Bachelor thesis

A handy implementation of Generalized Polylogarithms

by

Luca Naterop

at the
Department of Physics
University of Zürich

Supervisors
Prof. Dr. Adrian Signer
Yannik Ulrich

July 2019

Abstract

Generalized polylogarithms are a class of functions that appear in perturbative quantum field theory. In this work an existing algorithm for the numerical evaluation of generalized polylogarithms is reviewed. In addition, *handyG*, a fast and novel implementation of this algorithm in the Fortran 2008 language is provided and compared to an existing implementation in GiNaC. A notable speedup with respect to the implementation in GiNaC is obtained.

Acknowledgements

I would like to thank Prof. Adrian Signer for giving me the opportunity to work in this great environment that I had been exposed to during my thesis. Moreover I would like to express profound gratitude to Yannick Ulrich for his prompt and useful answers to any question ranging from understanding the algorithm to computer problems and everything in between.

Contents

1 Introduction

Generalized Polylogarithms (GPLs), originally introduced by Kummer [1] and Poincaré [2] are a class of functions made of repeated integrals that appear in many different contexts in high-energy physics [3]. Specifically, in higher order corrections in quantum field theory GPLs often appear through the integration of rational functions. Thus we require a way to numerically evaluate GPLs in an efficient manner.
Important special cases of generalized polylogarithms are harmonic polylogarithms, Nielsen polylogarithms as well as the classical polylogarithms [4, 5]. Fast implementations for these special cases exist for example for the case of Harmonic Polylogarithms [6, 7].
For the case of GPLs there is a Mathematica implementation [8] which evaluates G-functions up to depth four. For GPLs of arbitrary depth an algorithm for the numerical evaluation has been given by Vollinga and Weinzierl [9]. Alongside the algorithm Vollinga and Weinzierl also provided an implementation within the GiNaC symbolic manipulation library for the C++ programming language. This implementation uses the symbolic capabilities of this library and cannot be used indepenently of GiNaC, which itself includes many tools that need to be installed in order to be used. In addition, reyling on symbolic manipulation significantly reduces speed of evaluation.
Hence the idea to build something more standalone and fast, without computer algebra. This work is intended to close this gap. For that purpose the algorithm was reviewed and made explicit enough to allow an implementation in a non-symbolic environment. As a result, a handy and fast implementation for the numerical evaluation of GPLs of any given depth is provided in the Fortran 2008 language.
This work is divided as follows: In Chapter 2 we introduce G-functions, their various notations and we discuss their properties that are used for the numerical evaluation. In Chapter 3 the algorithm for the numerical evaluation is reviewed and explicitly carried out in an example. Finally, in Chapter 4 the Fortran implementation is given, a brief overview of its usage is provided and speed of our implementation is compared to GiNaC.

2 Theoretical background

Generalized Polylogarithms (GPLs) $G(z_1, ..., z_k; y)$ are the objects of interest in this work. Also called G-functions, they are complex-valued functions which depend on the complex parameters $z_1, ..., z_k$ as well as the (complex) argument y. We can define a G-function as a nested integral

$$G(z_1, ..., z_k; y) = \int_0^y \frac{dt_1}{t_1 - z_1} \int_0^{t_1} \frac{dt_2}{t_2 - z_2} ... \int_0^{t_1} \frac{dt_k}{t_k - z_k}. \tag{2.1}$$

Alternatively they can also be defined in recursive form as

$$G(z_1, ..., z_k; y) = \int_0^y \frac{dt_1}{t_1 - z_1} G(z_2, ..., z_k; t_1). \tag{2.2}$$

We wish to reduce G-functions to elementary objects that can be evaluated. For example, if we have only two arguments ($k = 1$) then the integration can be carried out and we obtain a logarithm

$$G(z; y) = \int_0^y \frac{dt_1}{t_1 - z} = \log(y - z) - \log(-z) = \log\left(\frac{z - y}{z}\right) = \log\left(1 - \frac{y}{z}\right). \tag{2.3}$$

Similarly if all the parameters z_i are equal to zero leaving only the argument y, then the G-function evaluates to

$$G(\underbrace{0, ..., 0}_{k}; y) = \frac{(\log y)^k}{k!}. \tag{2.4}$$

We call $G(z_1, ..., z_k; y)$ *flat* since all arguments are explicitly written down. However this notation can be cumbersome if many of the z_i are zero. Then we introduce the *condensed* notation which uses parameters m_i in order to keep track of the number of zeroes before the arguments z_i, allowing us to write

$$G_{m_1,...,m_k}(z_1, ..., z_k; y) = G(\underbrace{0, ..., 0}_{m_1-1}, z_1, ..., z_{k-1}, \underbrace{0, ..., 0}_{m_k-1}, z_k; y). \tag{2.5}$$

Both notations will be used interchangeably. We define the *depth* k of the G-function as the number of non-zero parameters. Moreover we define the *weight* of a G-function as

$$m = \sum_i m_i \tag{2.6}$$

which gives the total number of parameters (in flat form).

Multiple Polylogarithms (MPLs) are a related class of functions that also generalize logartihms. They are defined as an infinite nested series

$$\mathrm{Li}_{m_1,...,m_k}(x_1, ..., x_k) = \sum_{i_1 > ... > i_k}^{\infty} \frac{x_1^{i_1}}{i_1^{m_1}} ... \frac{x_k^{i_k}}{i_k^{m_k}}, \tag{2.7}$$

where $m_1, ..., m_k$ are integer weights. If there is only one argument present, they reduce to classical polylogarithms $\mathrm{Li}_m x$ which for $m = 1$ is simply a logarithm.
MPLs are closely related to GPLs through the relation

$$\mathrm{Li}_{m_1,...,m_k}(x_1, ..., x_k) = (-1)^k G_{m_1,...,m_k}\Big(\frac{1}{x_1}, \frac{1}{x_1 x_2}, ..., \frac{1}{x_1...x_k}; y\Big). \tag{2.8}$$

This relation can be inverted by performing an iterated substitution

$$u_1 = \frac{1}{x_1},\ \ u_2 = \frac{1}{x_1 x_2} = \frac{u_1}{x_1},\ \ ...\ ,\ \ u_k = \frac{1}{x_1...x_k} = \frac{u_{k+1}}{x_k}, \tag{2.9}$$

enabling us to write the GPLs in terms of MPLs

$$G_{m_1,...,m_k}(u_1, ..., u_k; 1) = (-1)^k \mathrm{Li}_{m_1,...,m_k}\Big(\frac{1}{u_1}, \frac{u_1}{u_2}, ..., \frac{u_{k-1}}{u_k}\Big). \tag{2.10}$$

In (2.10) the left-hand side is an integral representation whereas the right-hand side is a series representation. G-functions with arbitrary arguments have one redundant degree of freedom as they satisfy the scaling relation

$$G(z_1, ..., z_k; y) = G(xz_1, ..., xz_k; xy) \tag{2.11}$$

for any number $x \neq 0$. (2.10) assumes the last argument of G being equal to one. Using the scaling relation one can normalize $G(x_1, ..., x_k; y)$ with $x = 1/y$ to guarantee just that.
The series representation is especially convenient for a numerical evaluation. For the numerical evaluation the main idea will be to compute G-functions by reducing them to their corresponding series representation (2.10).

Convergence properties

If we want to use infinite series for numerical evaluation of G-functions, the series needs to be convergent. $\mathrm{Li}_{m_1,...,m_k}(x_1, ..., x_k)$ is convergent if the conditions

$$|x_1...x_k| < 1 \quad \text{and} \quad (m_1, x_1) \neq (1, 1) \tag{2.12}$$

are satisfied. Using the relation (2.10) this translates to a convergence criterion for the integral representation. We find that $G_{m_1,...,m_k}(z_1, ..., z_k; y)$ is convergent if

$$|y| < |z_i| \quad \text{for } i = 1, .., k \ \text{ and } \quad (m_1, y/z_1) \neq (1, 1) \tag{2.13}$$

Thus we require a method to write any G-function in terms of G-functions that are convergent according to (2.13). Such a method was found by [9] and it will be explained in Chapter 3.

Shuffle algebra and trailing zeroes

If $z_k = 0$ in $G_{m_1,...,m_k}(z_1, ..., z_k; y)$ then the convergence criterion is not fulfilled (in that case we cannot write the trailing zeroes in terms of the m_i since the m_i track the amount of zeroes *before* a z_i). Therefore in order to evaluate $G_{m_1,...,m_k}(z_1, ..., z_k; y)$ it is required that $z_k \neq 0$, in other words, there are no trailing zeroes. If there are trailing zeroes we then need a method to remove them. We can exploit the fact that G-functions satisfy the shuffle algebra

$$G(\vec{a}; y)\, G(\vec{b}; y) = \sum_{\vec{c}=\vec{a}\,\sqcup\!\sqcup\,\vec{b}} G(\vec{c}; y). \tag{2.14}$$

The sum in the left-hand side of eq. (2.14) runs over all the shuffles of the list $\vec{a} = (a_1, ..., a_n)$ with the list $\vec{b} = (b_1, ..., b_m)$. The shuffle product gives the set of all permutations of the elements in $\vec{a}$ and $\vec{b}$ that preserve

the respective orderings of $\vec{a}$ and $\vec{b}$
The shuffle product is defined recursively [10] in the following way: If $\vec{w}_1$, $\vec{w}_2$ and $\vec{w}$ are ordered lists and α, β two elements, the shuffle product is defined as

$$(\alpha, \vec{w}_1) \sqcup\!\sqcup (\beta, \vec{w}_2) = \Big(\alpha, \big(\vec{w}_1 \sqcup\!\sqcup (\beta, \vec{w}_2)\big)\Big) + \Big(\beta, \big((\alpha, \vec{w}_1) \sqcup\!\sqcup \vec{w}_2\big)\Big) \tag{2.15}$$

$$() \sqcup\!\sqcup \vec{w} = \vec{w} \sqcup\!\sqcup () = \vec{w} \tag{2.16}$$

where () is an empty list and $(\alpha, \vec{w})$ denotes the concatenation of an element with a list. Here the + operator simply indicates that we end up with two shuffles. To find the shuffle product of two vectors we may use (2.15) recursively. A simple example for the shuffle product of two lists is

$$(a, b) \sqcup\!\sqcup (c) = (a, b, c) + (a, c, b) + (c, a, b) \tag{2.17}$$

The shuffle algebra can be considered as a type of Hopf algebra. Another Hopf algebra is the stuffle algebra which is fulfilled by the series representation $\mathrm{Li}_{m_1,\ldots,m_k}(x_1, \ldots, x_k)$. We shall not discuss the stuffle algebra further as it isn't used in this work.

3 The algorithm

The central idea of numerically evaluating G-functions is to first map their arguments to the domain where the corresponding series representation is convergent (2.13) and to then use the series expansion up to some finite order. Thus we will first look at how to remove trailing zeroes in Section 3.1, and then how to make a G-function without trailing zeroes convergent in Section 3.2. In Section 3.3 we comment on accelerating the convergence of already convergent G-functions. Finally, in Section 3.4 we apply the algorithm to an explicit example.

3.1 Removal of trailing zeroes

Consider a G-function with k parameters, $z_j \neq 0$ and then $k-j$ trailing zeroes, i.e.

$$G(z_1, ..., z_j, \underbrace{0, ..., 0}_{k-j}; y) = G(z_1, ..., z_j, 0_{k-j}; y) \tag{3.1}$$

where 0_n is a short-hand for n zeroes. We now shuffle $\vec{a} = (z_1, ..., z_j, 0_{k-j})$ with $\vec{b} = (0)$ then the resulting shuffles have the extra zero inserted at every place in $\vec{a}$. All the terms in which the zero ends up next to one of the existing trailing zeroes in $\vec{a}$ then have exactly $k-j$ trailing zeroes. Thus these terms which then appear $k-j$ times have the arguments in (3.1). In addition one obtains a sum that runs over the shuffles $\vec{s} = (z_1, ..., z_{j-1}) \sqcup\!\sqcup (0)$:

$$G(0; y)G(z_1, ..., z_j, 0_{k-j-1}; y) = (k-j)G(z_1, ..., z_j, 0_{k-j}; y) + \sum_{\vec{s}} G(s_1, ..., s_j, z_j, 0_{k-j-1}; y) \tag{3.2}$$

Solving for the G-function with $(k-j)$ trailing zeroes and using $G(0, y) = \log(y)$ yields

$$G(z_1, ..., z_j, 0_{k-j}; y) = \frac{1}{k-j}\Big(G(0; y)\, G(z_1, ..., z_j, 0_{k-j-1}; y) - \sum_{\vec{s}} G(s_1, ..., s_j, z_j, 0_{k-j-1}; y)\Big) \tag{3.3}$$

The right-hand side of (3.3) contains only terms that have at most $k-j-1$ trailing zeroes. Thus applying (3.3) recursively we remove all trailing zeroes.

3.2 Making G-functions convergent

In this section we discuss the core of the algorithm [9] to make arbitrary G-functions convergent.

3.2.1 Reduction to pending integrals

Consider a G-function of the form

$$G(a_1, ..., a_{i-1}, s_r, a_{i+1}, ..., a_w; y) \tag{3.4}$$

where s_r has the smallest absolute value among all the non-zero arguments in G. If we have $|s_r| < |y|$ (3.4) is not convergent. Thus we would like to get rid of s_r to obtain simpler G-functions. This we can achive by applying the fundamental theorem of calculus to generate terms where s_r is either integrated over or not present anymore

$$G(a_1,...,a_{i-1},s_r,a_{i+1},...,a_w;y) = G(a_1,...,a_{i-1},0,a_{i+1},...,a_w;y) + \int_0^{s_r} ds_{r+1}\frac{\partial}{\partial s_{r+1}}G(a_1,...,a_{i-1},s_{r+1},a_{i+1},...,a_w;y). \tag{3.5}$$

Even though this looks more complicated, for the second term we use partial fraction decomposition and integration by parts. Then we obtain different results depending on wether s_r is first in the list of arguments or not.

If s_r appears first in the list (i.e. $i = 1$) we find

$$G(s_r,a_2,...,a_w;y) = G(0,a_2,..,a_w;y) + \overbrace{\int_0^{s_r}\frac{ds_{r+1}}{s_{r+1}-y}}^{G(y;s_r)} G(a_2,...,a_w;y) + \underbrace{\int_0^{s_r}\frac{ds_{r+1}}{s_{r+1}-a_2}G(s_{r+1},a_3,..,a_w;y)}_{\text{pending integral}} - \underbrace{\int_0^{s_r}\frac{ds_{r+1}}{s_{r+1}-a_2}}_{G(a_2;s_r)} G(a_2,...,a_w;y). \tag{3.6}$$

Now, in the first term, s_r is absent. Therefore the resulting G-function is simpler (it might still be non-convergent, but we will use this method recursively on the resulting G-functions until we end up with convergent G-functions). In the second and fourth terms s_{r+1} does not appear in $G(...)$, and the integral corresponds to a G-function of depth one, which is just a logarithm. The third term does have the integration variable s_{r+1} in $G(...)$ and therefore yields what we refer to as a pending integral. This object is not a single G-function anymore, but a more general object which we will discuss in Section 3.2.2. Note that all G-functions have depth reduced by one (also the first one due to the zero being condensed).

If s_r appears in the middle of the list (i.e. $1 < i < w$) we find

$$\begin{aligned} G(a_1,...,a_i-1,s_r,a_{i+1},...,a_w;y) = &\\ &+ G(a1,...,a_{i-1},0,a_{i+1},...,a_w;y)\\ &- \int_0^{s_r}\frac{ds_{r+1}}{s_{r+1}-a_{i-1}}G(a_1,...,a_{i-2},s_{r+1},a_{i+1},...,a_w;y)\\ &+ \underbrace{\int_0^{s_r}\frac{ds_{r+1}}{s_{r+1}-a_{i-1}}}_{G(a_{i-1},s_r)}G(a_1,...,a_{i-1},a_{i+1},...,a_w;y)\\ &+ \int_0^{s_r}\frac{ds_{r+1}}{s_{r+1}-a_{i+1}}G(a_1,...,a_{i-1}s_{r+1},a_{i+2},...,a_w;y)\\ &- \underbrace{\int_0^{s_r}\frac{ds_{r+1}}{s_{r+1}-a_{i+1}}}_{G(a_{i+1},s_r)}G(a_1,...,a_{i-1},a_{i+1},...,a_w;y). \end{aligned} \tag{3.7}$$

And again we obtain simpler G-functions (without s_r or lower depth) as well as pending integrals.

If s_r appears last in the list (i.e. $i = w$) we use the shuffle algebra to remove s_r from the last place. Consider the case where we have $m-1$ zeroes before s_r, i.e.

$$G(a_1,...,a_k 0_{m-1},s_r;y) \tag{3.8}$$

which is the first term produced if we shuffle

$$G(a_1, ..., a_k; y)\, G(0_{m-1}, s_r; y)\,. \tag{3.9}$$

Therefore, if we perform the shuffle and solve for (3.1) we find

$$G(a_1, ..., a_k, 0_{m-1}, s_r; y) = G(a_1, ..., a_k; y) G(0_{m-1}, s_r; y) - \sum_{\vec{c}} G(\vec{c}; y) \tag{3.10}$$

where in the sum we omit the first term of the shuffle

$$\vec{c} = (a_1, ..., a_k) \sqcup\!\sqcup\; (0_{m-1}, s_r) \text{ without } (a_1, ..., a_k, 0_{m-1}, s_r)\ . \tag{3.11}$$

Thus the G-function with s_r appearing last has been reduced to a term without s_r, a G-function of depth one and G-functions where s_r is either in the middle or at the beginning.

3.2.2 Evaluation of pending integrals

The steps from the last section are repeated recursively also for the G-functions under pending integrals. The most general term created by this algorithm is then of the form

$$\mathrm{PI}(\vec{p} = (y_1, \vec{b}), i, \vec{g} = (\vec{a}, y)) \int_0^{y_1} \frac{ds_1}{s_1 - b_1} \int_0^{s_1} \frac{ds_2}{s_2 - b_2} ... \int_0^{s_{r-1}} \frac{ds_r}{s_r - b_r} G(a_1, ..., a_{i-1}, s_r, a_{i+1}, ..., a_w; y)\,. \tag{3.12}$$

Here we have adopted the convention that if $i = 0$ then that means that the integration variable does not appear inside of the G-function. An example is

$$\mathrm{PI}(\vec{p} = (1, 2, 3), 0, (4, 5)) = \int_0^1 \frac{ds_1}{s_1 - 2} \frac{ds_2}{s_2 - 3} G(4; 5) \tag{3.13}$$

and another example where the integration variable *does* appear in G is

$$\mathrm{PI}(\vec{p} = (1, 2, 3), 2, (4, 5)) = \int_0^1 \frac{ds_1}{s_1 - 2} \frac{ds_2}{s_2 - 3} G(4, s_2; 5)\,. \tag{3.14}$$

As we use (3.6), (3.7) and (3.10) for the G-function inside of the pending integral we can express the resulting terms with G-functions and pending integrals. To give an example, we obtain for the case where s_r appears first

$$\begin{aligned}
\mathrm{PI}(\vec{p} = (y_1, \vec{b}), 1, \vec{g} = (\vec{a}, y)) &= \int_0^{y_1} \frac{ds_1}{s_1 - b_1} ... \int_0^{s_{r-1}} \frac{ds_r}{s_r - b_r} G(s_r, ..., a_{i+1}, ..., a_w; y) = \\
&+ \underbrace{\int_0^{y_1} \frac{ds_1}{s_1 - b_1} ... \int_0^{s_{r-1}}}_{\mathrm{PI}(\vec{p},0,())} G(0, a_{i+1}, ..., a_w; y) \\
&+ \underbrace{\int_0^{y_1} \frac{ds_1}{s_1 - b_1} ... \int_0^{s_{r-1}} \frac{ds_r}{s_r - b_r} \int_0^{s_r} \frac{ds_{r+1}}{s_{r+1} - y}}_{\mathrm{PI}((\vec{p},y),0,())} G(a_{i+1}, ..., a_w; y) \\
&+ \underbrace{\int_0^{y_1} \frac{ds_1}{s_1 - b_1} ... \int_0^{s_{r-1}} \frac{ds_r}{s_r - b_r} \int_0^{s_r} \frac{ds_{r+1}}{s_{r+1} - a_{i+1}} G(s_{r+1}, a_{i+2}, ..., a_w; y)}_{\mathrm{PI}((\vec{p},a_{i+1}),1,(a_{i+1},...,a_w;y))} \\
&- \underbrace{\int_0^{y_1} \frac{ds_1}{s_1 - b_1} ... \int_0^{s_{r-1}} \frac{ds_r}{s_r - b_r} \int_0^{s_r} \frac{ds_{r+1}}{s_{r+1} - a_{i+1}}}_{\mathrm{PI}((\vec{p},a_{i+1}),0,())} G(a_{i+1}, ..., a_w; y).
\end{aligned} \tag{3.15}$$

Here (a, b) denotes the concatenation into a vector of a and b. All other cases of reduction for pending integrals are not written out here explicitly but instead the results are given in Appendix A.
As we recursively apply (3.6), (3.7) and (3.10) we increase the number of pending integrals in front but decrease the depth of the G-functions by one unit in every recursion step. This scheme is performed until we hit any one of a number of base cases which are conditions for the G-function inside the pending integral:

1. $G_m(s_{r\pm}; y)$ is of depth one,
2. s_r appears in $G(...)$ at the last place as argument y, or
3. there is no G-function around anymore in the pending integral.

In the following we discuss these base cases in detail and see what they evaluate to.

Case 1.

For GPLs of depth one i.e. $G_m(s_{r\pm}; y)$ we have if $m = 1$

$$G_1(s_{r\pm}; y) = G_1(y_{2\mp}; s_r) - G(0; s_r) + \log(-y) \tag{3.16}$$

Here it has to be remembered that we have pending integrals in front, thus each term gives again a simpler pending integral. The first and second terms reduce to (2) and the third term is case (3). If $m > 1$ use

$$G_m(s_{r\pm}; y) = -\zeta_m + \int_0^y \frac{dt}{t} G_{m-1}(t_\pm; y) - \int_0^{s_r} \frac{dt}{t} G_{m-1}(t_\pm; y). \tag{3.17}$$

The first summand is case (b) and the second and third terms are pending integrals with G-functions with reduced weight. Recursively using (3.17) thus yields pending integrals with $G_1(...)$ functions.

Case 2.

In this case what we end up with is simply one large G-function:

$$\begin{aligned}&\int_0^{y_1} \frac{ds_1}{s_1 - b_1} ... \int_0^{s_{r-1}} \frac{ds_r}{s_r - b_r} G(\vec{a}; s_r) = \\ &\int_0^{y_1} \frac{ds_1}{s_1 - b_1} ... \int_0^{s_{r-1}} \frac{ds_r}{s_r - b_r} \int_0^{s_r} \frac{ds_{r+1}}{s_{r+1} - a_1} ... \int_0^{s_{|}r+w-1} \frac{ds_{r+w-1}}{s_{r+w-1} - a_w} = G((\vec{b}, \vec{a}); y_1).\end{aligned} \tag{3.18}$$

Case 3.

When there is no G-function under the pending integral then the integrals evaluate to a G-function:

$$\int_0^{y_1} \frac{ds_1}{s_1 - b_1} ... \int_0^{s_{r-1}} \frac{ds_r}{s_r - b_r} = G(b_1, ..., b_r; y_1) \tag{3.19}$$

In each case we end up with G-functions that are simpler in the sense that s_r has been eliminated. These might still be non-convergent due to another (non-zero) z_i element being smaller in absolute value than y. But applying the removal of s_r recursively we can eliminate all z_i for which $|z_i| < |y|$. Therefore in the end we always obtain convergent G-functions.

3.3 Increase rate of convergence

When a G-function is convergent then that does not imply that the convergence is fast. To increase the rate of convergence we can use the fact that G-functions satisfy the Hölder convolution

$$G(z_1,...,z_k;1) = \sum_{j=0}^{k} (-1)^j G(1-z_j,...,1-z_1;1-1/p) G(z_{j+1},...,z_k;1/p). \tag{3.20}$$

which holds for any value of p. Writing out explicitly the first and last term of the sum we obtain

$$\begin{aligned} G(z_1,...,z_k;1) &= G(z1,..,z;1/p) + (-1)^k G(1-z_k,...,1-z_1;1-1/p) \\ &+ \sum_{j=1}^{k-1} (-1)^j G(1-z_j,...,1-jz_1;1-1/p) G(z_{j+1},...,z_k;1/p) \end{aligned} \tag{3.21}$$

We begin by normalizing a convergent G-function to $y = 1$. The convergence is slow if some z_i is close to the unit circle; that is, if

$$1 \leq |z_i| \leq \lambda \tag{3.22}$$

where we typically use for instance $\lambda = 1.1$. Next we apply the Hölder convolution for $p = 2$. A proof for why this method increases the rate of convergence of all terms and does not lead to an infinite recursion is not given here as it is non-trivial, see [9].

3.4 An example reduction

For clarity we include here an example of how the algorithm works. For this purpose we reduce $G(1, 0, 3; 2)$ according to this algorithm until we end up with logarithms, polylogs and convergent multiple polylogaritms. In our notation of a non-convergent G-function we have

$$G(\underbrace{1}_{s_r}, \underbrace{0}_{a_2}, \underbrace{3}_{a_3}, \underbrace{2}_{y}) = G(0, 0, 3; 2) + \int_0^1 ds_1 \frac{\partial}{\partial s_1} G(s_1, 0, 3; 2) . \tag{3.23}$$

The first term corresponds to $G_3(3; 2)$ and therefore it is a convergent trilogarithm. The second term has s_r appearing at the first place. Using (3.6) we obtain for the second term

$$\int_0^1 ds_1 \frac{\partial}{\partial s_1} G(s_1, 0, 3; 2) = \int_0^1 \frac{ds_1}{s_1 - 2} G(0, 3; 2) + \int_0^1 \frac{ds_1}{s_1 - 0} G(s_1, 3; 2) - \int_0^1 \frac{ds_1}{s_1 - 0} G(0, 3; 2) \tag{3.24}$$

The first and last terms are products of a logarithm with a dilog which can be readily evaluated. What remains is discussing how to evaluate the second term which involves the integration variable s_1 inside of the G-function. It is a pending integral. In order to evaluate it, we apply again (3.6) to the G-function under the pending integral to find

$$G(s_1, 3; 2) = G(0, 3; 2) + \int_0^{s_1} \frac{ds_2}{s_2 - 2} G(3; 2) + \int_0^{s_1} \frac{ds_2}{s_2 - 3} G(s_2; 2) - \int_0^{s_1} \frac{ds_2}{s_2 - 3} G(3, 2) . \tag{3.25}$$

Replacing in 3.24 and simply writing out again gives

$$\begin{aligned} \int_0^1 \frac{ds_1}{s_1 - 0} G(s_1, 3; 2) = \int_0^1 \frac{ds_1}{s_1} G(0, 3; 2) + \int_0^1 \frac{ds_1}{s_1} \int_0^{s_1} \frac{ds_2}{s_2 - 2} G(3; 2) + \\ \int_0^1 \frac{ds_1}{s_1} \int_0^{s_1} \frac{ds_2}{s_2 - 3} G(s_2; 2) - \int_0^1 \frac{ds_1}{s_1} \int_0^{s_1} \frac{ds_2}{s_2 - 3} G(3, 2) . \end{aligned} \tag{3.26}$$

Here the first term is the product of a logarithm times a dilogarithm. The second and fourth terms are products of a dilogarithm with a logarithm. Thus we are left with discussing the evaluation of the third term which is a pending integral. But this time what we have is the depth one base case as the G-function under the integral is of depth one. Thus for the third term we write

$$\begin{aligned} &\int_0^1 \frac{ds_1}{s_1} \int_0^{s_1} \frac{ds_2}{s_2 - 3} G(s_2; 2) = \\ &\int_0^1 \frac{ds_1}{s_1} \int_0^{s_1} \frac{ds_2}{s_2 - 3} \Big(G(2; s_2) - G(0; s_2) + \log(-2) \Big) . \end{aligned} \tag{3.27}$$

The first term has s_r as the argument y and the whole thing is $G(0, 3, 2; 1)$ which reduces to a convergent multiple polylogarithm $Li_{2,1}$. In the last term the G-function is independent of s_r and the integrals in front are $G(0, 3, 1)$ which is a dilogarithm. Finally, the second term is

$$-\int_0^1 \frac{ds_1}{s_1} \int_0^{s_1} \frac{ds_2}{s_2 - 3} \int_0^{s_2} \frac{ds_3}{s_3} = -G(0, 3, 0; 1) \tag{3.28}$$

This G-function has one trailing zero. To remove it, we shuffle $G(0, 3; 1)$ with $G(0; 1)$ to find

$$G(0, 3; 1) G(0; 1) = \sum_{\vec{c} = (0,3) ⧢ (0)} G(\vec{c}; 1) = G(0, 3, 0; 1) + G(0, 0, 3; 1) \times 2 \tag{3.29}$$

Therefore

$$-G(0,3,0;1) = -G(0,3;1)G(0;1) + 2\times G(0,0;3,1) \tag{3.30}$$

where the first term is the product of a dilogarithm with a logarithm and the second is a convergent multiple polylogarithm Li_3. Gathering all terms we obtain

$$\begin{aligned}
&G(1,0,3;2) = \\
&\underbrace{G(0,0,3;2)}_{-\mathrm{Li}_3(2/3)} + \underbrace{G(2;1)}_{\log(1/2)}\underbrace{G(0,3;2)}_{-\mathrm{Li}_2(2/3)} - \underbrace{G(0;1)}_{\log(1)=0}G(0,3;2) \\
&+ \underbrace{G(0;1)}_{0}G(0,3;2) + \underbrace{G(0,2;1)}_{-\mathrm{Li}_2(1/3)}\underbrace{G(3;2)}_{\log(1/3)} - \underbrace{G(0,3;2)}_{-\mathrm{Li}_2(2/3)}\underbrace{G(3;2)}_{\log(1/3)} \\
&+ \underbrace{G(0,3,2;1)}_{\mathrm{Li}_{2,1}(1/3,3/2)} + \underbrace{G(0,3;1)}_{-\mathrm{Li}_2(1/3)}\log(-2) + G(0,3;1)\underbrace{G(0,1)}_{0} + 2\underbrace{G(0,0,3;1)}_{-\mathrm{Li}_3(1/3)} = \\
&-\mathrm{Li}_3(2/3) - \log(1/2)\mathrm{Li}_2(2/3) - \mathrm{Li}_2(1/2)\log(1/3) + \mathrm{Li}_2(1/3)\log(1/3) + \mathrm{Li}_{2,1}(1/3,3/2) \\
&-\mathrm{Li}_2(1/3)\log(-2) - 2\mathrm{Li}_3(1/3) \\
&= -0.81809 - 1.15049i
\end{aligned} \tag{3.31}$$

4 Implementation in Fortran: handyG

We have implemented the algorithm in a Fortran 2008 library called handyG. It is available for public use under the GNU GPLv3 license.
Our implementation is available on Github under `https://github.com/lucnat/handyG` and can be downloaded with

```
$ git clone https://github.com/lucnat/handyG
```

which clones the repository to the current directory. HandyG supports the three evaluation modes

1. Direct evaluation from within Fortran
2. Evaluation from G-functions given by a text file
3. Evaluation from within Mathematica through an interface

The `README.md` file in the root directory gives information on how to use the code to evaluate G-functions. The first thing to do is to run the `configure` script. To get help on how to use it run

```
$ ./configure --help
```

Comparison to GiNaC

The evaluation speed was compared with that of the GiNaC implementation. Here both implementations were tested on the G-functions from [11, 12, 13] and the evaluation speed was measured. A histogram over all G-functions indicating the amount of G-functions that were evaluated at different evaluation times is given in figure 4.1.

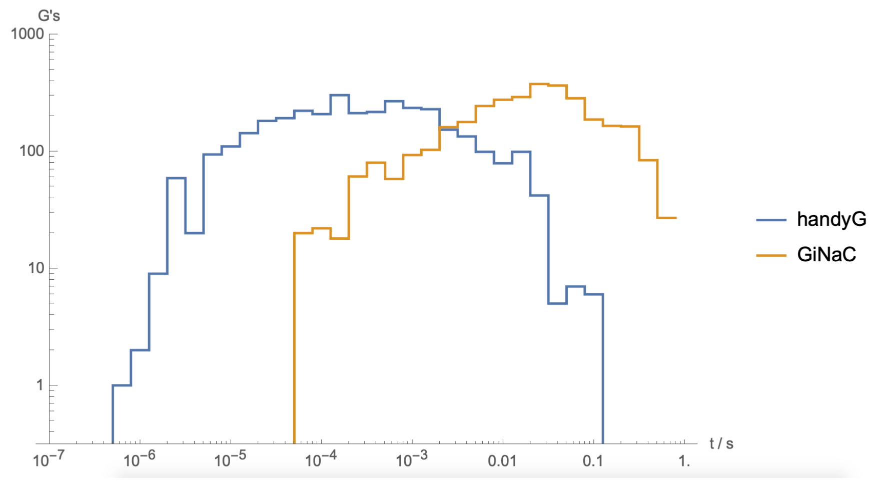

Figure 4.1: HandyG versus GiNaC evaluation speed comparison. HandyG is about 30 times faster.

5 Conclusion

We have reviewed the algorithm by Vollinga and Weinzierl. We have made it explicit enough to allow for an implementation in Fortran 2008. The resulting implementation is publicly available and can be used to evaluate GPLs in different environments. It is many times faster than existing implementations.

A Reduction formulas for pending integrals

Here we provide the formulas for recursively reducing the pending integrals generated by the algorithm to simpler pending integrals, G-functions and other objects.

If G is of depth one and m = 1

$$\begin{aligned} \mathrm{PI}(\vec{p}, 1, y_2) &= \int_0^{y_1} \frac{ds_1}{s_1 - b_1} \dots \int_0^{s_{r-1}} \frac{ds_r}{s_r - b_r} \qquad \underbrace{G_1(s_{r\pm}; y_2)}_{G_1(y_{2\mp};s_r)-G(0;s_r)+ln(-y_2)} \\ &= \mathrm{PI}(\vec{p}, 2, y_{2\pm}) - \mathrm{PI}(\vec{p}, 2, 0) + ln(-y_{2\mp}) G(\vec{b}, y_1) \end{aligned} \tag{A.1}$$

If G is of depth one and m > 1

$$\begin{aligned} &\mathrm{PI}(\vec{p}, m, (\underbrace{0, ..., 0}_{m-1}, y_2)) = \\ &- \zeta_m \mathrm{PI}(\vec{p}, 0, ()) + \mathrm{PI}((y_2, 0), m-1, (\underbrace{0, ..., 0}_{m-2}, y_2)) \mathrm{PI}(\vec{p}, 0, ()) - \mathrm{PI}((\vec{p}, 0), m-1, (\underbrace{0, ..., 0}_{m-2}, y_2)) \end{aligned} \tag{A.2}$$

If s_r is at the last place then $i = size(\vec{g}) + 1$ and we obtain a G-function

$$\mathrm{PI}((\vec{p} = (y_1, \vec{b})), 1, \vec{g} = (\vec{a}, y_2)) = G((\vec{b}, \vec{g}); y_1) \tag{A.3}$$

If s_r is not inside G then $i = 0$ an the integrals in front give us a G-function

$$\mathrm{PI}(\vec{p} = (y_1, \vec{b}), 0, \vec{g}) = G(\vec{b}, y_1) G(\vec{a}, y_2) \tag{A.4}$$

If G is of higher depth and s_r at the beginning
This case was written out eplicitly in the chapter 4. Here we give the result:

$$\begin{aligned} &\mathrm{PI}((\vec{p} = (y_1, \vec{b})), 1, \vec{g} = (\vec{a}, y_2)) = \\ &+ \mathrm{PI}(\vec{p}, 0, ())\, G(0, a_{i+1}, ..., a_w; y_2) \\ &+ \mathrm{PI}((\vec{p, y_2}), 0, ())\, G(a_{i+1}, ..., a_w; y_2) \\ &+ \mathrm{PI}((\vec{p}, a_{i+1}), 1, (a_{i+2}, ..., a_w; y_2)) \\ &- \mathrm{PI}((\vec{p}, a_{i+1}), 0, ())\, G(a_{i+1}, ..., a_w; y_2) \end{aligned} \tag{A.5}$$

If G is of higher depth and s_r in the middle

$$\begin{aligned}
&\mathrm{PI}(\vec{p}=(y_1,\vec{b}), i, \vec{g}=(\vec{a},y_2)) = \\
&+\mathrm{PI}(\vec{p},0,())\,G(a_1,...,a_{i-1},0,a_{i+1},...,a_w;y_2) \\
&-\mathrm{PI}((\vec{p},a_{i-1}),i-1,(a_{i+1},...,a_w,y_2)) \\
&+\mathrm{PI}((\vec{p},a_{i-1}),0,())\,G(a_1,...,a_{i-1},a_{i+1},...,a_w;y_2) \\
&+\mathrm{PI}((\vec{p},a_{i+1}),i,(a_1,...,a_{i-1},a_{i+2},...,a_w,y_2) \\
&-\mathrm{PI}((\vec{p},a_{i+1}),1,())\,G(a_1,...,a_{i-1},a_{i+1},...,a_w)
\end{aligned} \tag{A.6}$$

Bibliography

[1] E. E. Kummer, *Ueber die transcendenten, welche aus wiederholten integrationen rationaler formeln entstehen*, *Journal für Math. (Crelle), vol. 21, p. 74* (1840) .

[2] H. Poincaré, *Sur les groupes des équations linéaires*, *Acta mathematica, vol. 4, p. 215* (1883) .

[3] A. B. Goncharov, *Multiple polylogarithms, cyclotomy and modular complexes*, *Math Res. Letters 5, (1998) 497-516* (2011) [arXiv:1105.2076].

[4] L. Lewin, *Polylogarithms and associated functions*, *(North Holland, Amsterdam, 1981)* .

[5] N. Nielsen*Nova Acta Leopoldina (Halle) 90, 123 (1909)* (2011) .

[6] T. Gehrmann and E. Remiddi, *Numerical evaluation of harmonic polylogarithms*, arXiv:hep-ph/0107173.

[7] D. Maitre, *HPL, a mathematica implementation of the harmonic polylogarithms*, *Comput. Phys. Commun.* **174** (2006) 222 [hep-ph/0507152].

[8] H. Frellesvig, D. Tommasini and C. Wever, *On the reduction of generalized polylogarithms to Li_n and $Li_{2,2}$ and on the evaluation thereof*, arXiv:1601.02649.

[9] J. Vollinga and S. Weinzierl, *Numerical evaluation of multiple polylogarithms*, arXiv:hep-ph/0410259.

[10] C. Duhr and F. Dulat, *Polylogtools - polylogs for the masses*, 2019.

[11] L.-B. Chen, *Two-loop master integrals for heavy-to-light form factors of two different massive fermions*, arXiv:1801.01033.

[12] S. D. Vita, S. Laporta, P. Mastrolia, A. Primo and U. Schubert, *Master integrals for the nnlo virtual corrections to μe scattering in qed: the non-planar graphs*, arXiv:1806.08241.

[13] P. Mastrolia, M. Passera, A. Primo and U. Schubert, *Master integrals for the nnlo virtual corrections to μe scattering in qed: the planar graphs*, arXiv:1709.07435.