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

Karlsruhe Institute of Technology

Outline

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]

- **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]

- **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]

Symanzik polynomials and Pak algorithm

- Feynman parametric representation of an L-loop scalar Minkowskian integral

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

  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

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

  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

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

  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 ...

- 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 [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 [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 [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 [Pak, 2012]

- 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

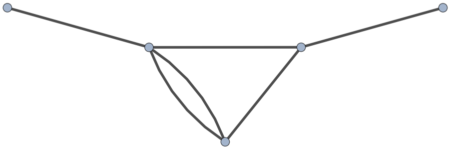
- Two new **FEYNCALC** symbols for multiloop calculations
- `FCTopology[id,{propagators}, {loop momenta}, {external momenta}, {kinematics}, {}]` denotes a loop integral family `id`
- `GLI[id,{propagator powers}]` is a loop integral belonging to the integral family `id`
- Syntax inspired by **FIRE** and **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

- 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`)

```
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

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

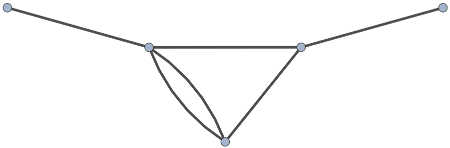
Out[2]= $G^{\text{prop2L}}(1,1,0,1,1)$

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

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

Out[3]= $\dfrac{1}{(p1^2-m1^2)\,(p2^2-m2^2)\,((p2+q)^2-m4^2)\,((p1-p2)^2-m5^2)}$

- 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`)

```
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

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

Out[2]= $G^{\text{prop2L}}(1,1,0,1,1)$

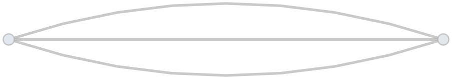
- Simplest manipulation: convert a `GLI` symbol to the `(S)FAD`-notation

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

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 $\mathcal{U}$ and $\mathcal{F}$ polynomials of the given integral?
- Example: 2-loop massive tadpole (here we use the `(S)FAD`-notation)



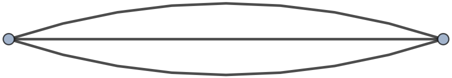
- $\mathcal{U}$ and $\mathcal{F}$: first two entries in the list returned by `FCFeynmanPrepare`

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

Out[4]//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^μ 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 $\mathcal{U}$ and $\mathcal{F}$ polynomials of the given integral?
- Example: 2-loop massive tadpole (here we use the `(S)FAD`-notation)



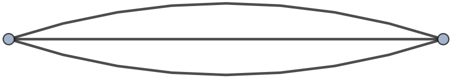
- $\mathcal{U}$ and $\mathcal{F}$: first two entries in the list returned by `FCFeynmanPrepare`

```
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^μ 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 $\mathcal{U}$ and $\mathcal{F}$ polynomials of the given integral?
- Example: 2-loop massive tadpole (here we use the `(S)FAD`-notation)



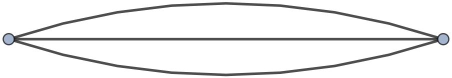
- $\mathcal{U}$ and $\mathcal{F}$: first two entries in the list returned by `FCFeynmanPrepare`

```
In[6]:= FCFeynmanPrepare[SFAD[{p1, m1^2}, {p2, m2^2}, {p1 - p2, m3^2}], {p1, p2}, Names -> x][[1 ;; 2]] // TableForm

Out[6]//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^μ 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 $\mathcal{U}$ 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[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 $\mathcal{U} = \det M$
  - J and Q^μ from $\mathcal{F} = \det\, M(QM^{-1}Q - J)$

- What about the characteristic polynomial $P = \mathcal{U} \times \mathcal{F}$?
- Same specimen, the 2-loop tadpole

```
In[8]:= FCLoopToPakForm[FAD[{p1, m1}, {p2, m2}, {p1 - p2, m3}], {p1, p2}, Names -> x][[2]][[1]]
```

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

- 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`

```
In[9]:= 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];
```

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

```
In[10]:= FCLoopPakOrder[poly, x, Rename -> True]
```

Out[10]= $-\frac{1}{4}\, x(3)\, x(1)^2-\frac{1}{4}\, x(5)\, x(1)^2+\frac{1}{2}\, x(2)\, x(3)\, x(1)-\frac{1}{4}\, x(2)^2\, x(3)-\frac{1}{4}\, x(2)^2\, x(4)+x(3)\, x(4)\, x(5)$

- What about the characteristic polynomial $P = \mathcal{U} \times \mathcal{F}$?
- Same specimen, the 2-loop tadpole

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

Out[11]= $m1^2\, x(1)^2\, x(2)+m1^2\, x(1)^2\, x(3)+m1^2\, x(1)\, x(2)\, x(3)+m2^2\, x(1)\, x(2)^2+m2^2\, x(2)^2\, x(3)+m2^2\, x(1)\, x(2)\, x(3)+m3^2\, x(1)\, x(3)^2+m3^2\, x(2)\, x(3)^2+m3^2\, x(1)\, x(2)\, x(3)+x(1)\, x(2)+x(1)\, x(3)+x(2)\, x(3)$

- 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`

```
In[12]:= 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];
```

Out[12]= $-\frac{1}{4}\, x(4)\, x(1)^2-\frac{1}{4}\, x(5)\, x(1)^2+\frac{1}{2}\, x(2)\, x(5)\, x(1)-\frac{1}{4}\, x(2)^2\, x(3)-\frac{1}{4}\, x(2)^2\, x(5)+x(3)\, x(4)\, x(5)$

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

Out[13]= $-\frac{1}{4}\, x(3)\, x(1)^2-\frac{1}{4}\, x(5)\, x(1)^2+\frac{1}{2}\, x(2)\, x(3)\, x(1)-\frac{1}{4}\, x(2)^2\, x(3)-\frac{1}{4}\, x(2)^2\, x(4)+x(3)\, x(4)\, x(5)$

- What about the characteristic polynomial $P = \mathcal{U} \times \mathcal{F}$?
- Same specimen, the 2-loop tadpole

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

Out[14]= $m1^2\, x(1)^2\, x(2)+m1^2\, x(1)^2\, x(3)+m1^2\, x(1)\, x(2)\, x(3)+m2^2\, x(1)\, x(2)^2+m2^2\, x(2)^2\, x(3)+m2^2\, x(1)\, x(2)\, x(3)+m3^2\, x(1)\, x(3)^2+m3^2\, x(2)\, x(3)^2+m3^2\, x(1)\, x(2)\, x(3)+x(1)\, x(2)+x(1)\, x(3)+x(2)\, x(3)$

- 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

```
In[15]:= 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];
```

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

```
In[16]:= FCLoopPakOrder[poly, x, Rename -> True]
```

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 $P = \mathcal{U} \times \mathcal{F}$?
- Same specimen, the 2-loop tadpole

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

Out[17]= $m1^2\, x(1)^2\, x(2)+m1^2\, x(1)^2\, x(3)+m1^2\, x(1)\, x(2)\, x(3)+m2^2\, x(1)\, x(2)^2+m2^2\, x(2)^2\, x(3)+m2^2\, x(1)\, x(2)\, x(3)+m3^2\, x(1)\, x(3)^2+m3^2\, x(2)\, x(3)^2+m3^2\, x(1)\, x(2)\, x(3)+x(1)\, x(2)+x(1)\, x(3)+x(2)\, x(3)$

- 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

```
In[18]:= 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];
```

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

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

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

```
In[20]:= FCLoopToPakForm[FAD[{p1, m1}, {p2, m2}, {p1 - p2, m3}], {p1, p2}, Names -> x][[2]][[1]]
```

Out[20]= $m1^2\, x(1)^2\, x(2)+m1^2\, x(1)^2\, x(3)+m1^2\, x(1)\, x(2)\, x(3)+m2^2\, x(1)\, x(2)^2+m2^2\, x(2)^2\, x(3)+m2^2\, x(1)\, x(2)\, x(3)+m3^2\, x(1)\, x(3)^2+m3^2\, x(2)\, x(3)^2+m3^2\, x(1)\, x(2)\, x(3)+x(1)\, x(2)+x(1)\, x(3)+x(2)\, x(3)$

- 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`

```
In[21]:= 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];
```

Out[21]= $-\frac{1}{4}\, x(4)\, x(1)^2-\frac{1}{4}\, x(5)\, x(1)^2+\frac{1}{2}\, x(2)\, x(5)\, x(1)-\frac{1}{4}\, x(2)^2\, x(3)-\frac{1}{4}\, x(2)^2\, x(5)+x(3)\, x(4)\, x(5)$

```
In[22]:= FCLoopPakOrder[poly, x, Rename -> True]
```

Out[22]= $-\frac{1}{4}\, x(3)\, x(1)^2-\frac{1}{4}\, x(5)\, x(1)^2+\frac{1}{2}\, x(2)\, x(3)\, x(1)-\frac{1}{4}\, x(2)^2\, x(3)-\frac{1}{4}\, x(2)^2\, x(4)+x(3)\, x(4)\, x(5)$

- 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]= $\{\text{FCTopology}(\text{topo1},\{\frac{1}{p1^2},\frac{1}{p2^2},\frac{1}{p3^2},\frac{1}{(p1+p3)^2},\frac{1}{(p2+p3)^2},\frac{1}{(p2-Q)^2},\frac{1}{(p1+p3-Q)^2},\frac{1}{(p2+p3-Q)^2},\frac{1}{(p1+p2+p3-Q)^2}\},\{p1,p2,p3\},\{Q\},\{\},\{\}),$
$\text{FCTopology}(\text{topo2},\{\frac{1}{p1^2},\frac{1}{p2^2},\frac{1}{p3^2},\frac{1}{(p1+p3)^2},\frac{1}{(p2+p3)^2},\frac{1}{(p1-Q)^2},\frac{1}{(p2-Q)^2},\frac{1}{(p1+p3-Q)^2},\frac{1}{(p1+p2+p3-Q)^2}\},\{p1,p2,p3\},\{Q\},\{\},\{\})\}$

```
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]= $\{\text{FCTopology}(\text{prop3L},\{\frac{1}{p1^2},\frac{1}{p2^2},\frac{1}{p3^2},\frac{1}{(p2+p3)^2},\frac{1}{(p1-Q)^2},\frac{1}{(p2-Q)^2},\frac{1}{(p1+p3-Q)^2},\frac{1}{(p2+p3-Q)^2},\frac{1}{(p1+p2+p3-Q)^2}\},\{p1,p2,p3\},\{Q\},\{\},\{\})\}$

- 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

```
In[26]:= 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[26]= $\{\mathrm{FCTopology}(\mathrm{topo1},\{\frac{1}{p1^2},\frac{1}{p2^2},\frac{1}{p3^2},\frac{1}{(p1+p3)^2},\frac{1}{(p2+p3)^2},\frac{1}{(p2-Q)^2},\frac{1}{(p1+p3-Q)^2},\frac{1}{(p2+p3-Q)^2},\frac{1}{(p1+p2+p3-Q)^2}\},\{p1,p2,p3\},\{Q\},\{\},\{\}),$
$\mathrm{FCTopology}(\mathrm{topo2},\{\frac{1}{p1^2},\frac{1}{p2^2},\frac{1}{p3^2},\frac{1}{(p1+p3)^2},\frac{1}{(p2+p3)^2},\frac{1}{(p1-Q)^2},\frac{1}{(p2-Q)^2},\frac{1}{(p1+p3-Q)^2},\frac{1}{(p1+p2+p3-Q)^2}\},\{p1,p2,p3\},\{Q\},\{\},\{\})\}$

```
In[27]:= 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[27]= $\{\mathrm{FCTopology}(\mathrm{prop3L},\{\frac{1}{p1^2},\frac{1}{p2^2},\frac{1}{p3^2},\frac{1}{(p2+p3)^2},\frac{1}{(p1-Q)^2},\frac{1}{(p2-Q)^2},\frac{1}{(p1+p3-Q)^2},\frac{1}{(p2+p3-Q)^2},\frac{1}{(p1+p2+p3-Q)^2}\},\{p1,p2,p3\},\{Q\},\{\},\{\})\}$

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

```
In[28]:= 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

```
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]= $\{\mathrm{FCTopology}(\mathrm{topo1},\{\frac{1}{p1^2},\frac{1}{p2^2},\frac{1}{p3^2},\frac{1}{(p1+p3)^2},\frac{1}{(p2+p3)^2},\frac{1}{(p2-Q)^2},\frac{1}{(p1+p3-Q)^2},\frac{1}{(p2+p3-Q)^2},\frac{1}{(p1+p2+p3-Q)^2}\},\{p1,p2,p3\},\{Q\},\{\},\{\}),$
$\mathrm{FCTopology}(\mathrm{topo2},\{\frac{1}{p1^2},\frac{1}{p2^2},\frac{1}{p3^2},\frac{1}{(p1+p3)^2},\frac{1}{(p2+p3)^2},\frac{1}{(p1-Q)^2},\frac{1}{(p2-Q)^2},\frac{1}{(p1+p3-Q)^2},\frac{1}{(p1+p2+p3-Q)^2}\},\{p1,p2,p3\},\{Q\},\{\},\{\})\}$

```
In[30]:= 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[30]= $\{\mathrm{FCTopology}(\mathrm{prop3L},\{\frac{1}{p1^2},\frac{1}{p2^2},\frac{1}{p3^2},\frac{1}{(p2+p3)^2},\frac{1}{(p1-Q)^2},\frac{1}{(p2-Q)^2},\frac{1}{(p1+p3-Q)^2},\frac{1}{(p2+p3-Q)^2},\frac{1}{(p1+p2+p3-Q)^2}\},\{p1,p2,p3\},\{Q\},\{\},\{\})\}$

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

```
In[31]:= mappings = FCLoopFindTopologyMappings[Join[source, target], PreferredTopologies -> prop3L]
```

- Let us examine the output of

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

  in more details
- The first entry maps `topo1` to `prop3L`

  In[33]:= **mappings[[1]]**

  Out[33]= $\{\mathrm{FCTopology}(\mathrm{topo1},\{\frac{1}{p1^2},\frac{1}{p2^2},\frac{1}{p3^2},\frac{1}{(p1+p3)^2},\frac{1}{(p2+p3)^2},\frac{1}{(p2-Q)^2},\frac{1}{(p1+p3-Q)^2},\frac{1}{(p2+p3-Q)^2},\frac{1}{(p1+p2+p3-Q)^2}\},\{p1,p2,p3\},\{Q\},\{\},\{\}),$
  $\{p1\to -p1-p3+Q,p2\to -p2-p3+Q,p3\to p3\},\ G^{\mathrm{topo1}}(n7\_,n8\_,n3\_,n5\_,n6\_,n4\_,n1\_,n2\_,n9\_):\to G^{\mathrm{prop3L}}(n1,n2,n3,n4,n5,n6,n7,n8,n9)\}$
- 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 `topo1` to integrals from `prop3L`!

- Let us examine the output of

In[34]:= **mappings = FCLoopFindTopologyMappings[Join[source, target], PreferredTopologies -> {prop3L}]**

in more details

- The first entry maps `topo1` to `prop3L`

In[35]:= **mappings[[1]]**

Out[35]= $\{\text{FCTopology}(\text{topo1},\{\frac{1}{p1^2},\frac{1}{p2^2},\frac{1}{p3^2},\frac{1}{(p1+p3)^2},\frac{1}{(p2+p3)^2},\frac{1}{(p2-Q)^2},\frac{1}{(p1+p3-Q)^2},\frac{1}{(p2+p3-Q)^2},\frac{1}{(p1+p2+p3-Q)^2}\},\{p1,p2,p3\},\{Q\},\{\},\{\}),$
$\{p1\to -p1-p3+Q, p2\to -p2-p3+Q, p3\to p3\}, G^{\text{topo1}}(n7\_,n8\_,n3\_,n5\_,n6\_,n4\_,n1\_,n2\_,n9\_) :\to G^{\text{prop3L}}(n1,n2,n3,n4,n5,n6,n7,n8,n9)\}$

- 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 `topo1` to integrals from `prop3L`!

- Let us examine the output of

```
In[36]:= mappings = FCLoopFindTopologyMappings[Join[source, target], PreferredTopologies -> {prop3L}]
```

  in more details

- The first entry maps `topo1` to `prop3L`

```
In[37]:= mappings[[1]]
```

Out[37]= $\{\mathrm{FCTopology}(\mathrm{topo1},\{\frac{1}{p1^2},\frac{1}{p2^2},\frac{1}{p3^2},\frac{1}{(p1+p3)^2},\frac{1}{(p2+p3)^2},\frac{1}{(p2-Q)^2},\frac{1}{(p1+p3-Q)^2},\frac{1}{(p2+p3-Q)^2},\frac{1}{(p1+p2+p3-Q)^2}\},\{p1,p2,p3\},\{Q\},\{\},\{\}),$
$\{p1\to -p1-p3+Q,p2\to -p2-p3+Q,p3\to p3\},\ G^{\mathrm{topo1}}(n7\_,n8\_,n3\_,n5\_,n6\_,n4\_,n1\_,n2\_,n9\_):\ \to G^{\mathrm{prop3L}}(n1,n2,n3,n4,n5,n6,n7,n8,n9)\}$

- 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 `topo1` to integrals from `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.)**

- 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}]}
```

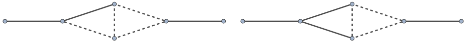
- FeynCalc identifies three mappings between our masters

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

Out[40]= $\{G^{\text{prop2Lv2}}(1,0,1,1,1)\to G^{\text{prop2Lv1}}(0,1,1,1,1), G^{\text{prop2Lv2}}(1,1,0,1,1)\to G^{\text{prop2Lv1}}(0,1,1,1,1), G^{\text{prop2Lv2}}(1,1,1,0,1)\to G^{\text{prop2Lv2}}(0,1,1,1,1)\}$

- 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



```
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}]}
```

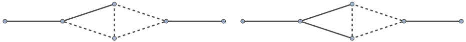
- FEYNCALC identifies three mappings between our masters

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

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

- 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



```
In[44]:= 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[45]:= 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}]}
```

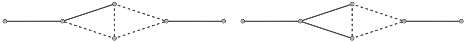
- **FeynCalc** identifies three mappings between our masters

```
In[46]:= FCLoopFindIntegralMappings[glis, topos][[1]]
```

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

- 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



```
In[47]:= 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[48]:= 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

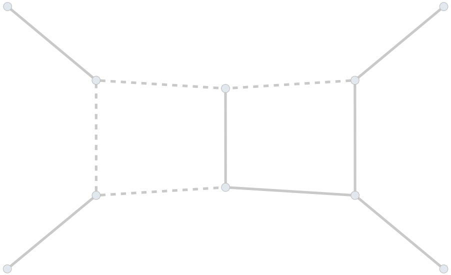
```
In[49]:= FCLoopFindIntegralMappings[glis, topos][[1]]
```

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

- Working principle based on `FindRules` from **FIRE**
- Can specify a set of preferred master integrals via the `PreferredIntegrals` option

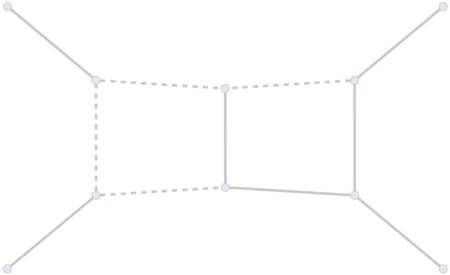
- Given a propagator representation of a loop integral, how to obtain its graph representation?
- Can be done using AZURITE [Georgoudis et al., 2017], PlanarityTest, [Bielas et al., 2013], LiteRed and now also FeynCalc
- Two-step approach: `FCLoopIntegralToGraph` reconstructs the graph, `FCLoopGraphPlot` plots it.

```
In[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[%]
```



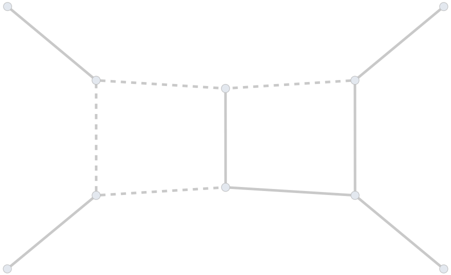
- Given a propagator representation of a loop integral, how to obtain its graph representation?
- Can be done using **AZURITE** [Georgoudis et al., 2017], **PlanarityTest**, [Bielas et al., 2013], **LiteRed** and now also **FeynCalc**
- Two-step approach: `FCLoopIntegralToGraph` reconstructs the graph, `FCLoopGraphPlot` plots it.

```
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[%]
```



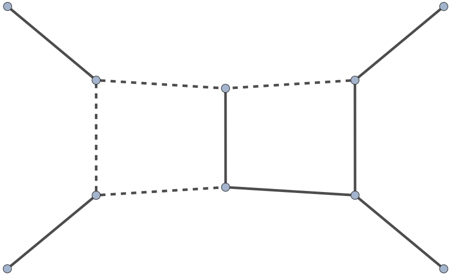
- Given a propagator representation of a loop integral, how to obtain its graph representation?
- Can be done using **AZURITE** [Georgoudis et al., 2017], **PlanarityTest**, [Bielas et al., 2013], **LiteRed** and now also **FeynCalc**
- Two-step approach: `FCLoopIntegralToGraph` reconstructs the graph, `FCLoopGraphPlot` plots it.

```
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[%]
```



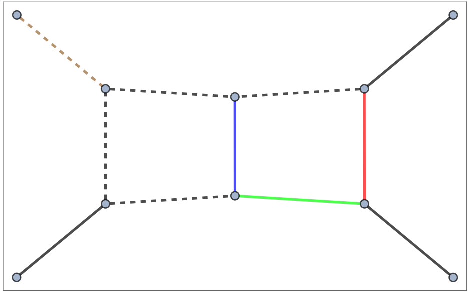
- Given a propagator representation of a loop integral, how to obtain its graph representation?
- Can be done using **AZURITE** [Georgoudis et al., 2017], **PLANARITYTEST**, [Bielas et al., 2013], **LITERED** and now also **FEYNCALC**
- Two-step approach: `FCLoopIntegralToGraph` reconstructs the graph, `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

```
In[54]:= FCLoopIntegralToGraph[FAD[{p1, m1}, {p2, m2}, {Q1 + p1, m3}, Q2 - p1, Q1 + p1 + p2, Q2 - p1 - p2, Q2 + Q3 - p1 - p2], {p1, p2}];
FCLoopGraphPlot[%, GraphPlot -> {MultiedgeStyle -> 0.35, Frame -> True}, Style -> {
{"InternalLine", _, _, mm_ /; ! FreeQ[mm, m1]} -> {Red, Thick},
{"InternalLine", _, _, mm_ /; ! FreeQ[mm, m2]} -> {Blue, Thick},
{"InternalLine", _, _, mm_ /; ! FreeQ[mm, m3]} -> {Green, Thick},
{"ExternalLine", Q1} -> {Brown, Thick, 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** [Panzer, 2015]
- Example: 2-loop self-energy with 2 eikonal propagators

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

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

```
In[56]:= FCFeynmanParametrize[int, {p1, p3}, Names -> x, FCReplaceD -> {D -> 4 - 2 ep},
FinalSubstitutions -> {SPD[n] -> 1, m -> 1}]
```

Out[56]= $\{(x(3)\,x(4)+x(5)\,x(4)+x(3)\,x(5))^{3\,ep-1}\,(m2^2\,x(3)\,x(4)^2+m2^2\,x(3)\,x(5)^2+$
$m2^2\,x(4)\,x(5)^2+m2^2\,x(3)^2\,x(4)+m2^2\,x(3)^2\,x(5)+m2^2\,x(4)^2\,x(5)+$
$3\,m2^2\,x(3)\,x(4)\,x(5)+x(2)^2\,x(3)+x(1)^2\,x(4)+x(1)^2\,x(5)+x(2)^2\,x(5)+2\,x(1)\,x(2)\,x(5))^{-2\,ep-1},$
$-\Gamma(2\,ep+1),$
$\{x(1),x(2),x(3),x(4),x(5)\}\}$

- 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 [Heinrich et al., 2021], c. f. also PhD thesis of Stefan Jahn [Jahn, 2020]

- 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

```
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}}]
```

Out[57]= $\left\{\frac{1}{p1^2-m2^2}, \frac{1}{p3^2-m2^2}, \frac{1}{(p3-p1)^2-m2^2}, \frac{1}{(2\,(n\cdot p1))}, \frac{1}{(2\,(n\cdot p3))}\right\}$

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

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

- 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** [Heinrich et al., 2021], c. f. also PhD thesis of Stefan Jahn [Jahn, 2020]

- 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

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

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

```
In[60]:= FCFeynmanParametrize[int, {p1, p3}, Names -> x, FCReplaceD -> {D -> 4 - 2 ep},
         FinalSubstitutions -> {SPD[n] -> 1, m -> 1}]
```

Out[60]= $\{(x(3)\,x(4)+x(5)\,x(4)+x(3)\,x(5))^{3\,ep-1}\,(m2^2\,x(3)\,x(4)^2+m2^2\,x(3)\,x(5)^2+$
$m2^2\,x(4)\,x(5)^2+m2^2\,x(3)^2\,x(4)+m2^2\,x(3)^2\,x(5)+m2^2\,x(4)^2\,x(5)+$
$3\,m2^2\,x(3)\,x(4)\,x(5)+x(2)^2\,x(3)+x(1)^2\,x(4)+x(1)^2\,x(5)+x(2)^2\,x(5)+2\,x(1)\,x(2)\,x(5))^{-2\,ep-1},$
$-\Gamma(2\,ep+1),$
$\{x(1),x(2),x(3),x(4),x(5)\}\}$

- 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** [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

```
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

```
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

```
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?)

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 σ.

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

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

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

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

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

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

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

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

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 σ.

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

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

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

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 σ.

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

$$\{\tilde{M}_1^{(123)} = \begin{pmatrix} \boldsymbol{c}_1 & \mathbf{0} & \mathbf{2} & 0 \\ \boldsymbol{c}_1 & \mathbf{2} & \mathbf{0} & 0 \\ \boldsymbol{c}_2 & \mathbf{0} & \mathbf{1} & 1 \\ \boldsymbol{c}_2 & \mathbf{1} & \mathbf{0} & 1 \end{pmatrix}, \tilde{M}_1^{(132)} = \begin{pmatrix} \boldsymbol{c}_1 & \mathbf{0} & \mathbf{0} & 2 \\ \boldsymbol{c}_1 & \mathbf{2} & \mathbf{0} & 0 \\ \boldsymbol{c}_2 & \mathbf{0} & \mathbf{1} & 1 \\ \boldsymbol{c}_2 & \mathbf{1} & \mathbf{1} & 0 \end{pmatrix},$$

$$\tilde{M}_1^{(213)} = \begin{pmatrix} \boldsymbol{c}_1 & \mathbf{0} & \mathbf{2} & 0 \\ \boldsymbol{c}_1 & \mathbf{2} & \mathbf{0} & 0 \\ \boldsymbol{c}_2 & \mathbf{0} & \mathbf{1} & 1 \\ \boldsymbol{c}_2 & \mathbf{1} & \mathbf{0} & 1 \end{pmatrix} \tilde{M}_1^{(231)} = \begin{pmatrix} \boldsymbol{c}_1 & \mathbf{0} & \mathbf{0} & 2 \\ \boldsymbol{c}_1 & \mathbf{2} & \mathbf{0} & 0 \\ \boldsymbol{c}_2 & \mathbf{0} & \mathbf{1} & 1 \\ \boldsymbol{c}_2 & \mathbf{1} & \mathbf{1} & 0 \end{pmatrix}\}$$

Max vector among the 3rd columns : $(\mathbf{2}, \mathbf{0}, \mathbf{1}, \mathbf{0})^T$
$\Rightarrow$ 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)