyMappings[Join[source, target], PreferredTopologies -> prop3L]
```
Suppose that we have a set of source topologies that can be mapped into a set of target topologies.

Working with amplitudes, not graphs: need explicit momentum shifts that describe these mappings.

Example: 3-loop propagator-type massless integrals, 2 source and 1 target topologies:

```math
\text{In[26]} := \text{source} = \{\text{FCTopology[topo1, \{FAD[p1], FAD[p2], FAD[p3], FAD[p1+p3], FAD[p2+p3], FAD[p2-Q], FAD[p1+p3-Q], FAD[p2+p3-Q], FAD[p1+p2+p3-Q]\}, \{p1, p2, p3\}, \{Q\}, \{}\}\},
\text{FCTopology[topo2, \{FAD[p1], FAD[p2], FAD[p3], FAD[p1+p3], FAD[p2+p3], FAD[p1-Q], FAD[p2-Q], FAD[p1+p3-Q], FAD[p1+2+p3-Q], FAD[p2-Q], FAD[p1+2+p3-Q]\}, \{p1, p2, p3\}, \{Q\}, \{}\}\}
```

```math
\text{Out[26]} = \{\text{FCTopology[topo1, \{1, 1, 1, 1, 1, 1, 1, 1\}, \{p1, p2, p3\}, \{Q\}, \{}\}\},
\text{FCTopology[topo2, \{1, 1, 1, 1, 1, 1, 1, 1\}, \{p1, p2, p3\}, \{Q\}, \{}\}\}
```

We can get the mappings to the target topology with just one command:

```math
\text{In[27]} := \text{target} = \{\text{FCTopology[prop3L, \{FAD[p1], FAD[p2], FAD[p3], FAD[p2+p3], FAD[p1-Q], FAD[p2-Q], FAD[p1+p3-Q], FAD[p1+2+p3-Q]\}, \{p1, p2, p3\}, \{Q\}, \{}\}\}
```

```math
\text{Out[27]} = \{\text{FCTopology[prop3L, \{1, 1, 1, 1, 1, 1, 1, 1\}, \{p1, p2, p3\}, \{Q\}, \{}\}\}
```

We can get the mappings to the target topology with just one command:

```math
\text{In[28]} := \text{mappings} = \text{FCLoopFindTopologyMappings[Join[source, target], PreferredTopologies -> prop3L]}
```
Suppose that we have a set of source topologies that can be mapped into a set of target topologies.

Working with amplitudes, not graphs: need explicit momentum shifts that describe these mappings.

Example: 3-loop propagator-type massless integrals, 2 source and 1 target topologies.

```mathematica
In[29]:=
source = {FCTopology[topo1, {FAD[p1], FAD[p2], FAD[p3], FAD[p1 + p3], FAD[p2 + p3], FAD[p2 - Q], FAD[p1 + p3 - Q], FAD[p2 + p3 - Q], FAD[p1 + p2 + p3 - Q]}, {p1, p2, p3}, {Q}, {}, {}],
          FCTopology[topo2, {FAD[p1], FAD[p2], FAD[p3], FAD[p1 + p3], FAD[p2 + p3], FAD[p1 - Q], FAD[p2 - Q], FAD[p1 + p3 - Q], FAD[p1 + p2 + p3 - Q]}, {p1, p2, p3}, {Q}, {}, {}}};

Out[29]= {FCTopology[topo1, {{1, 1, 1, 1, 1, 1, 1, 1}, {p1, p2, p3}, {Q}, {}, {}}],
          FCTopology[topo2, {{1, 1, 1, 1, 1, 1, 1, 1, 1}, {p1, p2, p3}, {Q}, {}, {}}]}
```

We can get the mappings to the target topology with just one command.

```mathematica
In[30]:= mappings = FCLoopFindTopologyMappings[Join[source, target], PreferredTopologies -> prop3L]
Out[30]= {FCTopology[prop3L, {{1, 1, 1, 1, 1, 1, 1, 1, 1}, {p1, p2, p3}, {Q}, {}, {}}]}
```

We can get the mappings to the target topology with just one command.

```mathematica
In[31]:= mappings = FCLoopFindTopologyMappings[Join[source, target], PreferredTopologies -> prop3L]
Out[31]= {FCTopology[prop3L, {{1, 1, 1, 1, 1, 1, 1, 1, 1}, {p1, p2, p3}, {Q}, {}, {}}]}
```
Let us examine the output of

\[
\text{In[32]:= } \text{mappings = FCLoopFindTopologyMappings[Join[source, target], PreferredTopologies -> \{prop3L\}]}\]

in more details

- The first entry maps \text{topo1} to \text{prop3L}

\[
\text{In[33]:= } \text{mappings[[1]]}
\]

\[
\text{Out[33]= } \{\text{FCTopology(topo1,\{1/p_1^2, 1/p_2^2, 1/p_3^2, (p_1+p_3)^2, (p_2+p_3)^2, (p_2-Q)^2, (p_1+p_3-Q)^2, (p_2+p_3-Q)^2\},\{p_1,p_2,p_3\},\{Q\},\{\}\}),}
\]

- Content: original source topology, a list of momenta shifts and a replacement rule for scalar integrals
- All information that you need to convert scalar and tensor integrals from \text{topo1} to integrals from \text{prop3L}!
Let us examine the output of

\[ \text{In[34]}:= \text{mappings} = \text{FCLoopFindTopologyMappings[Join[source, target], PreferredTopologies} \rightarrow \{\text{prop3L}\}] \]

in more details

The first entry maps `topo1` to `prop3L`

\[ \text{In[35]}:= \text{mappings[[1]]} \]

\[ \text{Out[35]}= \{\text{FCTopology}(\text{topo1},\{\frac{1}{p_1^2}, \frac{1}{p_2^2}, \frac{1}{p_3^2}, \frac{1}{(p_1+p_3)^2}, \frac{1}{(p_2+p_3)^2}, \frac{1}{(p_1+p_3-Q)^2}, \frac{1}{(p_2+p_3-Q)^2}, \frac{1}{(p_1+p_2+p_3-Q)^2}\},\{p_1,p_2,p_3\},\{Q\},\{\})), \{p_1 \rightarrow -p_1-p_3+Q, p_2 \rightarrow -p_2-p_3+Q, p_3 \rightarrow p_3\}, G^{\text{topo1}}(n_7,n_8,n_3,n_5,n_6,n_4,n_1,n_2,n_9) \rightarrow G^{\text{prop3L}}(n_1,n_2,n_3,n_4,n_5,n_6,n_7,n_8,n_9)\} \]

- Content: original source topology, a list of momenta shifts and a replacement rule for scalar integrals
- All information that you need to convert scalar and tensor integrals from `topo1` to integrals from `prop3L`
Let us examine the output of

\[ \text{mappings} = \text{FCLoopFindTopologyMappings}([\text{source, target}], \text{PreferredTopologies} \rightarrow \{\text{prop3L}\}) \]

in more details

The first entry maps \text{topo1} to \text{prop3L}

\[ \text{mappings}[[1]] \]

\[ \text{Out}[37] = \{\text{FCTopology}([1, 1, 1, 1, 1, 1, 1, 1], \{p1, p2, p3\}, \{Q\}, \{\}, \{\}), \{p1 \rightarrow -p1-p3+Q, p2 \rightarrow -p2-p3+Q, p3 \rightarrow p3\}, \text{G}^{\text{topo1}}(n7_, n8_, n3_, n5_, n6_, n4_, n1_, n2_, n9_) \rightarrow \text{G}^{\text{prop3L}}(n1, n2, n3, n4, n5, n6, n7, n8, n9)\} \]

Content: original source topology, a list of momenta shifts and a replacement rule for scalar integrals

All information that you need to convert scalar and tensor integrals from \text{topo1} to integrals from \text{prop3L}!
Applying this machinery to **amplitudes** is already possible, but somewhat cumbersome

- `FCLoopFindTopologies[exp, q1, q2, ...]` to identify distinct topologies in the expression
- `FCLoopFindTopologyMappings[topo1, topo2, ...]` to minimize the number of topologies
- `FCLoopApplyTopologyMappings[exp, mappings]` to apply the so-obtained mappings

Some aspects that require further attention/optimization

- Handling of tensor integrals
- Automatically augmenting incomplete topologies to fit them into existing (or new) complete topologies
- Performance optimizations

What we actually want is to calculate multiloop amplitudes with FORM, not **Mathematica**!

The goal is to have a hybrid framework: toolbox-like FORM library for heavy computations and a **Mathematica** library (**FeynCalc**) for everything else (e.g. topology identification, R&D, etc.)
Applying this machinery to **amplitudes** is already possible, but somewhat cumbersome

- `FCLoopFindTopologies[exp, q1, q2, ...]` to identify distinct topologies in the expression
- `FCLoopFindTopologyMappings[topo1, topo2, ...]` to minimize the number of topologies
- `FCLoopApplyTopologyMappings[exp, mappings]` to apply the so-obtained mappings

Some aspects that require further attention/optimization

- Handling of tensor integrals
- Automatically augmenting incomplete topologies to fit them into existing (or new) complete topologies
- Performance optimizations

What we actually want is to calculate multiloop amplitudes with FORM, not MATHEMATICA!

The goal is to have a hybrid framework: toolbox-like FORM library for heavy computations and a MATHEMATICA library (FeynCalc) for everything else (e.g. topology identification, R&D, etc.)
Using FeynCalc in multiloop calculations  

Topology identification

Applying this machinery to **amplitudes** is already possible, but somewhat cumbersome

- FCLoopFindTopologies[exp, q1, q2, ...] to identify distinct topologies in the expression
- FCLoopFindTopologyMappings[topo1, topo2, ...] to minimize the number of topologies
- FCLoopApplyTopologyMappings[exp, mappings] to apply the so-obtained mappings

Some aspects that require further attention/optimization

- Handling of tensor integrals
- Automatically augmenting incomplete topologies to fit them into existing (or new) complete topologies
- Performance optimizations

What we actually want is to calculate multiloop amplitudes with FORM, not Mathematica!

The goal is to have a hybrid framework: toolbox-like FORM library for heavy computations and a Mathematica library (FeynCalc) for everything else (e.g. topology identification, R&D, etc.)
Applying this machinery to amplitudes is already possible, but somewhat cumbersome
- FCLoopFindTopologies[exp, q1, q2, ...] to identify distinct topologies in the expression
- FCLoopFindTopologyMappings[topo1, topo2, ...] to minimize the number of topologies
- FCLoopApplyTopologyMappings[exp, mappings] to apply the so-obtained mappings

Some aspects that require further attention/optimization
- Handling of tensor integrals
- Automatically augmenting incomplete topologies to fit them into existing (or new) complete topologies
- Performance optimizations

What we actually want is to calculate multiloop amplitudes with FORM, not MATHEMATICA!

The goal is to have a hybrid framework: toolbox-like FORM library for heavy computations and a MATHEMATICA library (FeynCalc) for everything else (e. g. topology identification, R&D, etc.)
Master integrals
Suppose that you are done with the IBP-reduction [Chetyrkin & Tkachov, 1981; Tkachov, 1981] of your loop integrals, what are the next steps?

- Find mappings between master integrals from different integral families
- Visualize your unique master integrals
- Try to calculate some of them analytically?

Let us see how FeynCalc can help us in tackling those tasks!
Suppose that you are done with the IBP-reduction [Chetyrkin & Tkachov, 1981; Tkachov, 1981] of your loop integrals, what are the next steps?

- Find mappings between master integrals from different integral families
- Visualize your unique master integrals
- Try to calculate some of them analytically?

Let us see how **FeynCalc** can help us in tackling those tasks!
With Pak algorithm at our disposal finding mappings between master integrals is straightforward

Example: 2-loop self-energies with one or two massive lines

```
In[38]:= topos = {FCTopology[prop2Lv1, {FAD[p1], FAD[(p1 + q1, m)], FAD[p2], FAD[p2 + q1], FAD[p1 - p2]}, {p1, p2}, {q1}, {}, {}],
                  FCTopology[prop2Lv2, {FAD[p1], FAD[(p2, m)], FAD[(p2 + q1, m)], FAD[p1 + q1], FAD[-p1 + p2]}, {p1, p2}, {q1}, {}, {}]};
```

We have a list of master integrals that are not all unique

```
In[39]:= glis = {GLI[prop2Lv1, {0, 1, 1, 1, 1}], GLI[prop2Lv2, {0, 1, 1, 1, 1}], GLI[prop2Lv2, {1, 0, 1, 1, 1}],
                  GLI[prop2Lv2, {1, 1, 0, 1, 1}], GLI[prop2Lv2, {1, 1, 1, 0, 1}]};
```

**FeynCalc** identifies three mappings between our masters

```
In[40]:= FCLoopFindIntegralMappings[glis, topos][[1]]
```

```
Out[40]= {Gprop2Lv2(1,0,1,1,1) → Gprop2Lv1(0,1,1,1,1), Gprop2Lv2(1,1,0,1,1) → Gprop2Lv1(0,1,1,1,1), Gprop2Lv2(1,1,1,0,1) → Gprop2Lv2(0,1,1,1,1)}
```

Working principle based on FindRules from **FIRE**

Can specify a set of preferred master integrals via the PreferredIntegrals option
With Pak algorithm at our disposal finding mappings between master integrals is straightforward.

Example: 2-loop self-energies with one or two massive lines.

```
In[41]:=
  topos = {FCTopology[prop2Lv1, {FAD[p1], FAD[{p1 + q1, m}], FAD[p2], FAD[p2 + q1], FAD[p1 - p2]}, {p1, p2}, {q1}, {}, {}],
                     FCTopology[prop2Lv2, {FAD[p1], FAD[{p2, m}], FAD[{p2 + q1, m}], FAD[p1 + q1], FAD[-p1 + p2]}, {p1, p2}, {q1}, {}, {}]};
```

We have a list of master integrals that are not all unique.

```
In[42]:=
  glis = {GLI[prop2Lv1, {0, 1, 1, 1, 1}], GLI[prop2Lv2, {0, 1, 1, 1, 1}], GLI[prop2Lv2, {1, 0, 1, 1, 1}],
           GLI[prop2Lv2, {1, 1, 0, 1, 1}], GLI[prop2Lv2, {1, 1, 1, 0, 1}]};
```

```
FeynCalc` identifies three mappings between our masters.
```

```
In[43]:=
  FCLoopFindIntegralMappings[glis, topos][[1]]
```

```
Out[43]=
  {G\text{prop2Lv2}^{(1,0,1,1,1)} \rightarrow G\text{prop2Lv1}^{(0,1,1,1,1)}, G\text{prop2Lv2}^{(1,1,0,1,1)} \rightarrow G\text{prop2Lv1}^{(0,1,1,1,1)}, G\text{prop2Lv2}^{(1,1,1,0,1)} \rightarrow G\text{prop2Lv2}^{(0,1,1,1,1)}}
```

Working principle based on 

FindRules from FIRE.

Can specify a set of preferred master integrals via the PreferredIntegrals option.
With Pak algorithm at our disposal finding mappings between master integrals is straightforward.

Example: 2-loop self-energies with one or two massive lines

\[
\text{\includegraphics[width=0.5\textwidth]{diagram.png}}
\]

\[
\text{In}[44]:= \text{topos = \{FCTopology[prop2Lv1, \{FAD[p1], FAD[\{p1 + q1, m\}], FAD[p2], FAD[p2 + q1], FAD[p1 - p2], \{p1, p2\}, \{q1\}, \{\}, \{\}], FCTopology[prop2Lv2, \{FAD[p1], FAD[\{p2, m\}], FAD[\{p2 + q1, m\}], FAD[p1 + q1], FAD[-p1 + p2], \{p1, p2\}, \{q1\}, \{\}, \{\}]\};}
\]

We have a list of master integrals that are not all unique

\[
\text{In}[45]:= \text{glis = \{GLI[prop2Lv1, \{0, 1, 1, 1, 1\}], GLI[prop2Lv2, \{0, 1, 1, 1, 1\}], GLI[prop2Lv2, \{1, 0, 1, 1, 1\}], GLI[prop2Lv2, \{1, 1, 0, 1, 1\}], GLI[prop2Lv2, \{1, 1, 1, 0, 1\}]\};}
\]

\textbf{\textsc{FeynCalc}} identifies three mappings between our masters

\[
\text{In}[46]:= \text{FCLoopFindIntegralMappings[glis, topos][[1]]}
\]

\[
\text{Out}[46]= \{G^{\text{prop2Lv2}}(1,0,1,1,1) \rightarrow G^{\text{prop2Lv1}}(0,1,1,1,1), G^{\text{prop2Lv2}}(1,1,0,1,1) \rightarrow G^{\text{prop2Lv1}}(0,1,1,1,1), G^{\text{prop2Lv2}}(1,1,1,0,1) \rightarrow G^{\text{prop2Lv2}}(0,1,1,1,1)\}
\]

Working principle based on \text{FindRules} from \text{FIRE}

Can specify a set of preferred master integrals via the \text{PreferredIntegrals} option.
With Pak algorithm at our disposal finding mappings between master integrals is straightforward.

Example: 2-loop self-energies with one or two massive lines

\[
\begin{align*}
\text{In[47]:=} & \quad \text{topos} = \{\text{FCTopology[prop2Lv1, \{FAD[p1], FAD[\{p1 + q1, m\}], FAD[p2], FAD[p2 + q1], FAD[p1 - p2]\}, \{p1, p2\}, \{q1\}, \{\}, \{\}],}
\\ & \text{FCTopology[prop2Lv2, \{FAD[p1], FAD[\{p2, m\}], FAD[\{p2 + q1, m\}], FAD[p1 + q1], FAD[-p1 + p2]\}, \{p1, p2\}, \{q1\}, \{\}, \{\}]};
\end{align*}
\]

We have a list of master integrals that are not all unique.

\[
\text{In[48]:=} \quad \text{glis} = \{\text{GLI[prop2Lv1, \{0, 1, 1, 1, 1\}], GLI[prop2Lv2, \{0, 1, 1, 1\}], GLI[prop2Lv2, \{0, 1, 1, 1\}],}
\\ & \text{GLI[prop2Lv2, \{1, 0, 1, 1\}], GLI[prop2Lv2, \{1, 1, 0, 1\}], GLI[prop2Lv2, \{1, 1, 1, 0\}]}
\]

\textbf{FeynCalc} identifies three mappings between our masters.

\[
\text{In[49]:=} \quad \text{FCLoopFindIntegralMappings[glis, topos][[1]]}
\]

\[
\text{Out[49]=} \quad \{G^{\text{prop2Lv2}}(1,0,1,1,1) \rightarrow G^{\text{prop2Lv1}}(0,1,1,1,1), G^{\text{prop2Lv2}}(1,0,1,1,1) \rightarrow G^{\text{prop2Lv1}}(1,0,1,1,1), G^{\text{prop2Lv2}}(1,1,0,1) \rightarrow G^{\text{prop2Lv2}}(0,1,1,1,1)\}
\]

Working principle based on \texttt{FindRules} from \texttt{FIRE}.

Can specify a set of preferred master integrals via the \texttt{PreferredIntegrals} option.
Given a propagator representation of a loop integral, how to obtain its graph representation?

Can be done using **AZURITE** [Georgoudis et al., 2017], **PLANARITYTEST** [Bielas et al., 2013], **LITERED** and now also **FeynCalc**

Two-step approach: FCLoopIntegralToGraph reconstructs the graph, FCLoopGraphPlot plots it.

```plaintext
In[50]: = FCLoopIntegralToGraph[FAD[{p1, m1}, {p2, m2}, {Q1 + p1, m3}, Q2 - p1, Q1 + p1 + p2, Q2 - p1 - p2, Q2 + Q3 - p1 - p2}, {p1, p2}];
FCLoopGraphPlot[%]
```
Given a propagator representation of a loop integral, how to obtain its graph representation?

Can be done using **AZURITE** [Georgoudis et al., 2017], **PLANARITY TEST**, [Bielas et al., 2013], **LITE RED** and now also **FeynCalc**

Two-step approach: `FCLoopIntegralToGraph` reconstructs the graph, `FCLoopGraphPlot` plots it.

```math
In[51]:= FCLoopIntegralToGraph[FAD[{p1, m1}, {p2, m2}, {Q1 + p1, m3}, Q2 - p1, Q1 + p1 + p2, Q2 - p1 - p2, Q2 + Q3 - p1 - p2}, {p1, p2}];
FCLoopGraphPlot[%]
```
Given a propagator representation of a loop integral, how to obtain its graph representation?

Can be done using **AZURITE** [Georgoudis et al., 2017], **PLANARITYTest**, [Bielas et al., 2013], **LITERED** and now also **FeynCalc**

Two-step approach: `FCLoopIntegralToGraph` reconstructs the graph, `FCLoopGraphPlot` plots it.

```plaintext
In[52]:= FCLoopIntegralToGraph[FAD[{p1, m1}, {p2, m2}, {Q1 + p1, m3}, Q2 - p1, Q1 + p1 + p2, Q2 - p1 - p2, Q2 + Q3 - p1 - p2, {p1, p2}];
FCLoopGraphPlot[%]
```
Given a propagator representation of a loop integral, how to obtain its graph representation? Can be done using **AZURITE** [Georgoudis et al., 2017], **PLANARITYTEST** [Bielas et al., 2013], **LITERED** and now also **FeynCalc**

Two-step approach: FCLoopIntegralToGraph reconstructs the graph, FCLoopGraphPlot plots it.

```
In[53]:= FCLoopIntegralToGraph[FAD[{p1, m1}, {p2, m2}, {Q1 + p1, m3}, Q2 - p1, Q1 + p1 + p2, Q2 - p1 - p2, Q2 + Q3 - p1 - p2}, {p1, p2}]; FCLoopGraphPlot[%]
```
The plot can be styled to make it more visually appealing

\[
\text{In}[54]= \text{FCLoopIntegralToGraph}[\text{FAD}\{\{\text{p1}, \text{m1}\}, \{\text{p2}, \text{m2}\}, \{\text{Q1} + \text{p1}, \text{m3}\}, \text{Q2} - \text{p1}, \text{Q1} + \text{p1} + \text{p2}, \text{Q2} - \text{p1} - \text{p2}, \text{Q2} + \text{Q3} - \text{p1} - \text{p2}, \{\text{p1}, \text{p2}\}\}]; \\
\text{FCLoopGraphPlot}[\%], \text{GraphPlot} \rightarrow \{\text{MultiedgeStyle} \rightarrow 0.35, \text{Frame} \rightarrow \text{True}\}, \text{Style} \rightarrow \{ \\
\{"\text{InternalLine}" , _ , _ , \text{mm} /; \text{!FreeQ}[\text{mm}, \text{m1}]\} \rightarrow \{\text{Red}, \text{Thick}\}, \\
\{"\text{InternalLine}" , _ , _ , \text{mm} /; \text{!FreeQ}[\text{mm}, \text{m2}]\} \rightarrow \{\text{Blue}, \text{Thick}\}, \\
\{"\text{InternalLine}" , _ , _ , \text{mm} /; \text{!FreeQ}[\text{mm}, \text{m3}]\} \rightarrow \{\text{Green}, \text{Thick}\}, \\
\{"\text{ExternalLine}" , \text{Q1}\} \rightarrow \{\text{Brown}, \text{Thick}, \text{Dashed}\}\}]
\]
Last but not least: Automatic derivation of Feynman parametrizations for the given integral/topology

Useful if you want to calculate the integral directly e.g. using **HyperInt** \[\text{[Panzer, 2015]}\]

Example: 2-loop self-energy with 2 eikonal propagators

\[
\text{In[55]:=} \quad \text{int} = \text{SFAD}\{(p1, m^2), \text{SFAD}\{(p3, m^2), \text{SFAD}\{((p3-p1), m^2), \text{SFAD}\{\{(0, 2 p1.n, n)\}, \text{SFAD}\{\{0, 2 p3.n\}\})}
\]

\[
\text{Out[55]=} \quad \{ \frac{1}{p1^2-m2^2}, \frac{1}{p3^2-m2^2}, \frac{1}{(p3-p1)^2-m2^2}, \frac{1}{2(n\cdot p1)}, \frac{1}{2(n\cdot p3)} \}
\]

\[
\text{In[56]:=} \quad \text{FCFeynmanParametrize}\{\text{int}, \{p1, p3\}, \text{Names} \rightarrow x, \text{FCReplaceD} \rightarrow \{D \rightarrow 4 - 2\epsilon\}, \text{FinalSubstitutions} \rightarrow \{\text{SPD}\{n\} \rightarrow 1, m \rightarrow 1\} \}
\]

\[
\text{Out[56]=} \quad \{(x(3) x(4)+x(5) x(4)+x(3) x(5))^3 \epsilon^{-1} (m^2 x(3) x(4)^2+m^2 x(3) x(5)^2+
+ m^2 x(4) x(5)^2+m^2 x(3)^2 x(4)+m^2 x(3)^2 x(5)+m^2 x(4)^2 x(5)+
+ 3 m^2 x(3) x(4) x(5)+x(2)^2 x(3)+x(1)^2 x(4)+x(1)^2 x(5)+x(2)^2 x(5)+2 x(1) x(2) x(5))^3 \epsilon^{-1},
-\Gamma(2\epsilon+1),
\{x(1),x(2),x(3),x(4),x(5)\}\}
\]

3-element output: integrand, prefactor, list of integration variables

Many options to fine-tune the output ...

Supports tensor integrals, scalar products in numerators, symbolic propagator powers, Euclidean metric, Cartesian integrals, ...

Some tricks borrowed from **pySecDec** \[\text{[Heinrich et al., 2021]}\], c.f. also PhD thesis of Stefan Jahn \[\text{[Jahn, 2020]}\]
Using FeynCalc in multiloop calculations

- Last but not least: Automatic derivation of Feynman parametrizations for the given integral/topology
- Useful if you want to calculate the integral directly e.g. using **HyperInt** [Panzer, 2015]
- Example: 2-loop self-energy with 2 eikonal propagators

```plaintext
In[57]:= int = SFAD[{{p1, m^2}}, SFAD[{{p3, m^2}}, SFAD[{{(p3 - p1), m^2}}, SFAD[{{0, 2 p1 . n}}, SFAD[{{0, 2 p3 . n}}]]]]
```

```plaintext
Out[57]= \{ \frac{1}{p1^2-m2^2}, \frac{1}{p3^2-m2^2}, \frac{1}{(p3-p1)^2-m2^2}, \frac{1}{(2 (n . p1))/(2 (n . p3))} \}
```

```plaintext
In[58]:= FCFeynmanParametrize[int, {p1, p3}, Names -> x, FCReplaceD -> {D -> 4 - 2 ep}, FinalSubstitutions -> {SPD[n] -> 1, m -> 1}]
```

```plaintext
Out[58]= \{(x(3) x(4)+x(5) x(4)+x(3) x(5))^3 \Gamma(2 ep+1), (m2^2 x(3) x(4) x(5))^{ep-1}, \Gamma(2 ep+1), \}
```

3-element output: integrand, prefactor, list of integration variables

- Many options to fine-tune the output ...
- Supports tensor integrals, scalar products in numerators, symbolic propagator powers, Euclidean metric, Cartesian integrals, ...
- Some tricks borrowed from **pySecDec** [Heinrich et al., 2021], c.f. also PhD thesis of Stefan Jahn [Jahn, 2020]
Using FeynCalc in multi-loop calculations

- Last but not least: Automatic derivation of Feynman parametrizations for the given integral/topology
- Useful if you want to calculate the integral directly e.g. using HyperInt [Panzer, 2015]
- Example: 2-loop self-energy with 2 eikonal propagators

\[
\text{In}[59]]= \text{int} = \text{SFAD}([p1, m^2]), \text{SFAD}([p3, m^2]), \text{SFAD}([(p3 - p1), m^2]), \text{SFAD}([[0, 2 p1 . n]], \text{SFAD}([[0, 2 p3 . n]]))
\]

\[
\text{Out}[59]= \{ \frac{1}{p1^2 - m2^2}, \frac{1}{p3^2 - m2^2}, \frac{1}{(p3-p1)^2 - m2^2}, \frac{1}{(2 (n \cdot p1))}, \frac{1}{(2 (n \cdot p3))} \}
\]

\[
\text{In}[60]]= \text{FCFeynmanParametrize}([\text{int}, \{p1, p3\}], \text{Names} \rightarrow x, \text{FCReplaceD} \rightarrow \{D \rightarrow 4 - 2 \ ep\}, \\
\text{FinalSubstitutions} \rightarrow \{\text{SPD}[n] \rightarrow 1, m \rightarrow 1\})
\]

\[
\text{Out}[60]= \{ \Gamma(2 \ ep+1), \langle x(1), x(2), x(3), x(4), x(5) \rangle \}
\]

- 3-element output: integrand, prefactor, list of integration variables
- Many options to fine-tune the output …
- Supports tensor integrals, scalar products in numerators, symbolic propagator powers, Euclidean metric, Cartesian integrals, …
- Some tricks borrowed from pySecDec [Heinrich et al., 2021], c.f. also PhD thesis of Stefan Jahn [Jahn, 2020]
Summary and Outlook
Summary

**FeynCalc** 10 will feature many flexible and easy-to-use routines for multiloop calculations.

- This functionality can be quite useful for your **FORM**-based setups.
- Standing on the shoulders of giants: **FIRE**, **FIESTA**, **TopoID**, **LiteRed**, **pySecDec**, ...
- Sincere gratitude to the authors of these open-source tools!

Public preview

- Interested? Try it out
  ```plaintext
  In[1]:= Import["https://raw.githubusercontent.com/FeynCalc/feyncalc/master/install.m"]
  InstallFeynCalc[InstallFeynCalcDevelopmentVersion -> True]
  ```

- Everything already documented: [https://feyncalc.github.io/referenceDev](https://feyncalc.github.io/referenceDev)
- Questions or comments? Visit our forum: [https://github.com/FeynCalc/feyncalc/discussions](https://github.com/FeynCalc/feyncalc/discussions)

Outlook

- Topology identification is *mostly* not a performance bottleneck, **Mathematica** sufficient for the time being.
- Multiloop amplitude evaluation: need a **FORM**-based library with **FeynCalc** characteristics (WIP)
- Next milestone: support helicity amplitude methods (**FeynCalc** 11?)
Summary

- **FeynCalc** 10 will feature many flexible and easy-to-use routines for multiloop calculations.
- This functionality can be quite useful for your FORM-based setups.
- Standing on the shoulders of giants: FIRE, FIESTA, TopoID, LiteRed, pySecDec, ...
- Sincere gratitude to the authors of these open-source tools!

Public preview

- Interested? Try it out
  ```mathematica
  In[1]:= Import["https://raw.githubusercontent.com/FeynCalc/feyncalc/master/install.m"]
  InstallFeynCalc[InstallFeynCalcDevelopmentVersion -> True]
  
  Everything already documented: https://feyncalc.github.io/referenceDev
  
  Questions or comments? Visit our forum: https://github.com/FeynCalc/feyncalc/discussions
  
Outlook

- Topology identification is *mostly* not a performance bottleneck, Mathematica sufficient for the time being.
- Multiloop amplitude evaluation: need a FORM-based library with FeynCalc characteristics (WIP)
- Next milestone: support helicity amplitude methods (FeynCalc 11?)
Summary

- **FeynCalc** 10 will feature many flexible and easy-to-use routines for multiloop calculations
- This functionality can be quite useful for your **FORM**-based setups
- Standing on the shoulders of giants: **FIRE**, **FIESTA**, **TopoID**, **LiteRed**, **pySecDec**, ...
- Sincere gratitude to the authors of these open-source tools!

Public preview

- Interested? Try it out
  ```plaintext
  In[1]:= Import["https://raw.githubusercontent.com/FeynCalc/feyncalc/master/install.m"]
  InstallFeynCalc[InstallFeynCalcDevelopmentVersion -> True]
  ```
- Everything already documented: [https://feyncalc.github.io/referenceDev](https://feyncalc.github.io/referenceDev)
- Questions or comments? Visit our forum: [https://github.com/FeynCalc/feyncalc/discussions](https://github.com/FeynCalc/feyncalc/discussions)

Outlook

- Topology identification is *mostly* not a performance bottleneck, **Mathematica** sufficient for the time being.
- Multiloop amplitude evaluation: need a **FORM**-based library with **FeynCalc** characteristics (WIP)
- Next milestone: support helicity amplitude methods (**FeynCalc** 11?)
Backup
Brief summary of Pak's algorithm:

- Char. polynomial $P$ (l terms, $n$ Feynman parameters $x_i$) as an $l \times (n + 1)$ matrix
- Set $i = 1$, switch the $(i + 1)$-th column with each of the next columns keep track of the permutations $\Rightarrow$ new matrices
- looking only at the first $(i + 1)$-columns of the new matrices, sort their rows (e.g. lexicographically)
- sorted matrices: extract the $i$-th column $\Rightarrow$ list of vectors
- sort the vectors and select the first one in the sorted list
- keep all matrices containing the selected column, discard the rest $\Rightarrow$ final set of matrices in this iteration
- $i + +$ while $i < n - 1$, otherwise return the final permutations $\sigma$.

$$P = c_2 x_2 x_3 + c_1 x_2^2 + c_2 x_1 x_3 + c_1 x_1^2 \Rightarrow \begin{pmatrix} c_2 & 0 & 1 & 1 \\ c_1 & 0 & 2 & 0 \\ c_2 & 1 & 0 & 1 \\ c_1 & 2 & 0 & 0 \end{pmatrix} \equiv M_0^{(123)}$$

1st iteration: $i = 1$, start with $\{ M_0^{(123)} \}$ permute the 2nd column

$$\{ M_0^{(123)} = \begin{pmatrix} c_2 & 0 & 1 & 1 \\ c_1 & 0 & 2 & 0 \\ c_2 & 1 & 0 & 1 \\ c_1 & 2 & 0 & 0 \end{pmatrix}, \quad M_0^{(213)} = \begin{pmatrix} c_2 & 1 & 0 & 1 \\ c_1 & 2 & 0 & 0 \\ c_2 & 0 & 1 & 1 \\ c_1 & 0 & 2 & 0 \end{pmatrix} \}$$

$$M_0^{(321)} = \begin{pmatrix} c_2 & 1 & 1 & 0 \\ c_1 & 0 & 2 & 0 \\ c_2 & 1 & 0 & 1 \\ c_1 & 0 & 0 & 2 \end{pmatrix}$$

After sorting rows w.r.t to the first 2 columns

$$\{ \tilde{M}_0^{(123)} = \begin{pmatrix} c_1 & 0 & 2 & 0 \\ c_1 & 2 & 0 & 0 \\ c_2 & 0 & 1 & 1 \\ c_2 & 1 & 0 & 1 \end{pmatrix}, \quad \tilde{M}_0^{(213)} = \begin{pmatrix} c_1 & 0 & 2 & 0 \\ c_1 & 2 & 0 & 0 \\ c_2 & 0 & 1 & 1 \\ c_2 & 1 & 0 & 1 \end{pmatrix} \}$$

$$\tilde{M}_0^{(321)} = \begin{pmatrix} c_1 & 0 & 2 & 0 \\ c_1 & 0 & 0 & 2 \\ c_2 & 1 & 1 & 0 \\ c_2 & 1 & 0 & 1 \end{pmatrix}$$

Max vector among the 2nd columns: $(0, 2, 0, 1)^T \Rightarrow$ keep $\tilde{M}_0^{(123)}$ and $\tilde{M}_0^{(213)}$
Backup

Brief summary of Pak's algorithm:

- Char. polynomial $P$ ($l$ terms, $n$ Feynman parameters $x_i$) as an $l \times (n + 1)$ matrix
- Set $i = 1$, switch the $(i + 1)$-th column with each of the next columns keep track of the permutations $\Rightarrow$ new matrices
- looking only at the first $(i + 1)$-columns of the new matrices, sort their rows (e.g. lexicographically)
- sorted matrices: extract the $i$-th column $\Rightarrow$ list of vectors
- sort the vectors and select the first one in the sorted list
- keep all matrices containing the selected column, discard the rest $\Rightarrow$ final set of matrices in this iteration
- $i +=$ while $i < n - 1$, otherwise return the final permutations $\sigma$.

2nd iteration: $i = 2$, start with

$$\{\tilde{M}_0^{(123)} = \begin{pmatrix} c_1 & 0 & 2 & 0 \\ c_1 & 2 & 0 & 0 \\ c_2 & 0 & 1 & 1 \\ c_2 & 1 & 0 & 1 \end{pmatrix}, \tilde{M}_0^{(213)} = \begin{pmatrix} c_1 & 0 & 2 & 0 \\ c_1 & 2 & 0 & 0 \\ c_2 & 0 & 1 & 1 \\ c_2 & 1 & 0 & 1 \end{pmatrix}\}$$

Permute the 3rd column

$$\{M_1^{(123)} = \begin{pmatrix} c_1 & 0 & 2 & 0 \\ c_1 & 2 & 0 & 0 \\ c_2 & 0 & 1 & 1 \\ c_2 & 1 & 0 & 1 \end{pmatrix}, M_1^{(132)} = \begin{pmatrix} c_1 & 0 & 0 & 2 \\ c_1 & 2 & 0 & 0 \\ c_2 & 0 & 1 & 1 \\ c_2 & 1 & 1 & 0 \end{pmatrix}\},$$

$$M_1^{(213)} = \begin{pmatrix} c_1 & 0 & 2 & 0 \\ c_1 & 2 & 0 & 0 \\ c_2 & 0 & 1 & 1 \\ c_2 & 1 & 0 & 1 \end{pmatrix} \quad M_1^{(231)} = \begin{pmatrix} c_1 & 0 & 0 & 2 \\ c_1 & 2 & 0 & 0 \\ c_2 & 0 & 1 & 1 \\ c_2 & 1 & 1 & 0 \end{pmatrix}\}$$
Backup

Brief summary of Pak’s algorithm:

- Char. polynomial $P$ ($l$ terms, $n$ Feynman parameters $x_i$) as an $l \times (n + 1)$ matrix

- Set $i = 1$, switch the $(i + 1)$-th column with each of the next columns keep track of the permutations $\Rightarrow$ new matrices

- looking only at the first $(i + 1)$-columns of the new matrices, sort their rows (e.g. lexicographically)

- sorted matrices: extract the $i$-th column $\Rightarrow$ list of vectors

- sort the vectors and select the first one in the sorted list

- keep all matrices containing the selected column, discard the rest $\Rightarrow$ final set of matrices in this iteration

- $i++$ while $i < n - 1$, otherwise return the final permutations $\sigma$.

After sorting rows w.r.t the first 3 columns (no changes here)

$$
\{ \tilde{M}_1^{(123)} = \begin{pmatrix} c_1 & 0 & 2 & 0 \\ c_1 & 2 & 0 & 0 \\ c_2 & 0 & 1 & 1 \\ c_2 & 1 & 0 & 1 \end{pmatrix}, \tilde{M}_1^{(132)} = \begin{pmatrix} c_1 & 0 & 0 & 2 \\ c_1 & 2 & 0 & 0 \\ c_2 & 0 & 1 & 1 \\ c_2 & 1 & 1 & 0 \end{pmatrix}, \\
\tilde{M}_1^{(213)} = \begin{pmatrix} c_1 & 0 & 2 & 0 \\ c_1 & 2 & 0 & 0 \\ c_2 & 0 & 1 & 1 \\ c_2 & 1 & 1 & 0 \end{pmatrix}, \tilde{M}_1^{(231)} = \begin{pmatrix} c_1 & 0 & 0 & 2 \\ c_1 & 2 & 0 & 0 \\ c_2 & 0 & 1 & 1 \\ c_2 & 1 & 1 & 0 \end{pmatrix} \}$$

Max vector among the 3rd columns: $(2, 0, 1, 0)^T$  
$\Rightarrow$ keep $\tilde{M}_1^{(123)}$ and $\tilde{M}_1^{(213)}$

Since $i = 3 \geq n - 1 = 3$ the algorithm terminates here.

Symmetries under renamings of $x_i$: $\sigma = \{(123), (213)\}$

Meaning that

$$
P^{(123)} = c_2 x_2 x_3 + c_1 x_2^2 + c_2 x_1 x_3 + c_1 x_1^2$$

$$
P^{(213)} = c_2 x_1 x_3 + c_1 x_1^2 + c_2 x_2 x_3 + c_1 x_2^2$$

are equivalent (obviously)