Computer Physics Communications 207 (2016) 432–444

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

New developments in FeynCalc 9.0☆

Vladyslav Shtabovenko [a,*], Rolf Mertig [b,*], Frederik Orellana [c]

[a] Technische Universität München, Physik-Department T30f, James-Franck-Str. 1, 85747 Garching, Germany
[b] GluonVision GmbH, Bötzowstr. 10, 10407 Berlin, Germany
[c] Technical University of Denmark, Anker Engelundsvej 1, Building 101A, 2800 Kgs. Lyngby, Denmark

ARTICLE INFO

Article history:
Received 14 February 2016
Received in revised form
4 June 2016
Accepted 14 June 2016
Available online 1 July 2016

Keywords:
High energy physics
Feynman diagrams
Loop integrals
Dimensional regularization
Dirac algebra
Color algebra
Tensor reduction

ABSTRACT

In this note we report on the new version of FeynCalc, a Mathematica package for symbolic semi-automatic evaluation of Feynman diagrams and algebraic expressions in quantum field theory. The main features of version 9.0 are: improved tensor reduction and partial fractioning of loop integrals, new functions for using FeynCalc together with tools for reduction of scalar loop integrals using integration-by-parts (IBP) identities, better interface to FeynArts and support for $SU(N)$ generators with explicit fundamental indices.

Program summary

Program title: FeynCalc

Catalogue identifier: AFBB_v1_0

Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AFBB_v1_0.html

Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland

Licensing provisions: GNU Public Licence 3

No. of lines in distributed program, including test data, etc.: 734115

No. of bytes in distributed program, including test data, etc.: 6890074

Distribution format: tar.gz

Programming language: Wolfram Mathematica 8 and higher.

Computer: Any computer that can run Mathematica 8 and higher.

Operating system: Windows, Linux, OS X.

Classification: 4.4, 5, 11.1.

External routines: FeynArts [2] (Included)

Nature of problem: Symbolic semi-automatic evaluation of Feynman diagrams and algebraic expressions in quantum field theory.

Solution method: Algebraic identities that are needed for evaluation of Feynman

Reasons for new version: Compatibility with Mathematica 10, improved performance and new features regarding manipulation of loop integrals.

Restrictions: Slow performance for multi-particle processes (beyond $1 \to 2$ and $2 \to 2$) and processes that involve large (>100) number of Feynman diagrams.

Additional comments: The original FeynCalc paper was published in Comput. Phys. Commun., 64 (1991) 345, but the code was not included in the Library at that time.

Reasons for the new version: Compatibility with Mathematica 10, improved performance and new features regarding manipulation of loop integrals.

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding authors.
E-mail addresses: v.shtabovenko@tum.de (V. Shtabovenko), rolfm@gluonvision.com (R. Mertig).

http://dx.doi.org/10.1016/j.cpc.2016.06.008

Summary of revisions: Tensor reduction of 1-loop integrals is extended to arbitrary rank and multiplicity with proper handling of integrals with zero Gram determinants. Tensor reduction of multi-loop integrals is now also available (except for cases with zero Gram determinants). Partial fractioning algorithm of [1] is added to decompose loop integrals into terms with linearly independent propagators. Feynman diagrams generated by FeynArts can be directly converted into FeynCalc input for subsequent evaluation.

Running time: Depends on the complexity of the calculation. Seconds for few simple tree level and 1-loop Feynman diagrams; Minutes or more for complicated diagrams.

References:

[1] F. Feng, $Apart: A Generalized Mathematica Apart Function, Comput. Phys. Commun., 183, 2158–2164, (2012), arXiv:1204.2314.
[2] T. Hahn, Generating Feynman Diagrams and Amplitudes with FeynArts 3, Comput. Phys. Commun., 140, 418–431, (2001), arXiv:hep-ph/0012260.

1. Introduction

In the last decades, the importance of computer tools for higher order perturbative calculations in quantum field theory (QFT) has increased tremendously. Indeed, some recent achievements [1–3] in this field would hardly have been possible to complete within a reasonable time frame, if such projects were to be carried out only by pen and paper. The question most QFT practitioners pose themselves today is not whether to use software tools or not, but rather which combination of tools will be the most efficient for the endeavored project. It is clear that, in principle, there can be no universal package to cover any demand of any particle theorist. Instead, specific programs that provide different level of automation should be used for specific tasks. One of such specific tools is FeynCalc [4], that recently was released in the version 9.0.

FeynCalc is a Mathematica package for algebraic calculations in QFT and semi-automatic evaluation of Feynman diagrams. The very first public version of FeynCalc, developed by Rolf Mertig with guidance from Ansgar Denner and Manfred Böhm, appeared in 1991. The main developments and improvements between 1991 and 1996 were triggered by the work of Mertig in electroweak theory [5–8] and perturbative QCD [9]. In 1998 Rolf Mertig and Rainer Scharf released TARCER [10], a Mathematica package that implements reduction of 2-loop propagator type integrals with arbitrary masses using O.V. Tarasov's recurrence algorithm [11,12]. Since then TARCER is a part of FeynCalc that can be loaded on-demand. Between 1997 and 2000, important contributions to the project came from Frederik Orellana, who, besides working on the general code, contributed the sub-package PHI for using FeynCalc in Chiral Perturbation Theory (χPT) [13] and interfacing to FeynArts 3 [14]. From 2001 until 2014, with both developers out of theoretical physics, the development of FeynCalc was mostly constrained to bug fixing and providing support through the mailing list,[1] although some interesting projects with external collaborators still were conducted [15]. In 2014, the developer team was joined by Vladyslav Shtabovenko, who started to work on rewriting some parts of the existing code and implementing new features. In the same year the source code repository of FeynCalc was moved to GitHub,[2] where the master branch of the repository represents the current development snapshot of the package. Not only the stable releases, but also the development version of FeynCalc can be anonymously downloaded by everyone at any time free of charge. The code is licensed under the General Public License (GPL) version 3. To minimize the number of new bugs and regressions, an extensive unit testing framework[3] with over 3000 tests was introduced.

This note is organized in the following way. Section 2 compares FeynCalc to other packages for automatic evaluation of 1-loop Feynman diagrams and discusses setups, in which FeynCalc can be particularly useful. Section 3 provides an overview of interesting new features and improvements in FeynCalc 9.0. Section 4 gives an example of using FeynCalc to determine matching coefficients in NRQCD [16], a non-relativistic effective field theory (EFT) for heavy quarkonia. Finally, we summarize and draw our conclusions in Section 5.

2. Comparison to similar tools

In view of the existence of several well-known symbolic packages (FormCalc [17], GoSam [18], FDC [19], GRACE [20], Diana [21]) that offer almost fully automatic evaluation of Feynman diagrams at 1-loop from Lagrangian to cross-section, it appears necessary to explain how FeynCalc differs from such tools and why it is useful.

FeynCalc by itself does not provide a fully automatic way of computing cross sections or decay rates. Indeed, FeynCalc cannot generate Feynman diagrams and has no built-in capabilities for the numerical evaluation of master integrals and for the phase space integration. Therefore, these two important steps should be done using other tools.

Second, FeynCalc normally performs all the algebraic manipulations using Mathematica. This leads to a slower performance when compared to tools that rely e.g. on FORM [22] for the symbolics. Despite some possibilities [15] to link FeynCalc with FORM, one should keep in mind that FeynCalc is not very well suited for evaluating hundreds, thousands or millions of Feynman diagrams.

Finally, FeynCalc does not impose any particular ordering in which different parts (Dirac matrices, $SU(N)$ matrices, loop integrals, etc.) of the amplitudes are supposed to be computed. It is always up to the user to decide what is the most useful way to carry out the calculation. This particular feature makes FeynCalc very different from tools that attempt to automatize all the steps of the evaluation process. Such tools usually stick to a particular workflow which roughly consists of the following steps:

1. The user specifies the process that needs to be computed.

[1] http://www.feyncalc.org/forum.
[2] https://github.com/FeynCalc.
[3] https://github.com/FeynCalc/feyncalc/tree/master/Tests.

2. If the given process is available in the standard configuration, load the corresponding model (e.g. Standard model (SM)). Otherwise the user must create a new model that contains this process.
3. Using the loaded model, generate relevant Feynman diagrams for the given process.
4. Evaluate the amplitudes by performing all the necessary algebraic simplifications.
5. Square the amplitude and sum/average over the spins of the involved particles.
6. Integrate over phase space.

In this list, already the second step might turn out to be problematic. The list of built-in models usually includes SM and some popular (e.g. SUSY inspired) extensions, while more exotic theories require custom model files to be added by the user. If the Lagrangian of such a theory looks very different from $\mathcal{L}_{\text{SM}}$ (e.g. in EFTs that are not strictly renormalizable (with an arbitrary number of legs in vertices) like χPT or even not manifestly Lorentz covariant like non-relativistic QCD (NRQCD) [16] or potential non-relativistic QCD (pNRQCD) [23]), then its implementation becomes a formidable task. On the other hand, even if the model can be implemented with a limited amount of effort, it still might cost more time than just writing down the amplitudes by hand and then manually entering them into the program. Although it is possible to make fully automatic tools accept such amplitudes as input, this is usually much less straightforward than the standard way of just specifying the process, launching the diagram generator and letting the automatics do the rest.

FeynCalc avoids such difficulties by accepting any kind of input that consists of valid FeynCalc objects. Hence, one can enter e.g. standalone Dirac traces, Lorentz vectors or loop integrals and then manipulate them with suitable FeynCalc functions. In this sense FeynCalc can be used much like a "calculator" for QFT expressions.

For manual input of Feynman diagrams FeynCalc contains some functions (`FeynRule`, `FunctionalD`, `CovariantD`, `QuantumField`, etc.) for deriving Feynman rules from Lagrangians that are manifestly Lorentz covariant. Furthermore, it is also possible to evaluate Feynman diagrams that were generated automatically (e.g. by FeynArts), so that the user always can choose the most efficient strategy to get the calculation done.

Steps 4 and 5 usually imply that the user is not supposed to interfere too much with the evaluation process. Instead, one should rely on the available options to influence the outcome of the calculation. For example, when an automatic tool handles the Dirac algebra, it would normally try to simplify everything it can. While in general, this approach is perfectly fine, sometimes one would like to simplify only some of the Dirac structures, leaving the others (e.g. all the traces involving an odd number of γ^5) untouched. In principle, provided that the particular tool is open source, one can always modify its code accordingly to obtain the desired output. Depending on the complexity of the code and the amount of documentation, this might, however, take some time and even introduce new bugs.

With FeynCalc, the same result can be achieved in a more simple way, as one always has full access to all kinds of intermediate expressions. For this purpose FeynCalc also provides various helper functions (e.g. `Collect2`, `Expand2`, `Factor2`, `Isolate`, `ExpandScalarProduct`, `DiracGammaExpand`, `MomentumCombine`, `FCLoopSplit`, `FCLoopIsolate`, `FCLoopExtract`) that can be used to expand, sort, abbreviate and collect the given expressions with respect to particular structures.

Thus we see that FeynCalc should not be regarded as a direct competitor to highly automatized packages like e.g. FormCalc, because it neither provides routines for numerical evaluation nor offers a fully automatic workflow to evaluate a scattering process.

For studies that can be carried out using an automatic tool from the beginning to the end, it obviously would not be very efficient to stick to FeynCalc. While one certainly can chain FeynCalc with appropriate tools and libraries to obtain the same result, this would require more time and effort — which could be invested elsewhere.

There are indeed also other publicly available software packages (HEPMath [24] and Package-X [25]) that follow the semi-automatic approach to QFT calculations and exhibit many similarities to FeynCalc. Therefore, let us provide a short comparison between those two packages and FeynCalc.

Just as FeynCalc, HEPMath is an open-source project licensed under GPL version 3. This way the users are able to study the source code and possibly modify HEPMath to suit their specific needs. Package-X is on the contrary distributed as a closed source freeware. It can be downloaded directly from the project homepage, but the source code is encrypted which prevents any possible modifications or extensions by the user. Like FeynCalc both HEPMath and Package-X can manipulate standalone expressions that do not need to represent a valid Feynman diagram. This is done by providing special functions for index contractions, treatment of Dirac algebra and manipulations of loop integrals. In D-dimensions all three packages can work with anticommuting γ^5, but only HEPMath and FeynCalc can also treat γ^5 using the Breitenlohner–Maison–t'Hooft–Veltman scheme [26,27]. Tensor reduction of 1-loop integrals via Passarino–Veltman technique [28] and subsequent simplification of Passarino–Veltman coefficient functions are implemented in each package. FeynCalc 9 and Package-X can work with arbitrarily high-rank tensor integrals, but HEPMath is currently limited to rank 4. As far as the color algebra in concerned, while FeynCalc can deal with general $SU(N)$ generators, HEPMath only supports $SU(3)$ and Package-X does not offer any routines for working with color structures. Both FeynCalc and HEPMath provide an interface to FeynArts for generating Feynman diagrams. HEPMath, however, also contains built-in interfaces to LoopTools [17] and LHAPDF [29] that are missing in the two other packages. On the other hand, Package-X comes with a library of explicit *analytic* results for 1-, 2- and 3-point Passarino–Veltman functions with almost arbitrary kinematics. This is a very useful feature that is, to our knowledge, not present in any other automatic or semi-automatic packages.

The above comparison shows that all three packages have somewhat different capabilities, but follow essentially the same philosophy to provide the user with convenient and flexible tools for doing calculations in QFT. Since HEPMath and Package-X were released quite recently, they are still not so widely used in research as FeynCalc. However, as future versions of these packages will likely introduce new useful features and improvements, they will also become more visible in the high energy physics community and thus further promote the idea of using semi-automatic tools in suitable computations.

The niche that FeynCalc often fills is calculations that are too specific to be done in a fully automatic fashion but also too challenging to be done (only) by pen and paper, so that semi-automatic evaluation is very welcome.

One example for such problems is the determination of matching coefficients in EFTs. Matching coefficients are extracted by comparing suitable quantities (e.g. Green's functions) between the higher energy theory and its EFT at energies, where both theories should agree by construction. Then the quantity in the higher energy theory usually needs to be expanded in small scales and massaged into a form that resembles the same quantity in the lower energy theory, so that one can read off the values of the matching coefficients.

Such calculations are usually too special to be automatized in a full generality, but they can benefit a lot from functions provided by FeynCalc. This is one of the reasons, why FeynCalc enjoys

certain popularity in the heavy quarkonium physics community, where it is used to perform the matching between QCD and NRQCD for production [30–32] or decay [33,34] of heavy quarkonia. Other studies where FeynCalc was used involve such fields as Higgs [35–37] and top quark physics [38], leptonic decays [39] phenomena in hadronic interactions [40,41], dark matter [42,43], neutrino physics [44–46] and gravity [47]. It is worth noticing that FeynCalc was also used at some stages of NNLO [48,49] calculations. Indeed, FeynCalc can be well employed for small or medium size multi-loop processes if one connects it to suitable tools for IBP-reduction (e.g. FIRE [50]) and numeric evaluation of multi-loop integrals (e.g. FIESTA [51] or SecDec [52]).

Last but not least, FeynCalc can be also useful for educational purposes. The possibility of easily getting hands-on experience with computing Feynman diagrams and exploring the different steps involved can be very helpful and motivating for students of quantum field theory.

3. New features in FeynCalc 9.0

3.1. Improved tensor decomposition

In the very early versions of FeynCalc, tensor decomposition of 1-loop integrals (via Passarino–Veltman technique [28]) could be done only using the function `OneLoop`, where the maximal rank of the integrals was limited to 4 and the output was always written in terms of Passarino–Veltman coefficient functions. Although one could reduce Passarino–Veltman coefficient functions with rank higher than 4 using `PaVeReduce`, the tensor basis for such higher rank integrals had to be constructed by hand.

While working on [9], Rolf Mertig added to FeynCalc 3.0 a tool (`Tdec`) for tensor decomposition of multi-loop integrals of arbitrary rank and multiplicity (for non-zero Gram determinants) and even included a database (`TIDL`) to load already computed decompositions, but only a very small amount of this functionality was turned into a user-friendly routine `TID` (1-loop only), while the rest remained to "lie idle" in the source code. `TID` was limited to 4-point functions of rank 4 and could not handle kinematic configurations with zero Gram determinants, so that for such cases one was forced to use `OneLoop`. However, in FeynCalc 4 the reduction of rank 4 tensor integrals via `OneLoop` was disabled due to its poor efficiency. As a consequence of all these developments the tensor reduction of 1-loop integrals (especially with rank higher than 3) in the recent FeynCalc versions often turned to be cumbersome and inconvenient.

In FeynCalc 9.0 `TID` was rewritten almost from scratch to allow for 1-loop tensor decompositions of any rank and multiplicity. At the beginning, the function computes Gram determinants for all the unique 1-loop integrals in the expression. If the determinant vanishes, the decomposition for that integral is done in terms of the Passarino–Veltman coefficient functions.

```
In[1]:= FCClearScalarProducts[];
        ScalarProduct[p1, p1] = 0;
        int = FCI[SPD[p2, q] FAD[{q, m0}, {q + p1, m1}]]
```

$$\text{Out[1]:= } \frac{\text{p2}\cdot q}{\left(q^2-\text{m0}^2\right).\left((\text{p1}+q)^2-\text{m1}^2\right)}$$

```
In[2]:= TID[int, q]
```

$$\text{Out[2]:= } i\pi^2(\text{p1}\cdot\text{p2})\,\text{B}_1\left(0,\text{m0}^2,\text{m1}^2\right)$$

Otherwise, `TID` will output the result in terms of scalar 1-loop integrals.

```
In[1]:= FCClearScalarProducts[];
        int = FCI[SPD[p2, q] FAD[{q, m0}, {q + p1, m1}]]
```

$$\text{Out[1]:= } -\frac{\left(\text{m0}^2-\text{m1}^2+\text{p1}^2\right)(\text{p1}\cdot\text{p2})}{2\text{p1}^2\left(q^2-\text{m0}^2\right).\left((q-\text{p1})^2-\text{m1}^2\right)}+\frac{\text{p1}\cdot\text{p2}}{2\text{p1}^2\left(q^2-\text{m0}^2\right)}-\frac{\text{p1}\cdot\text{p2}}{2\text{p1}^2\left(q^2-\text{m1}^2\right)}$$

If needed, those scalar integrals can be converted to Passarino–Veltman scalar functions by using `ToPaVe`, which is also available since FeynCalc 9.0.

```
In[2]:= TID[int, q] // ToPaVe[#, q] &
```

$$\text{Out[2]:= } -\frac{i\pi^2\left(\text{m0}^2-\text{m1}^2+\text{p1}^2\right)(\text{p1}\cdot\text{p2})\,\text{B}_0\left(\text{p1}^2,\text{m0}^2,\text{m1}^2\right)}{2\text{p1}^2}+\frac{i\pi^2\,\text{A}_0\left(\text{m0}^2\right)(\text{p1}\cdot\text{p2})}{2\text{p1}^2}-\frac{i\pi^2\,\text{A}_0\left(\text{m1}^2\right)(\text{p1}\cdot\text{p2})}{2\text{p1}^2}$$

The decompositions in terms of scalar integrals tend to become very large already for 3-point functions, so to obtain more compact expressions it might be desirable to use the basis of Passarino–Veltman coefficient functions, even if there are no zero Gram determinants. This can be easily achieved via the option `UsePaVeBasis`.

```
In[1]:= int = FCI[FVD[q, mu] FVD[q, nu] FAD[{q, m0}, {q + p1, m1}, {q + p2, m2}]]
```

$$\text{Out[1]:= } \frac{g^{\text{mu}\,\text{nu}}q^{\text{mu}}q^{\text{nu}}}{\left(q^2-\text{m0}^2\right).\left((\text{p1}+q)^2-\text{m1}^2\right).\left((\text{p2}+q)^2-\text{m2}^2\right)}$$

```
In[2]:= TID[int/(I*Pi^2), q, UsePaVeBasis -> True]
```

$$\begin{aligned}\text{Out[2]:= } & g^{\text{mu}\,\text{nu}}\text{C}_{00}\left(\text{p1}^2,-2(\text{p1}\cdot\text{p2})+\text{p1}^2+\text{p2}^2,\text{p2}^2,\text{m0}^2,\text{m1}^2,\text{m2}^2\right)\\ & +\text{p1}^{\text{mu}}\text{p1}^{\text{nu}}\text{C}_{11}\left(\text{p1}^2,-2(\text{p1}\cdot\text{p2})+\text{p1}^2+\text{p2}^2,\text{p2}^2,\text{m0}^2,\text{m1}^2,\text{m2}^2\right)\\ & +\left(\text{p2}^{\text{mu}}\text{p1}^{\text{nu}}+\text{p1}^{\text{mu}}\text{p2}^{\text{nu}}\right)\text{C}_{12}\left(\text{p1}^2,-2(\text{p1}\cdot\text{p2})+\text{p1}^2\right.\\ & \left.+\,\text{p2}^2,\text{p2}^2,\text{m0}^2,\text{m1}^2,\text{m2}^2\right)\\ & +\text{p2}^{\text{mu}}\text{p2}^{\text{nu}}\text{C}_{22}\left(\text{p1}^2,-2(\text{p1}\cdot\text{p2})+\text{p1}^2+\text{p2}^2,\text{p2}^2,\text{m0}^2,\text{m1}^2,\text{m2}^2\right)\end{aligned}$$

All the Passarino–Veltman functions are defined as in LoopTools [17] and explicit definitions are encoded for functions with up to 5 legs. For integrals with even higher multiplicities the coefficient functions (denoted as `GenPaVe`) simply include the dependence on the external momenta that can be used to convert them to the LoopTools or any other convention.

```
In[1]:= int = FCI[FVD[q, mu] FVD[q,nu] FAD[{q, m0}, {q, m1}, {q, m2}, {q, m3}, {q +
          p4, m4}, {q + p5, m5}, {q + p6, m6}]]
```

$$\begin{aligned}\text{Out[1]:= } & \left(q^{\text{mu}}q^{\text{nu}}\right)/\left(q^2-\text{m0}^2\right).\left(q^2-\text{m1}^2\right).\left((\text{p2}+q)^2-\text{m2}^2\right).\\ & \left((\text{p3}+q)^2-\text{m3}^2\right).\\ & \left((\text{p4}+q)^2-\text{m4}^2\right).\left((\text{p5}+q)^2-\text{m5}^2\right).\left((\text{p6}+q)^2-\text{m6}^2\right)\end{aligned}$$

```
In[2]:= TID[int/(I*Pi^2), q, UsePaVeBasis -> True]
```

$$\begin{aligned}\text{Out[2]:= } & g^{\text{mu}\,\text{nu}}\,\text{GenPaVe}\left(\{0,0\},\begin{pmatrix}0&\text{m0}\\ \text{p1}&\text{m1}\\ \text{p2}&\text{m2}\\ \text{p3}&\text{m3}\\ \text{p4}&\text{m4}\\ \text{p5}&\text{m5}\\ \text{p6}&\text{m6}\end{pmatrix}\right)\\ & +\text{p1}^{\text{mu}}\text{p1}^{\text{nu}}\,\text{GenPaVe}\left(\{1,1\},\begin{pmatrix}0&\text{m0}\\ \text{p1}&\text{m1}\\ \text{p2}&\text{m2}\\ \text{p3}&\text{m3}\\ \text{p4}&\text{m4}\\ \text{p5}&\text{m5}\\ \text{p6}&\text{m6}\end{pmatrix}\right)+\ldots\end{aligned}$$

Here, `GenPaVe[{1,1},{{0,m0},{Momentum[p1],m1}, ..., {Momentum[p6],m6}}]` stands for the coefficient function of $p_1^\mu p_1^\nu$ in the tensor decomposition of

$$\int d^D q \times \frac{q^\mu q^\nu}{[q^2-m_0^2][(q-p_1)^2-m_1^2][(q-p_2)^2-m_2^2]\cdots[(q-p_6)^2-m_6^2]}. \quad (1)$$

Since this kind of output is useful if one explicitly wants to obtain coefficient functions defined in a different way than in LoopTools, it can be activated also for functions with lower multiplicities by setting the option `GenPaVe` of `TID` to `True`.

```
In[1]:= int = FCI[FVD[q, mu] FVD[q, nu] FAD[{q, m0}, {q + p1, m1}]]

Out[1]= q^mu q^nu / ((q^2 - m0^2).((p1+q)^2 - m1^2))

In[2]:= TID[int/( I*Pi^2), q, UsePaVeBasis -> True, GenPaVe -> True]

Out[2]= g^munu GenPaVe({0, 0}, ( 0  m0 ))
                               ( p1 m1 )
        + p1^mu p1^nu GenPaVe({1, 1}, ( 0  m0 ))
                                      ( p1 m1 )
```

One should also keep in mind that FeynCalc cannot perform any further simplifications of `GenPaVe` functions, because internally they are not recognized as Passarino–Veltman integrals (`PaVe`).

It is well known that for a general multi-loop multi-scale integral, tensor decomposition does not allow to cancel all the scalar products containing loop momenta in the numerator, as it is the case at 1-loop. Nevertheless, this technique is widely used also in calculations beyond 1-loop, especially if one needs to deal with integrals that have loop momenta contracted to Dirac matrices or epsilon tensors or even loop momenta with free Lorentz indices. FeynCalc uses a special reduction algorithm (implemented in `Tdec`) that consists of decomposing the integral into all tensor structures allowed by the symmetries and using a modified version of Gaussian elimination to obtain the coefficients of each tensor.

Since tensor decomposition of multi-loop integrals with FeynCalc's function `Tdec` is not very straightforward and usually requires some additional Mathematica code, in FeynCalc 9.0 a new function `FCMultiLoopTID` was added, that makes multi-loop tensor reduction work out of the box.

```
In[1]:= int = FCI[FVD[q1, mu] FVD[q2, nu] FAD[q1, q2, {q1 - p1},
        {q2 - p1}, {q1 - q2}]]

Out[1]= q1^mu q2^nu / (q1^2.q2^2.(q1-p1)^2.(q2-p2)^2.(q1-q2)^2)

In[2]:= FCMultiLoopTID[int, {q1, q2}]

Out[2]= D p1^mu p1^nu - p1^2 g^munu / (4(D-1)q2^2.q1^2.(q2-p1)^2.(q1-q2)^2.(q1-p1)^2)
        - D p1^mu p1^nu - p1^2 g^munu / (2(D-1)p1^4 q1^2.(q2-p1)^2.(q1-q2)^2)
        + p1^2 g^munu - p1^mu p1^nu / ((D-1)p1^2 q2^2.q1^2.(q1-q2)^2.(q1-p1)^2)
        - p1^2 g^munu - p1^mu p1^nu / (2(D-1)p1^2 q2^2.q1^2.(q2-p1)^2.(q1-p1)^2)
```

Unfortunately, the reduction breaks down when the corresponding Gram determinant vanishes. For such cases, in a future version it is planned to include a more useful algorithm.

3.2. New partial fractioning algorithm

Since the version 3, FeynCalc includes `ScalarProductCancel` and `Apart2` that can be used to rewrite loop integrals in a simpler form. `ScalarProductCancel` essentially applies the well known identity [28]

$$q \cdot p = \frac{1}{2}[(q+p)^2 + m_2^2 - (q^2 + m_1^2) - p^2 - m_2^2 + m_1^2] \tag{2}$$

repeatedly, until all scalar products containing loop momenta that can be canceled in this way are eliminated. `Apart2` uses the trivial identity

$$\frac{1}{(q^2 - m_1^2)(q^2 - m_2^2)} = \frac{1}{m_1^2 - m_2^2}\left(\frac{1}{q^2 - m_1^2} - \frac{1}{q^2 - m_2^2}\right) \tag{3}$$

to simplify suitable denominators. In principle, these two functions implement some aspects of partial fractioning, i.e., the decomposition of a loop integral with linearly dependent propagators into a sum of integrals where each integral contains only linearly independent propagators. Notice that here we count scalar products that involve loop momenta as propagators with negative exponents. Unfortunately, there are plenty of examples where neither `ScalarProductCancel` nor `Apart2` can partial fraction an integral with linearly dependent propagators, e.g.

$$\int d^D q \, \frac{1}{q^2 (q-p)^2 (q+p)^2} = \frac{1}{p^2}\int d^D q \left(\frac{1}{q^2 (q-p)^2} - \frac{1}{(q-p)^2 (q+p)^2}\right). \tag{4}$$

A general partial fractioning algorithm that is suitable for multi-loop integrals including its Mathematica implementation (APart[4]) was presented in [53]. The author has also shown how his code can be used together with FeynCalc in order to decompose different loop integrals. For this the user is required to convert a loop integral in the FeynCalc notation (with denominator encoded in `FeynAmpDenominator`) to a somewhat different form and to specify all the scalar products that contain loop momenta and appear in this loop integral. After the decomposition the resulting integrals need to be converted back into FeynCalc notation.

In FeynCalc 9.0 the algorithm from [53] was adopted and reimplemented to be the standard partial fractioning routine. As such, it is fully integrated with all other FeynCalc functions and objects and does not require any explicit conversion of the input or output.

```
In[1]:= int = FAD[{q}, {q - p}, {q + p}]

Out[1]= 1 / (q^2.(q-p)^2.(p+q)^2)

In[2]:= ApartFF[int1, {q}]

Out[2]= 1 / (p^2 q^2.(q-p)^2) - 1 / (p^2 q^2.(q-2p)^2)
```

The name of the corresponding function is `ApartFF` which stands for "Apart Feng Feng" and serves as an additional acknowledgment of the original author. One should also notice that while the original APart can be used for partial fractioning of general multivariate polynomials, the FeynCalc version is limited only to polynomials that appear in Feynman diagrams as propagators and scalar products. Thus, it is much less general than APart but is also more convenient when used with FeynCalc.

3.3. Tools for interfacing FeynCalc with packages for IBP-reduction

In modern multi-loop calculations, reduction of scalar loop integrals via integration-by-parts (IBP) identities [54] is a regular step needed to arrive to a smaller set of master integrals.

Although FeynCalc does not include a general purpose tool for IBP reduction (the built-in TARCER [10] is suitable only for 2-loop self-energy type integrals), this omission can be compensated by using one of the publicly available IBP-packages (FIRE [50], LiteRED [55], Reduze [56], AIR [57]). However, one should keep in mind that such tools usually expect their input to contain only loop integrals with linearly independent propagators that form a basis. For example, the integral

$$\int d^D q_1 \, d^D q_2 \, d^D q_3 \, \frac{1}{[q_1^2 - m^2]^2 [(q_1+q_3)^2 - m^2](q_2 - q_3)^2 q_2^2} \tag{5}$$

cannot be processed by FIRE in this form because q_1^2, q_2^2, $(q_1+q_3)^2$ and $(q_2-q_3)^2$ alone do not form a basis.

[4] https://github.com/F-Feng/APart.

```
In[1]:= << FIRE5`FIRE5`
        Internal = {q1, q2, q3};
        External = {};
        Propagators = {q1^2 - m^2, (q1 + q3)^2 - m^2, (q2 - q3)^2, q2^2};
        PrepareIBP[];

Out[1]:= FIRE, version 5.1
         DatabaseUsage: 0
         UsingFermat: False
         Not enough propagators. Add irreducible nominators
```

If one includes also q_3^2 and $q_1 \cdot q_2$ with zero exponentials, then we have a proper basis and the reduction works as it should. Also the integral

$$\int d^D q_1\, d^D q_2\, \frac{(p\cdot q_1)^2 (p\cdot q_2)^2}{[q_1^2 - m^2][q_2^2 - m^2](q_1 - p)^2 (q_2 - p)^2 (q_1 - q_2)^2} \tag{6}$$

cannot be reduced right away, this time because its propagators are linearly dependent.

```
In[1]:= << FIRE5`FIRE5`
        Internal = {q1, q2};
        External = {};
        Propagators = {q1^2 - m^2, q2^2 - m^2, (q1 - p)^2, (q2 - p)^2, (q1 - q2
              )^2, p q1, p q2};
        PrepareIBP[];

Out[1]:= FIRE, version 5.1
         DatabaseUsage: 0
         UsingFermat: False
         Linearly dependant propagators. Perform reduction first
```

To detect such problems before the reduction actually fails, FeynCalc 9.0 introduces two new special functions. When `FCLoopBasisIncompleteQ` is applied to a loop integral, it returns `True` if this integral does not contain enough irreducible propagators.

```
In[1]:= intP1 = FCI[FAD[{q1, m, 2}, {q1 + q3, m}, {q2 - q3}, q2]]

Out[1]= 1/((q1^2-m^2).(q1^2-m^2).((q1+q3)^2-m^2).(q2-q3)^2.q2^2)

In[2]:= FCLoopBasisIncompleteQ[intP1, {q1, q2, q3}]
Out[2]= True

In[3]:= FCLoopBasisIncompleteQ[SPD[q3, q3] SPD[q1, q2] intP1, {q1, q2, q3}]
Out[3]= False
```

An integral with linearly dependent propagators will be detected by `FCLoopBasisOverdeterminedQ`,

```
In[1]:= intP2 = FCI[SPD[p, q1]^2 SPD[p, q2]^2 FAD[{q1, m}, {q2, m}, q1 - p, q2 - p,
        q1 - q2]]

Out[1]= (p.q1)^2 (p.q2)^2/((q1^2-m^2).(q2^2-m^2).(q1-p)^2.(q2-p)^2.(q1-q2)^2)

In[2]:= FCLoopBasisOverdeterminedQ[intP2, {q1, q2, q3}]
Out[2]= True
```

so that only an integral for which both functions return `False` can be reduced in a straightforward way.

In a practical calculation where one knows what integral topologies are involved, such issues can be easily resolved. In particular, a clever choice of additional propagators that are needed to have a basis, can greatly simplify the reduction. On the other hand, depending on the size of the problem and the number of topologies involved, a less clever but fully automatic solution may also be useful.

For an integral with linearly dependent propagators we can use `ApartFF`, that is guaranteed to decompose it into integrals where all propagators are linearly independent.

```
In[1]:= ApartFF[intP2, {q1, q2}]

Out[1]= (m^2+p^2)^4/(16(q1^2-m^2).(q2^2-m^2).(q2-p)^2.(q1-q2)^2.(q1-p)^2)
        - (m^2+p^2)^3/(8(q1^2-m^2).(q2^2-m^2).(q1-q2)^2.(q1-p)^2)
        - (m^2+p^2)(p.q1)/(8(q2^2-m^2).(q1-q2)^2.(q1-p)^2) + ...
```

For integrals with an incomplete basis of propagators one can use the new function `FCLoopBasisFindCompletion` that finds out which irreducible propagators (with zero exponents) are missing.

```
In[1]:= FCLoopBasisFindCompletion[intP1, {q1, q2, q3}]

Out[1]= {1/((q1^2-m^2).(q1^2-m^2).((q1+q3)^2-m^2).(q2-q3)^2.q2^2), {-(q1.q3)
        + q2.q3 + 2q3^2, q1.q2}}
```

With the suggested propagators the integral is guaranteed to have a complete basis, but the choice of the propagators themselves is usually not very clever. This is because in general FeynCalc cannot guess the topology of the given integral without any additional input. It is planned to provide a possibility for specifying the topology, which would admittedly make `FCLoopBasisFindCompletion` much more useful than it is now.

Still, with `ApartFF` and `FCLoopBasisFindCompletion` it is now possible to automatically bring any scalar multi-loop integral in FeynCalc notation to a form that can be directly (modulo notation conversion) forwarded to an IBP tool.

3.4. Advanced extraction of loop integrals

The idea to use FeynCalc as a sort of switch board for different computational tools in a larger framework (see e.g. [58]) is further developed in version 9.0 by the introduction of new functions that can extract different loop integrals from the given expression.

One of them, `FCLoopSplit`, breaks the given expression into four pieces, which are

1. Terms that contain no loop integrals.
2. Terms that only contain scalar loop integrals without any loop momenta in the denominators, e.g.
$$\int d^D q\, \frac{1}{q^2 - m^2}. \tag{7}$$
3. Terms that contain scalar loop integrals with loop momenta dependent scalar products in the denominators, e.g.
$$\int d^D q\, \frac{(q\cdot p)}{q^2 (q-p)^2}. \tag{8}$$
4. Terms that contain tensor loop integrals, e.g.
$$\int d^D q\, \frac{q^\mu q^\nu}{q^2 - m^2} \quad \text{or} \quad \int d^D q\, \frac{(\gamma\cdot q)}{q^2 (q-p)^2}. \tag{9}$$

```
In[1]:= int = FCI[(GSD[q - p] + m).GSD[x] FAD[q, {q - p, m}] + (m^2 + SPD[q, q])
        FAD[{q, m,2}]];
Out[1]= (m+γ.x.(q-p)).(γ.x)/(q^2.((q-p)^2-m^2)) + (m^2+q^2)/((q^2-m^2).(q^2-m^2))

In[2]:= FCLoopSplit[int, {q}]

Out[2]= {0, (mγ.x-(γ.p).(γ.x))/(q^2.((q-p)^2-m^2)) + m^2/((q^2-m^2).(q^2-m^2)),
        q^2/((q^2-m^2).(q^2-m^2)), (γ.q).(γ.x)/(q^2.((q-p)^2-m^2))}
```

This splitting makes it easier to handle different types of loop integrals and to simplify them with FeynCalc or other tools. For example, if one wants to perform tensor reduction of multi-loop integrals with FaRe [59] instead of `FCMultiLoopTID`, it can be done by applying `FCLoopSplit` to the given expression and working with the fourth element of the resulting list, while the

other elements remain unchanged and can be later added to the final expression.

To handle a larger number of loop diagrams in an efficient way, FCLoopSplit alone is not sufficient. This is because same integrals may appear multiple times in different diagrams and ignoring this fact would make the evaluation more complex than it actually is. To avoid this kind of problems one should better first analyze the amplitude and extract all the unique integrals. Then each unique integral needs to be evaluated only once, no matter how often it appears in the full expression. In FeynCalc 9.0 this can be conveniently done with FCLoopIsolate. The function wraps loop integers with the given head, such that the list of unique integrals can be quickly created with Mathematica's Cases and Union or just FeynCalc's Cases2

```
In[1]:= int = FCI[GSD[q - p1].(GSD[q - p2] + M).GSD[p3] SPD[q, p2] FAD[q, q - p1, {q - p2, m}]];
```

Out[1]:= $\frac{(\text{p2}\cdot q)(\gamma\cdot(q-\text{p1}))\cdot(M+\gamma\cdot(q-\text{p2}))\cdot(\gamma\cdot\text{p3})}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}$

```
In[2]:= res = FCLoopIsolate[int, {q}, Head -> loopInt]
```

Out[2]:= $\text{loopInt}\left(\frac{\text{p2}\cdot q}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right)((\gamma\cdot\text{p1})\cdot(\gamma\cdot\text{p2})\cdot(\gamma\cdot\text{p3})$

$-M(\gamma\cdot\text{p1})\cdot(\gamma\cdot\text{p3}))+$

$M\,\text{loopInt}\left(\frac{(\text{p2}\cdot q)(\gamma\cdot q)\cdot(\gamma\cdot\text{p3})}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right)-\text{loopInt}\left(\frac{(\text{p2}\cdot q)(\gamma\cdot\text{p1})\cdot(\gamma\cdot q)\cdot(\gamma\cdot\text{p3})}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right)$

$-\text{loopInt}\left(\frac{(\text{p2}\cdot q)(\gamma\cdot q)\cdot(\gamma\cdot\text{p2})\cdot(\gamma\cdot\text{p3})}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right)+\text{loopInt}\left(\frac{(\text{p2}\cdot q)(\gamma\cdot q)\cdot(\gamma\cdot q)\cdot(\gamma\cdot\text{p3})}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right)$

```
In[3]:= Cases2[res, loopInt]
```

Out[3]:= $\left\{\text{loopInt}\left(\frac{\text{p2}\cdot q}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right),\ \text{loopInt}\left(\frac{(\text{p2}\cdot q)(\gamma\cdot q)\cdot(\gamma\cdot\text{p3})}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right),\right.$

$\text{loopInt}\left(\frac{(\text{p2}\cdot q)(\gamma\cdot\text{p1})\cdot(\gamma\cdot q)\cdot(\gamma\cdot\text{p3})}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right),\ \text{loopInt}\left(\frac{(\text{p2}\cdot q)(\gamma\cdot q)\cdot(\gamma\cdot\text{p2})\cdot(\gamma\cdot\text{p3})}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right),$

$\left.\text{loopInt}\left(\frac{(\text{p2}\cdot q)(\gamma\cdot q)\cdot(\gamma\cdot q)\cdot(\gamma\cdot\text{p3})}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right)\right\}$

The combined application of FCLoopIsolate and FCLoopSplit is provided by FCLoopExtract. This function returns a list of three entries. The first one contains the part of the expression which is free of loop integrals. The second entry consists of the remaining expression where every loop integral is wrapped with the given head. Finally, the last entry contains a list of all the unique loop integrals in the expression.

```
In[4]:= FCLoopExtract[int, {q}, loopInt][[1]]
```

Out[4]:= 0

```
In[5]:= FCLoopExtract[int, {q}, loopInt][[2]] ===
 FCLoopIsolate[int, {q}, Head -> loopInt]
```

Out[5]:= True

```
In[6]:= FCLoopExtract[int, {q}, loopInt][[3]] ===
 Cases2[res, loopInt]
```

Out[6]:= True

Suppose that we want to evaluate these loop integrals using some custom function loopEval (in this example it is just a dummy function that computes the hash of each loop integral). All we need to do is to apply FCLoopExtract to the initial expression, map the list of the unique integrals to loopEval, create a substitution rule and apply this rule to our expression in order to get the final result.

```
In[7]:= {rest, loops, intsUnique} = FCLoopExtract[int, {q}, loopInt];

In[8]:= loopEval[x_] := ToString[Hash[x]];

In[9]:= solsList = loopEval /@ uniqueInts

Out[9]:= {2069116068,115167616,776830638,1878762839,1337833147}

In[10]:= repRule = MapThread[Rule[#1, #2] &, {intsUnique, solsList}]
```

Out[10]:= $\left\{\text{loopInt}\left(\frac{\text{p2}\cdot q}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right)\to 2069116068,\right.$

$\text{loopInt}\left(\frac{(\text{p2}\cdot q)(\gamma\cdot q)\cdot(\gamma\cdot\text{p3})}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right)\to 115167616,$

$\text{loopInt}\left(\frac{(\text{p2}\cdot q)(\gamma\cdot\text{p1})\cdot(\gamma\cdot q)\cdot(\gamma\cdot\text{p3})}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right)\to 776830638,$

$\text{loopInt}\left(\frac{(\text{p2}\cdot q)(\gamma\cdot q)\cdot(\gamma\cdot\text{p2})\cdot(\gamma\cdot\text{p3})}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right)\to 1878762839,$

$\left.\text{loopInt}\left(\frac{(\text{p2}\cdot q)(\gamma\cdot q)\cdot(\gamma\cdot q)\cdot(\gamma\cdot\text{p3})}{q^2\cdot(q-\text{p1})^2\cdot((q-\text{p2})^2-m^2)}\right)\to 1337833147\right\}$

```
In[11]:= res = rest + loops /. repRule
```

Out[11]:= $115167616M + 1337833147 - 1878762839+$

$2069116068((\gamma\cdot\text{p1})\cdot(\gamma\cdot\text{p2})\cdot(\gamma\cdot\text{p3})$

$-M(\gamma\cdot\text{p1})\cdot(\gamma\cdot\text{p3}))-776830638$

With FCLoopSplit, FCLoopIsolate and FCLoopExtract it is now much easier not only to manipulate loop integrals, but also to check which integrals actually appear in an expression. Unique loop integrals can be evaluated with tools outside of FeynCalc and then substituted back by just a couple of lines of Mathematica code.

3.5. *Better interface to* FeynArts

If FeynCalc needs to be used with a Feynman diagram generator, then FeynArts is usually the most convenient choice. Initially the syntax of both packages was adjusted to make them fully compatible with each other. In fact, for the very first version of FeynArts [60], FeynCalc was referred to as the standard tool to evaluate the generated amplitudes. As FeynArts was developed further, the full compatibility was lost, but even now, the output of FeynArts can be converted into valid FeynCalc input with only little effort. A more severe problem in using this setup arises when FeynArts and FeynCalc are loaded in the same Mathematica session. Unfortunately, both packages contain objects with same names but different contexts, definitions and properties (e.g. FourVector, DiracMatrix or FeynAmpDenominator) such that it is not possible to use them together without risking inconsistencies. To avoid these issues FeynCalc is able to automatically patch the source code of FeynArts by renaming all the conflicting symbols, such that e.g. FourVector becomes FAFourVector and no variable shadowing can occur. This patching mechanism was greatly improved in FeynCalc 9.0 both in terms of user friendliness and compatibility to other Mathematica packages. The patched copy of FeynArts now resides in the *FeynArts* directory inside the FeynCalc installation. By default this directory is empty. The user is expected to manually download the latest FeynArts tarball from the official website[5] and unpack its content to *FeynCalc/FeynArts*. When FeynCalc is loaded via

```
$LoadFeynArts=True;
<<FeynCalc`
```

[5] http://www.feynarts.de.

it will automatically detect FeynArts installation and offer the user to patch it. This procedure is required only once and after that one can use FeynArts and FeynCalc together without any problems.

After all the required diagrams have been generated and turned into amplitudes with FeynArts' function `CreateFeynAmp`, the output still needs to be converted into valid FeynCalc input. In FeynCalc 9.0 this is handled by the new function `FCFAConvert` that takes the output of `CreateFeynAmp` and generates proper FeynCalc expressions based on the given options. With `IncomingMomenta`, `OutgoingMomenta` and `LoopMomenta` the user can specify how the corresponding momenta should be named. Otherwise they will be denoted as `InMom1`, `InMom2` , . . . , `OutMom1`, `OutMom2` , . . . and `LoopMom1`, `LoopMom2` , ... Polarization vectors of external massless bosons are by default not transverse, but can be made so if the momenta of the bosons are listed in `TransversePolarizationVectors`. The splitting of fermion–fermion–boson couplings into left and right handed chirality projectors (default in FeynArts) can be undone with the option `UndoChiralSplittings`. For example, the amplitude for the tree level process $\gamma^* u \to u g$ is obtained via

```
In[1]:= $LoadFeynArts = True;
        $FeynCalcStartupMessages = False;
        << FeynCalc`;
        $FAVerbose = 0;

In[2]:= diags = InsertFields[ CreateTopologies[0, 2 -> 2], {F[3, {1}],
         V[1]} -> {V[5], F[3, {1}]},   InsertionLevel -> {Classes},
         Model -> "SMQCD"];

In[3]:= FCFAConvert[CreateFeynAmp[diags], IncomingMomenta -> {p1, kp},
        OutgoingMomenta -> {kg, p2}, UndoChiralSplittings -> True,
        TransversePolarizationVectors -> {kg}, DropSumOver -> True,
         List -> False] // Contract
```

$$\text{Out[3]:= } -\frac{2\text{EL}g_s T^{\text{Glu3}}_{\text{Col4Col1}}\left(\varphi(\overline{p2},\text{MU})\right).\left(\bar{\gamma}\cdot\bar{\varepsilon}^*(\text{kg})\right).\left(\bar{\gamma}\cdot\left(\overline{\text{kg}}+\overline{p2}\right)+\text{MU}\right).\left(\bar{\gamma}\cdot\bar{\varepsilon}(\text{kp})\right).\left(\varphi(\overline{p1},\text{MU})\right)}{3\left((-\text{kg}-p2)^2-\text{MU}^2\right)}$$
$$-\frac{2\text{EL}g_s T^{\text{Glu3}}_{\text{Col4Col1}}\left(\varphi(\overline{p2},\text{MU})\right).\left(\bar{\gamma}\cdot\bar{\varepsilon}(\text{kp})\right).\left(\bar{\gamma}\cdot\left(\overline{p2}-\overline{\text{kp}}\right)+\text{MU}\right).\left(\bar{\gamma}\cdot\bar{\varepsilon}^*(\text{kg})\right).\left(\varphi(\overline{p1},\text{MU})\right)}{3\left((\text{kp}-p2)^2-\text{MU}^2\right)}$$

3.6. Finer-grained expansions

To expand scalar products of Lorentz vectors FeynCalc provides the function `ExpandScalarProduct`. The standard behavior of this command is to expand every scalar product in the expression.

```
In[1]:= exp = SPD[q1, p1 + p2] SPD[q2, p3 + p4] SPD[p5 + p6, p7 + p8]

Out[1]= ((p1 + p2) · q1)((p3 + p4) · q2)((p5 + p6) · (p7 + p8))

In[2]:= ExpandScalarProduct[exp]

Out[2]= (p1 · q1 + p2 · q1)(p3 · q2 + p4 · q2)(p5 · p7 + p5 · p8
          +p6 · p7 + p6 · p8)
```

which might lead to an unnecessary increase of terms, if the user wants to expand only some particular scalar products. FeynCalc 9.0 improves `ExpandScalarProduct` by introducing the option `Momentum` which allows to specify a list of momenta that need to be contained in a scalar product that will be expanded. All the other scalar products will remain untouched.

```
In[1]:= exp = SPD[q1, p1 + p2] SPD[q2, p3 + p4] SPD[p5 + p6, p7 + p8]

Out[1]= ((p1 + p2) · q1)((p3 + p4) · q2)((p5 + p6) · (p7 + p8))

In[2]:= ExpandScalarProduct[exp, Momentum -> {q1}]

Out[2]= (p1 · q1 + p2 · q1)((p3 + p4) · q2)((p5 + p6) · (p7 + p8))

In[3]:= ExpandScalarProduct[exp, Momentum -> {q2}]

Out[2]= (p3 · q2 + p4 · q2)((p1 + p2) · q1)((p5 + p6) · (p7 + p8))
```

The same option is now present also in `DiracGammaExpand` that is used to expand Lorentz vectors contracted with Dirac matrices

```
In[1]:= exp = GSD[q1 + p1 + p2].GSD[q2 + p3 + p4].GSD[p5 + p6 + p7 + p8]

Out[1]= (γ · (p1 + p2 + q1)).(γ · (p3 + p4 + q2)).(γ · (p5 + p6 + p7 + p8))

In[2]:= DiracGammaExpand[exp]

Out[2]= (γ · p1 + γ · p2 + γ · q1).(γ · p3 + γ · p4 + γ · q2).(γ · p5
          +γ · p6 + γ · p7 + γ · p8)

In[3]:= DiracGammaExpand[exp, Momentum -> {q1}]

Out[3]= (γ · p1 + γ · p2 + γ · q1).(γ · (p3 + p4 + q2)).(γ · (p5
           +p6 + p7 + p8))

In[4]:= DiracGammaExpand[exp, Momentum -> {q2}]

Out[4]= (γ · (p1 + p2 + q1)).(γ · p3 + γ · p4 + γ · q2).(γ · (p5
           +p6 + p7 + p8))
```

3.7. $SU(N)$ generators with explicit fundamental indices

FeynCalc denotes $SU(N)$ generators in the fundamental representation as `SUNT[a]` where `a` stands for the adjoint index. The fundamental indices are suppressed, so that a chain of `SUNT`-matrices is understood to have only two free fundamental indices, e.g. `SUNT[a,b,c]` stands for $T^a_{ij}T^b_{jk}T^c_{kl}$ and it is not possible to express, say $T^a_{ij}T^b_{kl}$ with `SUNT` objects only.

Due to this limitation, evaluation of Feynman amplitudes with more than two free fundamental color indices (e.g $q\bar{q} \to q\bar{q}$ scattering in QCD) was very inconvenient and usually required additional Mathematica code to obtain the correct result. For this reason FeynCalc 9.0 introduces a new object `SUNTF[{a},i,j]` that stands for T^a_{ij}, an $SU(N)$ generator in the fundamental representation with explicit fundamental indices `i` and `j` and the adjoint index `a`. Hence expressions like $T^a_{ij}T^b_{kl}$ or $T^a_{ij}T^b_{jk}T^c_{lm}$ can be now conveniently expressed with `SUNTF[{a},i,j]*SUNTF[{b},k,l]` and `SUNTF[{a,b},i,k]*SUNTF[{c},l,m]` respectively. The new `SUNTF` objects are fully compatible with `SUNSimplify`, the standard routine for simplifying $SU(N)$ algebra.

```
In[1]:= exp1 = SUNTF[{a}, i, j] SUNTF[{b}, j, k] SUNTF[{c}, k, l]

Out[1]= T^a_ij T^b_jk T^c_kl

In[2]:= SUNSimplify[exp1]

Out[2]= (T^a T^b T^c)_il

In[3]:= exp2 = exp1 SUNFDelta[i, l]

Out[3]= δ_il T^a_ij T^b_jk T^c_kl

In[4]:= SUNSimplify[exp2]

Out[4]= tr(T^c.T^a.T^b)
```

4. Using FeynCalc with non-relativistic EFTs

Up to now we silently assumed that all the amplitudes and expressions that we want to evaluate stem from a theory that is manifestly Lorentz covariant. This nice property of relativistic QFTs is often taken for granted, but one surely should not forget about EFTs that are used to describe non-relativistic systems, where the corresponding Lagrangians often do not exhibit manifest Lorentz covariance.

To our knowledge, there are no public tools for doing algebraic calculations in non-relativistic EFTs, where one has to explicitly distinguish between temporal and spatial components of Lorentz tensors. Naively, one might think that to do a calculation in such a theory using computer, one would need to write a large amount of additional code almost from scratch. However, with such a versatile tool like FeynCalc, this estimate turns out to be too pessimistic. In the following we want to give a simple example of using FeynCalc in a non-relativistic calculation, where only a comparably small amount of additional Mathematica code is needed.

In Section 2 we have already mentioned NRQCD [16], which is an EFT of QCD that was developed to exploit the separation of scales

$$mv^2 \ll mv \ll m \tag{10}$$

in a heavy quarkonium. Here, m denotes the heavy quark mass and v stands for the relative velocity of heavy quarks in the quarkonium. The scales m, mv and mv^2 are usually called hard, soft and ultrasoft respectively.

NRQCD is obtained from QCD by integrating out all degrees of freedom above the soft scale. The hard contributions are of course not simply thrown away. Their effects are incorporated in the matching coefficients ω_n that multiply operators $\mathcal{O}_n$ of the NRQCD Lagrangian, which can be schematically written as

$$\mathcal{L}_{\text{NRQCD}} = \sum_n \frac{\omega_n}{m^n}\mathcal{O}_n. \tag{11}$$

Since for charm and bottom quarks we have

$$m \gg \Lambda_{\text{QCD}}, \tag{12}$$

with Λ_{QCD} being the QCD scale at which the perturbation theory breaks down, the matching can be always done perturbatively.

The matching coefficients are fixed by comparing suitable quantities in perturbative QCD and in perturbative NRQCD at finite order in the expansion in v. The NRQCD Lagrangian itself contains an infinite number of operators of arbitrary high dimensions that are compatible with the symmetries of QCD. Using the power counting rules of the theory, one can estimate the relative importance of the operators for each process of interest. For this reason, usually only a small number of NRQCD operators needs to be considered in a practical calculation.

In the following we want to use FeynCalc to perform the matching between QCD and NRQCD in order to extract the matching coefficients (at leading order in α_s) that enter the decay rate of $\chi_{c0,2} \to \gamma\gamma$ at leading order in v. Notice that the decay $\chi_{c_1} \to \gamma\gamma$ does not occur, because it is forbidden by the Landau–Yang theorem.

These matching coefficients have been calculated in the framework of NRQCD multiple times [16,32,61–63], with many of these calculations carried out in a fully covariant way using the covariant projector technique [33]. Nevertheless, for pedagogical reasons we want to stick to the explicit non-covariant matching in the spirit of [16] and [62]. We also would like to remark that the projector technique has not yet been generalized for higher quarkonium Fock states, that include not only two heavy quarks $|Q\bar{Q}\rangle$ but also gluons (e.g. $|Q\bar{Q}g\rangle$ or $|Q\bar{Q}gg\rangle$). For this reason, the presented approach might still be useful in calculations, where such higher order contributions have to be considered. We also want to stress that codes which offer out of the box support for doing NRQCD calculations already exist (e.g. FDC [19] package), so the current example merely shows a quick naive implementation not optimized for performance or flexibility.

The factorization formulas for the decay rates [16] are given by

$$\begin{aligned}\Gamma(\chi_{c_0} \to \gamma\gamma) = {} & \frac{2\text{Im}f_{em}(^3P_0)}{3m^4}\langle\chi_{c_0}|\chi^\dagger(-\tfrac{i}{2}\overleftrightarrow{\mathbf{D}}\cdot\boldsymbol{\sigma})\psi|0\rangle \\ & \times \langle 0|\psi^\dagger(-\tfrac{i}{2}\overleftrightarrow{\mathbf{D}}\cdot\boldsymbol{\sigma})\chi|\chi_{c_0}\rangle \end{aligned} \tag{13}$$

$$\begin{aligned}\Gamma(\chi_{c_2} \to \gamma\gamma) = {} & \frac{2\text{Im}f_{em}(^3P_2)}{m^4}\langle\chi_{c_2}|\chi^\dagger(-\tfrac{i}{2}\overleftrightarrow{\mathbf{D}}^{(i}\boldsymbol{\sigma}^{j)})\psi|0\rangle \\ & \times \langle 0|\psi^\dagger(-\tfrac{i}{2}\overleftrightarrow{\mathbf{D}}^{(i}\boldsymbol{\sigma}^{j)})\chi|\chi_{c_2}\rangle. \end{aligned} \tag{14}$$

Here, Pauli spinor field ψ (χ) annihilates (creates) a heavy quark (antiquark), $\boldsymbol{\sigma}$ is the Pauli vector and the covariant derivative is defined as

$$D^\mu = \partial^\mu + igA^\mu \equiv (D^0, -\mathbf{D}), \tag{15}$$

so that

$$iD^0 = i\partial^0 - gA^0, \tag{16}$$

$$i\mathbf{D} = i\boldsymbol{\nabla} + g\mathbf{A}, \tag{17}$$

where A^μ is the gluon field and g stands for the QCD coupling constant. Furthermore,

$$\psi^\dagger \overleftrightarrow{\mathbf{D}}\, \chi \equiv \psi^\dagger(\mathbf{D}\chi) - (\mathbf{D}\psi)^\dagger\chi, \tag{18}$$

$$\overleftrightarrow{\mathbf{D}}^{(i}\boldsymbol{\sigma}^{j)} \equiv \frac{1}{2}\left(\overleftrightarrow{\mathbf{D}}^{i}\boldsymbol{\sigma}^{j} + \overleftrightarrow{\mathbf{D}}^{j}\boldsymbol{\sigma}^{i}\right) - \frac{1}{3}\delta^{ij}\overleftrightarrow{\mathbf{D}}\cdot\boldsymbol{\sigma}. \tag{19}$$

The NRQCD long distance matrix elements (LDME)

$$\langle\chi_{c_0}|\chi^\dagger(-\tfrac{i}{2}\overleftrightarrow{\mathbf{D}}\cdot\boldsymbol{\sigma})\psi|0\rangle\langle 0|\psi^\dagger(-\tfrac{i}{2}\overleftrightarrow{\mathbf{D}}\cdot\boldsymbol{\sigma})\chi|\chi_{c_0}\rangle \tag{20}$$

and

$$\langle\chi_{c_2}|\chi^\dagger(-\tfrac{i}{2}\overleftrightarrow{\mathbf{D}}^{(i}\boldsymbol{\sigma}^{j)})\psi|0\rangle\langle 0|\psi^\dagger(-\tfrac{i}{2}\overleftrightarrow{\mathbf{D}}^{(i}\boldsymbol{\sigma}^{j)})\chi|\chi_{c_2}\rangle \tag{21}$$

are non-perturbative. They can be determined from fitting to the experimental data or computed on the lattice. On the other hand, the matching coefficients $f_{em}(^3P_0)$ and $f_{em}(^3P_2)$ can be calculated in perturbation theory from the matching condition [16]

$$\begin{aligned} & 2\,\text{Im}\,A\,(Q\bar{Q} \to Q\bar{Q})\Big|_{\text{pert. QCD}} \\ & = \frac{2\text{Im}f_{em}(^3P_0)}{3m^4}\langle Q\bar{Q}|\chi^\dagger(-\tfrac{i}{2}\overleftrightarrow{\mathbf{D}}\cdot\boldsymbol{\sigma})\psi|0\rangle \\ & \quad \times \langle 0|\psi^\dagger(-\tfrac{i}{2}\overleftrightarrow{\mathbf{D}}\cdot\boldsymbol{\sigma})\chi|Q\bar{Q}\rangle|_{\text{pert. NRQCD}} \\ & \quad + \frac{2\text{Im}f_{em}(^3P_2)}{m^4}\langle Q\bar{Q}|\chi^\dagger(-\tfrac{i}{2}\overleftrightarrow{\mathbf{D}}^{(i}\boldsymbol{\sigma}^{j)})\psi|0\rangle \\ & \quad \times \langle 0|\psi^\dagger(-\tfrac{i}{2}\overleftrightarrow{\mathbf{D}}^{(i}\boldsymbol{\sigma}^{j)})\chi|Q\bar{Q}\rangle|_{\text{pert. NRQCD}}, \end{aligned} \tag{22}$$

where on the right hand side we have displayed only spin triplet terms that contribute at leading order in v. The left hand side of Eq. (22) denotes twice the imaginary part of the perturbative QCD amplitude $Q\bar{Q} \to Q\bar{Q}$ with 2 photons in the intermediate state. It is understood that this amplitude also has to be expanded to second order in v.

We start the matching calculation by considering the on-shell amplitude for the perturbative process $Q(p_1)\bar{Q}(p_2) \to \gamma(k_1)\gamma(k_2)$ in QCD. The kinematics of this process reads

$$p_1 + p_2 = k_1 + k_2, \tag{23}$$

$$p_1^2 = p_2^2 = m^2, \tag{24}$$

$$k_1^2 = k_2^2 = 0, \tag{25}$$

with $p_i = (\sqrt{m^2 + \mathbf{p}_i^2}, \mathbf{p}_i) \equiv (E_i, \mathbf{p}_i)$ and $k_i = (|\mathbf{k}_i|, \mathbf{k}_i)$. Obviously, it is most convenient to work in the quarkonium rest frame, where

$$\mathbf{p}_1 = -\mathbf{p}_2 \equiv \mathbf{q}, \tag{26}$$

$$E_1 = E_2 \equiv E_q = \sqrt{m + \mathbf{q}^2}, \tag{27}$$

$$\mathbf{k}_1 = -\mathbf{k}_2. \tag{28}$$

For convenience, the photon polarization vectors can be chosen to be purely spatial, satisfying

$$\epsilon(k_1)^0 = \epsilon(k_2)^0 = 0, \tag{29}$$

$$\boldsymbol{\epsilon}(k_{1/2}) \cdot \mathbf{k}_{1/2} = \boldsymbol{\epsilon}(k_{1/2}) \cdot \mathbf{k}_{2/1} = 0. \tag{30}$$

We need to expand the QCD amplitude in v, i.e. in $|\mathbf{q}|/m$ up to second order which involves rewriting Dirac spinors for Q and $\bar{Q}$ in terms of the Pauli spinors. For the latter let us recall that in 4-dimensions we can decompose any chain of Dirac matrices into scalar, pseudoscalar, vector, axial vector and tensor (SPVAT) components. This decomposition stems from the fact that the 4 dimensional matrices I, γ^5, γ^μ, $\gamma^5\gamma^\mu$ and $\sigma^{\mu\nu} = \frac{i}{2}[\gamma^\mu, \gamma^\nu]$ form a basis, such that any 4×4 matrix M can be written as

$$M = c_1 I + c_2\gamma^5 + c_{3\mu}\gamma^\mu + c_{4\mu}\gamma^5\gamma^\mu + c_{5\mu\nu}\sigma^{\mu\nu}. \tag{31}$$

Therefore, there are only 5 unique spinor structures involving heavy quarks that we can encounter in any tree level amplitude. In fact, the only components that appear in this calculation are vector and axial vector, so that we do not need to consider the other three. Using the explicit form of the Dirac spinors with the non-relativistic normalization,

$$u(\mathbf{q}) = \sqrt{\frac{E_q + m}{2E_q}} \begin{pmatrix} \xi \\ \frac{\mathbf{q}\cdot\boldsymbol{\sigma}}{E_q + m}\xi \end{pmatrix}, \tag{32}$$

$$v(-\mathbf{q}) = \sqrt{\frac{E_q + m}{2E_q}} \begin{pmatrix} -\frac{\mathbf{q}\cdot\boldsymbol{\sigma}}{E_q + m}\eta \\ \eta \end{pmatrix}, \tag{33}$$

with ξ and η being 2-component spinors, we obtain

$$\bar{v}(-\mathbf{q})\gamma^0 u(\mathbf{q}) = 0, \tag{34}$$

$$\bar{v}(-\mathbf{q})\gamma^i v(\mathbf{q}) = \eta^\dagger \boldsymbol{\sigma}^i \xi - \frac{\mathbf{q}^i}{2m^2}\eta^\dagger \mathbf{q}\cdot\boldsymbol{\sigma}\xi + \mathcal{O}((|\mathbf{q}|/m)^3), \tag{35}$$

$$\bar{v}(-\mathbf{q})\gamma^0\gamma^5 u(\mathbf{q}) = \eta^\dagger \xi \left(1 - \frac{\mathbf{q}^2}{2m^2}\right) + \mathcal{O}((|\mathbf{q}|/m)^3), \tag{36}$$

$$\bar{v}(-\mathbf{q})\gamma^i\gamma^5 u(\mathbf{q}) = \frac{i}{m}\eta^\dagger (\mathbf{q}\times\boldsymbol{\sigma})^i \xi + \mathcal{O}((|\mathbf{q}|/m)^3). \tag{37}$$

Let us first ignore all the complications related to the non-relativistic expansion and see how far we can get with the QCD amplitude without breaking the covariant notation.

At this order in v and α_s, there are only two tree level diagrams to consider that can be trivially generated with FeynArts.

```
In[1]:= $LoadFeynArts = True;
        $FeynCalcStartupMessages = False;
        << FeynCalc`;
        $FAVerbose = 0;

In[2]:= diags = InsertFields[CreateTopologies[0, 2 -> 2],
      {F[3, {2, a}], -F[3, {2, b}]} -> {V[1], V[1]},
      InsertionLevel -> {Classes}, Model -> "SMQCD"];
```

Then the amplitudes are converted into FeynCalc notation and simplified using standard FeynCalc functions.

```
In[3]:= amps = (9/4 EQ^2*FCFAConvert[
    CreateFeynAmp[diags, Truncated -> False, PreFactor -> -1],
    IncomingMomenta -> {p1, p2}, OutgoingMomenta -> {k1, k2},
    UndoChiralSplittings -> True,
    TransversePolarizationVectors -> {k1, k2},
    ChangeDimension -> 4, List -> False]) // Contract // Factor
```

$$\text{Out[3]:= } i\text{EL}^2\text{EQ}^2\delta_{ab}\frac{(\varphi(-\overline{\text{p2}},\text{MC}))\cdot(\tilde{\gamma}\cdot\tilde{\varepsilon}^*(\text{k1}))\cdot(\tilde{\gamma}\cdot(\overline{\text{k1}-\text{p2}})+\text{MC})\cdot(\tilde{\gamma}\cdot\tilde{\varepsilon}^*(\text{k2}))\cdot(\varphi(\overline{\text{p1}},\text{MC}))}{(\overline{\text{p2}-\text{k1}})^2-\text{MC}^2}$$
$$+i\text{EL}^2\text{EQ}^2\delta_{ab}\frac{(\varphi(-\overline{\text{p2}},\text{MC}))\cdot(\tilde{\gamma}\cdot\tilde{\varepsilon}^*(\text{k2}))\cdot(\tilde{\gamma}\cdot(\overline{\text{k2}-\text{p2}})+\text{MC})\cdot(\tilde{\gamma}\cdot\tilde{\varepsilon}^*(\text{k1}))\cdot(\varphi(\overline{\text{p1}},\text{MC}))}{(\overline{\text{p2}-\text{k2}})^2-\text{MC}^2}$$

The next step is to put the external particles on-shell

```
In[4]:= FCClearScalarProducts[];
        ScalarProduct[k1, k1] = 0;
        ScalarProduct[k2, k2] = 0;
        ScalarProduct[p1, p1] = MC^2;
        ScalarProduct[p2, p2] = MC^2;
```

and perform the SPVAT decomposition of the spinor chains,

```
In[5]:= repRuleHideChains = {
  FCI[Spinor[-p2, MC].GA[x_].GA[5].Spinor[p1, MC]] :> FCI[FV[A, x]],
  FCI[Spinor[-p2, MC].GA[x_].Spinor[p1, MC]] :> FCI[FV[V, x]],
  FCI[Spinor[-p2, MC].GS[x_].GA[5].Spinor[p1, MC]] :> FCI[SP[A, x]],
  FCI[Spinor[-p2, MC].GS[x_].Spinor[p1, MC]] :> FCI[SP[V, x]]
  };

In[6]:= amps2 = amps // DiracSimplify // DiracReduce // FCI //
        ReplaceAll[#, repRuleHideChains] & //
        PropagatorDenominatorExplicit[#, Dimension -> 4] & //
        Contract // ReplaceAll[#, Pair[Momentum[k1 | k2],
        Momentum[Polarization[k1 | k2, ___]]] -> 0] &
```

$$\text{Out[6]:= } \frac{\text{EL}^2\text{EQ}^2\delta_{ab}\epsilon^{\overline{\text{Ak1}}\tilde{\varepsilon}^*(\text{k1})\tilde{\varepsilon}^*(\text{k2})}}{2(\overline{\text{k1}\cdot\text{p2}})} - \frac{\text{EL}^2\text{EQ}^2\delta_{ab}\epsilon^{\overline{\text{Ak2}}\tilde{\varepsilon}^*(\text{k1})\tilde{\varepsilon}^*(\text{k2})}}{2(\overline{\text{k2}\cdot\text{p2}})} + \frac{i\text{EL}^2\text{EQ}^2\delta_{ab}(\overline{\text{V}}\cdot\tilde{\varepsilon}^*(\text{k1}))(\overline{\text{p2}}\cdot\tilde{\varepsilon}^*(\text{k2}))}{\overline{\text{k2}}\cdot\overline{\text{p2}}}$$
$$+\frac{i\text{EL}^2\text{EQ}^2\delta_{ab}(\overline{\text{p2}}\cdot\tilde{\varepsilon}^*(\text{k1}))(\overline{\text{V}}\cdot\tilde{\varepsilon}^*(\text{k2}))}{\overline{\text{k1}}\cdot\overline{\text{p2}}} + \frac{i\text{EL}^2\text{EQ}^2\delta_{ab}(\overline{\text{k1}}\cdot\overline{\text{V}})(\tilde{\varepsilon}^*(\text{k1})\cdot\tilde{\varepsilon}^*(\text{k2}))}{2(\overline{\text{k1}}\cdot\overline{\text{p2}})}$$
$$+\frac{i\text{EL}^2\text{EQ}^2\delta_{ab}(\overline{\text{k2}}\cdot\overline{\text{V}})(\tilde{\varepsilon}^*(\text{k1})\cdot\tilde{\varepsilon}^*(\text{k2}))}{2(\overline{\text{k2}}\cdot\overline{\text{p2}})}$$

where for convenience we chose to abbreviate vector and axial vector chains as

$$V^\mu \equiv \bar{v}(\mathbf{p}_2)\gamma^\mu u(\mathbf{p}_1), \tag{38}$$

$$A^\mu \equiv \bar{v}(\mathbf{p}_2)\gamma^\mu\gamma^5 u(\mathbf{p}_1). \tag{39}$$

If we are to expand the resulting expression in $|\mathbf{q}|/m$, we must make the $\mathbf{q}$-dependence explicit in all parts of the amplitude. Since different components of the 4-vectors and spinor chains that appear in the computation depend on $|\mathbf{q}|$ in a different way, it now becomes necessary to break the covariant notation. However, by doing so in a naive way, e.g. by writing something like

$$V\cdot k_1 = V^0|\mathbf{k}| - \mathbf{V}\cdot\mathbf{k}, \tag{40}$$

$$\epsilon^{\mu\nu\rho\sigma}k_{1\mu}A_\nu\varepsilon^*_\rho(k_1)\varepsilon^*_\sigma(k_2) = \epsilon^{\mu 0\rho\sigma}k_{1\mu}A^0\varepsilon^*_\rho(k_1)\varepsilon^*_\sigma(k_2) - \epsilon^{\mu i\rho\sigma}k_{1\mu}A^i\varepsilon^*_\rho(k_1)\varepsilon^*_\sigma(k_2) \tag{41}$$

we introduce new objects that carry Cartesian indices and thus cannot be handled by the built-in routines for working with Lorentz tensors (e.g. `Contract`, `ScalarProduct`, `ExpandScalarProduct`, etc.). Fortunately, it is possible to completely avoid introducing any Cartesian tensors or tensors with mixed Lorentz and Cartesian indices by exploiting FeynCalc's built-in `TensorFunction` in a clever way.

This approach is based on [64], although we do not consider a boosted $Q\bar{Q}$-system and assume that the quarkonium is at rest. To see how this works, let us first define a symmetric tensor $E^{\mu\nu}$ with

$$E^{\mu\nu} = \begin{cases} 0 & \text{for } \mu = 0 \text{ or } \nu = 0, \\ \delta^{ij} & \text{for } \mu \neq 0 \text{ and } \nu \neq 0. \end{cases} \tag{42}$$

With $E^{\mu\nu}$ we can write any Cartesian scalar product $x^i y^i$ as

$$x^i y^i = x^i y^j \delta^{ij} = E^{\mu\nu}x_\mu y_\nu \equiv E(x, y), \tag{43}$$

where x^0 and y^0 can be anything, since they drop out by construction. If x is a pure Cartesian vector, then we can choose

$x^\mu = (0, x^i)$. Suppose that we want to expand $x^i y^i$ in $|\mathbf{x}|$ or $|\mathbf{y}|$. Then we can write

$$E(x, y) = |\mathbf{x}||\mathbf{y}|E(\hat{x}, \hat{y}), \tag{44}$$

with $x^i = |\mathbf{x}|\hat{x}^i$ and $y^i = |\mathbf{y}|\hat{y}^i$, where $E(\hat{x}, \hat{y})$ does not depend on $|\mathbf{x}|$ and $|\mathbf{y}|$. Therefore, a Minkowski scalar product $x \cdot y$ can be rewritten as

$$x^\mu y_\mu = x^0 y^0 + |\mathbf{x}||\mathbf{y}|E(\hat{x}, \hat{y}) \tag{45}$$

and if x and y are external 4-momenta, then we have

$$x^\mu y_\mu = \sqrt{m_x^2 + |\mathbf{x}|^2}\sqrt{m_y^2 + |\mathbf{y}|^2} + |\mathbf{x}||\mathbf{y}|E(\hat{x}, \hat{y}), \tag{46}$$

so that the expansion in the scalar variables $|\mathbf{x}|$ or $|\mathbf{y}|$ can be carried out without making any reference to 3-vectors. Some useful relations for dealing with E-tensors are

$$E^{\mu\nu} g_{\mu\nu} = -3, \tag{47}$$

$$E^{\mu\nu} E^{\rho\sigma} g_{\mu\rho} = -E^{\nu\sigma}. \tag{48}$$

In a similar manner we can also rewrite terms that involve 3-dimensional epsilon tensors by introducing

$$C_{\mu\nu\rho} = \begin{cases} 0 & \text{for } \mu = 0 \text{ or } \nu = 0 \text{ or } \rho = 0, \\ \varepsilon_{ijk} & \text{for } \mu \neq 0 \text{ and } \nu \neq 0 \text{ and } \rho \neq 0, \end{cases} \tag{49}$$

such that

$$\varepsilon^{ijk} x^i y^j z^k = -\varepsilon_{ijk} x^i y^j z^k = -C_{\mu\nu\rho} x^\mu y^\nu z^\rho \equiv -C(x, y, z). \tag{50}$$

Then, it is easy to see that

$$\begin{aligned} \varepsilon^{\sigma\mu\nu\rho} a_\sigma x_\mu y_\nu z_\rho &= \varepsilon^{ijk}(a^0 x_i y_j z_k - x^0 a_i y_j z_k + y^0 a_i x_j z_k - z^0 a_i x_j y_k) \\ &= a^0 C(x, y, z) - x^0 C(a, y, z) + y^0 C(a, x, z) - z^0 C(a, x, y), \end{aligned} \tag{51}$$

from where we again can easily expand in the 3-momenta of a, x, y or z, since

$$C(x, y, z) = |\mathbf{x}||\mathbf{y}||\mathbf{z}|C(\hat{x}, \hat{y}, \hat{z}), \tag{52}$$

with $C(\hat{x}, \hat{y}, \hat{z})$ being independent of $|\mathbf{x}|$, $|\mathbf{y}|$ and $|\mathbf{z}|$. The product of two C tensors can be expressed through

$$C^{\mu\nu\rho} C^{\alpha\beta\gamma} = \begin{vmatrix} E^{\mu\alpha} & E^{\mu\beta} & E^{\mu\gamma} \\ E^{\nu\alpha} & E^{\nu\beta} & E^{\nu\gamma} \\ E^{\rho\alpha} & E^{\rho\beta} & E^{\rho\gamma} \end{vmatrix}. \tag{53}$$

The basic properties of $E^{\mu\nu}$ (denoted as `NRPair`) and $C^{\mu\nu\rho}$ (denoted as `NREps`) can be implemented in FeynCalc with a minimal amount of extra code.

```
In[7]:= SetAttributes[NRPairContract, Orderless];
        TensorFunction[NREps, x, y, z];
        TensorFunction[{NRPair, "S"}, x, y];
        NREps[a___, x_, b___, x_, c___] := 0;
        NRPairContract /:
         NRPairContract[LorentzIndex[x_], LorentzIndex[x_]] := -3;
        NRPairContract /:
         NRPairContract[LorentzIndex[x_], y_] *
         NRPairContract[LorentzIndex[x_], z_] := - NRPairContract[y, z];
        NRPairContract /:
         NRPairContract[LorentzIndex[x_], y_] *
         NREpsContract[a___, LorentzIndex[x_], b___] := - NREpsContract[a, y, b
             ];
        NRPairContract /:
         NRPairContract[LorentzIndex[x_], y_]^2 := - NRPairContract[y, y];
        NREpsContract /:
         NREpsContract[x_, y_, z_]^2 := - 6;
         NREpsContract /:
        NREpsContract[mu_, nu_, rho_] NREpsContract[al_, be_, ga_] :=
          (Det[{{np[ mu, al],  np[ mu, be], np[ mu, ga]},
           {np[ nu, al ],  np[ nu, be],  np[nu, ga]},
           {np[ rho, al ],  np[ rho, be],  np[ rho, ga ]}}]  /.
           np -> NRPairContract);
```

Contractions of $E^{\mu\nu}$ and $C^{\mu\nu\rho}$ with each other or with the metric tensor are simplified by `NRContract`, while `NRExpand` implements Eq. (51).

```
In[8]:= NRContract[expr_] :=
         FixedPoint[(Expand2[Contract[#], {NRPair, NREps}] //. {NRPair ->
         NRPairContract, NREps -> NREpsContract}) &, expr] /.
         {NRPairContract -> NRPair, NREpsContract -> NREps};

In[9]:= NRExpand[expr_] :=
       FixedPoint[ReplaceRepeated[Expand2[#, {Eps, NREps}],
       {Eps[a_Momentum, x_Momentum, y_Momentum, z_Momentum] :>
         NREn[a] NREps[x, y, z] - NREn[x ] NREps[a, y, z] +
         NREn[y] NREps[a, x, z] - NREn[z] NREps[a, x, y]}] &, expr];
```

Here we use `NREn` to denote the temporal components of 4-momenta. The expansions of spinor chains given in Eqs. (34)–(37) are now straightforward to translate into FeynCalc notation.

```
In[10]:= repRuleExpandedChains = {
          Pair[v : Momentum[V], x_] :> -NRPair[v, x],
          Pair[a : Momentum[A], x_] :> NREn[a] NREn[x] - NRPair[a, x],
          NREn[Momentum[A]] -> 1 - (qvec^2) /(2 MC^2),
          NREn[Momentum[V]] -> 0,
          NRPair[x_, Momentum[V]] :> -((qvec^2 NRPair[Momentum[qhat],
              Momentum[{S, I}]] NRPair[Momentum[qhat], x])/(2 MC^2)) +
          NRPair[Momentum[{S, I}], x],
          NRPair[x_, Momentum[A]] :>
         -((I qvec NREps[Momentum[qhat], Momentum[{S, I}], x])/MC),
          NREps[x___, a : Momentum[A], y___] :> (li =
           LorentzIndex[Unique[]]; -NREps[x, li, y] NRPair[a, li]),
          NREps[x___, v : Momentum[V], y___] :>
         ( li  = LorentzIndex[Unique[]]; -NREps[x, li, y] NRPair[v, li])
        };
```

Simplifications that are specific to the kinematics of the process are also easy to define.

```
In[11]:= NREn[Momentum[Polarization[k1 | k2, ___]]] = 0;
          NREn[Momentum[k1 | k2]] = kvec;
          NREn[Momentum[p1 | p2]] = Sqrt[MC^2 + qvec^2];
          NREn[Momentum[qhat | qhatp | {S, _}]] = 0;
          NREn[Momentum[k1hat]] = 1;
          NRPair[Momentum[p1], x_] = qvec NRPair[Momentum[qhat], x];
          NRPair[Momentum[p2], x_] = -qvec NRPair[Momentum[qhat], x];
          NRPair[Momentum[p1p], x_] = qvec NRPair[Momentum[qhatp], x];
          NRPair[Momentum[p2p], x_] = -qvec NRPair[Momentum[qhatp], x];
          NRPair[Momentum[k1 | k2 | k1hat | k2hat],
            Momentum[Polarization[k2 | k1, ___]]] = 0;
          NRPair[Momentum[x_], Momentum[x_]] :=
          1 /;  MemberQ[{qhat, qhatp, p1hat, p2hat, k1hat, k2hat}, x];
          NRPair[Momentum[k1], x_] = kvec NRPair[Momentum[k1hat], x];
          NRPair[Momentum[k2], x_] = -kvec NRPair[Momentum[k1hat], x];
          kvec = Sqrt[MC^2 + qvec^2];

          repRuleExpansion = {
            FCI@SP[x_, a : Polarization[z_, ___]] /;  MemberQ[{k1, k2}, z] :>
             -NRPair[Momentum[x], Momentum[a]],
            FCI@SP[x_, (y : p1 | p2 | k1 | k2 | {S, I} | {S, -I} | k1hat | qhat)] :>
             NREn[Momentum[x]] NREn[Momentum[y]] -
              NRPair[Momentum[x], Momentum[y]],
            NREps[a___, Momentum[k1], z___] :> kvec NREps[a, Momentum[k1hat
                ], z],
            NREps[a___, Momentum[k2], z___] :> - kvec NREps[a, Momentum[
                k1hat], z],
            NREps[a___, Momentum[p1], b___] :> qvec NREps[a, Momentum[qhat
                ], b],
            NREps[a___, Momentum[p2], b___] :> -qvec NREps[a, Momentum[
                qhat], b]
          };
```

Finally, we can expand the amplitude up to second order in $|\mathbf{q}|/m$.

```
In[12]:= amps3 = amps2 // NRExpand //
          ReplaceRepeated[#, repRuleExpandedChains] & // NRContract //
          ReplaceRepeated[#, repRuleExpansion] & // Series[#, {qvec, 0, 2}] & //
          Normal // PowerExpand // NRContract;
```

To obtain $2\,\mathrm{Im}\,A\,(Q(p_1')\bar{Q}(p_2') \to Q(p_1)\bar{Q}(p_2))$ from our expanded amplitude, we need to multiply $A\,(Q(p_1')\bar{Q}(p_2') \to \gamma(k_1)\gamma(k_2))$

by $A^*\,(Q(p_1)\bar{Q}(p_2) \to \gamma(k_1)\gamma(k_2))$, sum over polarizations of the external photons and perform the phase space integration. For the latter we can use that

$$\int d\Omega_{k_1}\, \hat{k}_1^{i_1} \ldots \hat{k}_1^{i_{2n+1}} = 0, \tag{54}$$

$$\int d\Omega_{k_1}\, \hat{k}_1^{i_1} \ldots \hat{k}_1^{i_{2n}} = \frac{4\pi}{(n+2)!!}\left(\delta^{i_1 i_2} \ldots \delta^{i_{2n-1} i_{2n}} + \text{permutations}\right). \tag{55}$$

To implement these relations we need an auxiliary function that uncontracts the indices of $\hat{k}_1$

```
In[13]:= NRUncontract[expr_, l_List] :=
  expr /. {Power[t_NRPair, n_] :> times @@ Table[t, {i, 1, n}],
     Power[t_NREps, n_] :> times @@ Table[t, {i, 1, n}]} //. {
    NRPair[y_, x_] /; ! FreeQ2[y, l] && Head[x] =!= LorentzIndex :>
    ( li = Unique[$AL]; -NRPair[y, LorentzIndex[li]] NRPair[x, LorentzIndex[li]]),
    NREps[w___, y_, x___] /; ! FreeQ2[y, l] :> ( li = Unique[$AL];
    -NRPair[y, LorentzIndex[li]] NREps[w, LorentzIndex[li], x])
    } /. times -> Times;
```

and a replacement rule that handles the angular integration

```
In[14]:= angularIntegration[hat_] := {
   qHead[NRPair[i_, Momentum[hat]]] /; FreeQ2[{i}, {hat, S}] :> 0,
   qHead[a_Times] :> qHead[(List @@ a) /. NRPair[Momentum[hat], b_] :>
      {hat, b /. LorentzIndex[c_, _] :> c}],
   qHead[a_List] :> (Tdec[a, {}, List -> False, FCE -> False,
      Dimension -> 3] //. {(h : LorentzIndex | Momentum)[x_, 3] :>
       h[x], Pair -> NRPair})
   };
```

Then the left hand side of Eq. (22) is given by

```
In[15]:= res = (1/(16 Pi)) (Collect[(amps3 /. qhat -> qhatp)*
              ComplexConjugate[amps3] /. NRPair[x_, y_] :>
              -FCI@SP[x, y] + NREn[x] NREn[y], qvec] /. qvec^4 -> 0) //
               DoPolarizationSums[#, k1, k2] & //
               DoPolarizationSums[#, k2, k1] & //
               ReplaceRepeated[#, repRuleExpansion] & // Cancel //
               NRUncontract[#, {k1hat}] & //
               FCLoopIsolate[#, {k1hat}, Head -> qHead] & //
               ReplaceRepeated[#, angularIntegration[k1hat]] & // NRContract //
               ReplaceAll[#, {EL^4 -> 16 Pi^2 AlphaFS ^2}] & //
               SelectNotFree[#, S] &
```

$$\text{Out[15]:=}\ \frac{4\pi\alpha^2 \text{EQ}^4 \text{qvec}^2 \delta_{ab}^2 \text{NRPair}\left(\overline{\text{qhat}}, \overline{\{S,i\}}\right)\text{NRPair}\left(\overline{\text{qhatp}}, \overline{\{S,-i\}}\right)}{5\text{MC}^4} + \frac{22\pi\alpha^2 \text{EQ}^4 \text{qvec}^2 \delta_{ab}^2 \text{NRPair}\left(\overline{\text{qhat}}, \overline{\{S,-i\}}\right)\text{NRPair}\left(\overline{\text{qhatp}}, \overline{\{S,i\}}\right)}{15\text{MC}^4} + \frac{4\pi\alpha^2 \text{EQ}^4 \text{qvec}^2 \delta_{ab}^2 \text{NRPair}\left(\overline{\text{qhat}}, \overline{\text{qhatp}}\right)\text{NRPair}\left(\overline{\{S,-i\}}, \overline{\{S,i\}}\right)}{5\text{MC}^4}$$

An explicit expression for the right hand side of Eq. (22) can be obtained by using Fourier decompositions of the Pauli spinor fields (c.f. [65]), so that we end up with

$$\begin{aligned}
&\frac{4\alpha^2 Q^4 \pi}{5m^4}\mathbf{q}\cdot\mathbf{q}'\eta^\dagger\boldsymbol{\sigma}\xi\,\xi^\dagger\boldsymbol{\sigma}\eta + \frac{4\alpha^2 Q^4\pi}{5m^4}\eta^\dagger\mathbf{q}\cdot\boldsymbol{\sigma}\xi\,\xi^\dagger\mathbf{q}'\cdot\boldsymbol{\sigma}\eta \\
&\quad + \frac{22\alpha^2 Q^4\pi}{15m^4}\eta^\dagger\mathbf{q}'\cdot\boldsymbol{\sigma}\xi\,\xi^\dagger\mathbf{q}\cdot\boldsymbol{\sigma}\eta \\
&= \frac{\text{Im}f_{em}(^3P_2)}{m^4}\mathbf{q}\cdot\mathbf{q}'\,\eta^\dagger\boldsymbol{\sigma}\xi\,\xi^\dagger\boldsymbol{\sigma}\eta + \frac{\text{Im}f_{em}(^3P_2)}{m^4}\eta^\dagger\mathbf{q}\cdot\boldsymbol{\sigma}\xi\,\xi^\dagger\mathbf{q}'\cdot\boldsymbol{\sigma}\eta \\
&\quad + \frac{2}{3}\frac{\left(\text{Im}f_{em}(^3P_0) - \text{Im}f_{em}(^3P_2)\right)}{m^4}\eta^\dagger\mathbf{q}'\cdot\boldsymbol{\sigma}\xi\,\xi^\dagger\mathbf{q}\cdot\boldsymbol{\sigma}\eta,
\end{aligned} \tag{56}$$

from which we can immediately read off the values of the matching coefficients

$$\text{Im}f_{em}(^3P_0) = 3\alpha^2 Q^4\pi, \tag{57}$$

$$\text{Im}f_{em}(^3P_2) = \frac{4}{5}\alpha^2 Q^4\pi, \tag{58}$$

that agree with the known results from the literature [16,32,61–63].

5. Summary

We have presented new features and improvements in FeynCalc 9.0 and discussed cases in which FeynCalc can be used to obtain new results. Although the very first version of FeynCalc appeared almost 25 years ago, the development is still far from complete. New developments in theoretical particle physics show possible directions in which FeynCalc can evolve. This includes better support for multi-loop calculations and determination of matching coefficients in effective field theories, but also built-in interfaces to other useful software tools and the ability to work with non-relativistic theories.

Finally, we would like to observe that in the last two years some new general-purpose packages [24,25] for QFT calculations were released, which follow the approach similar to that of FeynCalc and thus provide a comparable level of flexibility. This development shows that even in the age of fully automatic packages for 1-loop calculations, user-friendly, semi-automatic tools like FeynCalc are still in demand and employed in many interesting research projects.

Acknowledgments

Two of the authors (RM and FO) would like to thank Daniel Wyler for his help and support in the earlier days of FeynCalc.

One of the authors (VS) wants to thank Hector Martinez Neira for bringing his attention to FeynCalc for the first time and his Ph.D. supervisor Nora Brambilla for encouraging him to work in this direction. Simone Biondini is acknowledged for testing 1-loop tensor decompositions with TID. VS would also like to express his gratitude to Antonio Vairo, Christoph Bobeth, Thomas Hahn, Georg Weiglein, Sergey Larin, Claude Duhr, Matthias Steinhauser, Alexander Smirnov, Massimo Passera and Yu Jia for useful discussions and explanations. His work has been supported by the DFG and the NSFC through funds provided to the Sino-German CRC 110 "Symmetries and the Emergence of Structure in QCD" (DFG/TR-110, NSFC Grant No. 11261130311), and by the DFG cluster of excellence "Origin and structure of the universe" (www.universe-cluster.de).

Last but not least, all the authors would like to thank the participants of the FeynCalc mailing list for their bug reports, feature requests, suggestions and encouragements.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.cpc.2016.06.008.

References

[1] C. Anastasiou, C. Duhr, F. Dulat, F. Herzog, B. Mistlberger, Phys. Rev. Lett. 114 (2015) 212001. arXiv:1503.06056.
[2] P. Marquard, A.V. Smirnov, V.A. Smirnov, M. Steinhauser, Phys. Rev. Lett. 114 (2015) 142002. arXiv:1502.01030.
[3] M. Beneke, Y. Kiyo, P. Marquard, A. Penin, J. Piclum, D. Seidel, M. Steinhauser, Phys. Rev. Lett. 112 (2014) 151801. arXiv:1401.3005.
[4] R. Mertig, M. Böhm, A. Denner, Comput. Phys. Comm. 64 (1991) 345–359.
[5] A. Denner, J. Küblbeck, R. Mertig, M. Böhm, Z. Phys. C 56 (1992) 261–272.
[6] W. Beenakker, A. Denner, S. Dittmaier, R. Mertig, T. Sack, Nuclear Phys. B 410 (1993) 245–279.
[7] W. Beenakker, A. Denner, W. Hollik, R. Mertig, T. Sack, D. Wackeroth, Nuclear Phys. B 411 (1994) 343–380.
[8] W. Beenakker, A. Denner, S. Dittmaier, R. Mertig, Phys. Lett. B 317 (1993) 622–630.
[9] R. Mertig, W.L. van Neerven, Z. Phys. C 70 (1996) 637–653. arXiv:hep-ph/9506451.
[10] R. Mertig, R. Scharf, Comput. Phys. Comm. 111 (1998) 265–273. arXiv:hep-ph/9801383.
[11] O.V. Tarasov, Phys. Rev. D 54 (1996) 6479–6490. arXiv:hep-th/9606018.
[12] O.V. Tarasov, Nuclear Phys. B 502 (1997) 455–482. arXiv:hep-ph/9703319.
[13] M. Buechler, G. Colangelo, J. Kambor, F. Orellana, Phys. Lett. B 521 (2001) 22–28. arXiv:hep-ph/0102287.

[14] T. Hahn, Comput. Phys. Comm. 140 (2001) 418–431. arXiv:hep-ph/0012260.
[15] F. Feng, R. Mertig, FormLink/FeynCalcFormLink : Embedding FORM in Mathematica and FeynCalc, 2012, arXiv:1212.3522.
[16] G.T. Bodwin, E. Braaten, G.P. Lepage, Phys. Rev. D 51 (1995) 1125–1171. arXiv:hep-ph/9407339.
[17] T. Hahn, M. Perez-Victoria, Comput. Phys. Comm. 118 (1999) 153–165. arXiv:hep-ph/9807565.
[18] G. Cullen, H. van Deurzen, N. Greiner, G. Heinrich, G. Luisoni, P. Mastrolia, E. Mirabella, G. Ossola, T. Peraro, J. Schlenk, J.F. von Soden-Fraunhofen, F. Tramontano, Eur. Phys. J. C 74 (8) (2014) 3001. arXiv:1404.7096.
[19] J.-X. Wang, Nucl. Instrum. Methods A 534 (2004) 241–245. arXiv:hep-ph/0407058.
[20] G. Belanger, F. Boudjema, J. Fujimoto, T. Ishikawa, T. Kaneko, K. Kato, Y. Shimizu, Phys. Rep. 430 (2006) 117–209. arXiv:hep-ph/0308080.
[21] M. Tentyukov, J. Fleischer, Comput. Phys. Comm. 132 (2000) 124–141. arXiv:hep-ph/9904258.
[22] J.A.M. Vermaseren, New features of FORM, 2007, arXiv:math-ph/0010025.
[23] N. Brambilla, A. Pineda, J. Soto, A. Vairo, Nuclear Phys. B 566 (2000) 275. arXiv:hep-ph/9907240.
[24] M. Wiebusch, Comput. Phys. Comm. 195 (2014) 172–190. arXiv:1412.6102.
[25] H.H. Patel, Comput. Phys. Comm. 197 (2015) 276–290. arXiv:1503.01469.
[26] G. 't Hooft, M. Veltman, Nuclear Phys. B 44 (1972) 189–213.
[27] P. Breitenlohner, D. Maison, Comm. Math. Phys. 52 (1977) 11–38.
[28] G. Passarino, M. Veltman, Nuclear Phys. B 160 (1979) 151.
[29] A. Buckley, J. Ferrando, S. Lloyd, K. Nordström, B. Page, M. Rüfenacht, M. Schönherr, G. Watt, Eur. Phys. J. C 75 (2015).
[30] P. Cho, A. Leibovich, Phys. Rev. D 53 (1996) 150–162. arXiv:hep-ph/9505329.
[31] Y.-J. Zhang, K.-T. Chao, Phys. Rev. Lett. 98 (2007) 092003. arXiv:hep-ph/0611086.
[32] A. Petrelli, M. Cacciari, M. Greco, F. Maltoni, M.L. Mangano, Nuclear Phys. B 514 (1998) 245–309. arXiv:hep-ph/9707223.
[33] G.T. Bodwin, A. Petrelli, Phys. Rev. D 66 (2002) 094011. arXiv:hep-ph/0205210.
[34] Y. Jia, X.-T. Yang, W.-L. Sang, J. Xu, J. High Energ. Phys. 2011 (2011) arXiv:arXiv:1104.1418.
[35] C.O. Dib, R. Rosenfeld, A. Zerwekh, J. High Energy Phys. 0605 (2006) 074. arXiv:hep-ph/0509179.
[36] D. Buttazzo, G. Degrassi, P.P. Giardino, G.F. Giudice, F. Sala, A. Salvio, A. Strumia, J. High Energ. Phys. 2013 (2013) arXiv:arXiv:1307.3536.
[37] D. de Florian, J. Mazzitelli, Phys. Rev. Lett. 111 (2013) 201801. arXiv:1309.6594.
[38] B. Xiao, Y.-K. Wang, Z.-Q. Zhou, S. hua Zhu, Phys. Rev. D 83 (2011) 057503. arXiv:1101.2507.
[39] M. Fael, L. Mercolli, M. Passera, J. High Energy Phys. 1507 (2015) 153. arXiv:1506.03416.
[40] L.S. Geng, J.M. Camalich, L. Alvarez-Ruso, M.J.V. Vacas, Phys. Rev. D 78 (2008) 014011. arXiv:0801.4495.
[41] D.R. Phillips, M.R. Schindler, R.P. Springer, Nuclear Phys. A 822 (2009) 1–19. arXiv:0812.2073.
[42] J. Kopp, V. Niro, T. Schwetz, J. Zupan, Phys. Rev. D 80 (2009) arXiv:arXiv:0907.3159.
[43] J.M. Cline, A.R. Frey, F. Chen, Phys. Rev. D 83 (2010) 083511. arXiv:1008.1784.
[44] S. Bray, J.S. Lee, A. Pilaftsis, Phys. Lett. B 628 (2005) 250–261. arXiv:hep-ph/0508077.
[45] R. Laha, B. Dasgupta, J.F. Beacom, Phys. Rev. D 89 (2013) 093025. arXiv:1304.3460.
[46] B. Dasgupta, J. Kopp, Phys. Rev. Lett. 112 (2013) 031803. arXiv:1310.6337.
[47] S. Foffa, R. Sturani, Phys. Rev. D 87 (2013) arXiv:1206.7087.
[48] J.R. Gaunt, M. Stahlhofen, J. High Energy Phys. 1412 (2014) 146. arXiv:1409.8281.
[49] F. Feng, Y. Jia, W.-L. Sang, Can NRQCD explain the $\gamma\gamma^* \to \eta_c$ transition form factor data? 2015, arXiv:1505.02665.
[50] A.V. Smirnov, V.A. Smirnov, Comput. Phys. Comm. 184 (2013) 2820–2827. arXiv:1302.5885.
[51] A.V. Smirnov, Comput. Phys. Comm. 185 (2013) 2090–2100. arXiv:1312.3186.
[52] S. Borowka, J. Carter, G. Heinrich, Comput. Phys. Comm. 184 (2012) 396–408. arXiv:1204.4152.
[53] F. Feng, Comput. Phys. Comm. 183 (2012) 2158–2164. arXiv:1204.2314.
[54] K. Chetyrkin, F. Tkachov, Nuclear Phys. B 192 (1981) 159–204.
[55] R.N. Lee, Presenting LiteRed: a tool for the Loop InTEgrals REDuction, 2012, arXiv:1212.2685.
[56] C. Studerus, Comput. Phys. Comm. 181 (2009) 1293–1300. arXiv:0912.2546.
[57] C. Anastasiou, A. Lazopoulos, J. High Energy Phys. 0407 (2004) 046. arXiv:hep-ph/0404258.
[58] F. Feng, Automated one-loop computation in quarkonium process within NRQCD framework, 2013, arXiv:1307.5587.
[59] M.R. Fiorentin, Internat. J. Modern Phys. C (2015) 1650027. arXiv:1507.03527.
[60] J. Küblbeck, M. Böhm, A. Denner, Comput. Phys. Comm. 60 (1990) 165–180.
[61] J.P. Ma, Q. Wang, Phys. Lett. B 537 (2002) 233–240. arXiv:hep-ph/0203082.
[62] N. Brambilla, E. Mereghetti, A. Vairo, J. High Energy Phys. 0608 (2006) 039. arXiv:hep-ph/0604190.
[63] W.-L. Sang, F. Feng, Y. Jia, S.-R. Liang, Next-to-next-to-leading-order QCD corrections to $\chi_{c0,2} \to \gamma\gamma$, 2015, arXiv:1511.06288.
[64] E. Braaten, Y.-Q. Chen, Phys. Rev. D 54 (1996) 3216–3227. arXiv:hep-ph/9604237.
[65] P. Cho, A.K. Leibovich, Phys. Rev. D 53 (1996) 6203–6217. arXiv:hep-ph/9511315.