Algorithms for the symbolic integration of hyperlogarithms with applications to Feynman integrals

Erik Panzer
Humboldt-Universität zu Berlin, Institut für Physik, Newton Straße 15, 12489 Berlin, Germany

Abstract
We provide algorithms for symbolic integration of hyperlogarithms multiplied by rational functions, which also include multiple polylogarithms when their arguments are rational functions. These algorithms are implemented in Maple and we discuss various applications. In particular, many Feynman integrals can be computed by this method.

Keywords: Feynman integrals, hyperlogarithms, polylogarithms, computer algebra, symbolic integration, ε-expansions

1. Introduction
An important class of special functions is given by multiple polylogarithms \( \mathbb{H} \) \( [1,2] \),

\[
\text{Li}_{n_1,\ldots,n_r}(z_1,\ldots,z_r) := \sum_{0<k_1<\cdots<k_r} \frac{z_1^{k_1} \cdots z_r^{k_r}}{k_1^{n_1} \cdots k_r^{n_r}} \quad (1.1)
\]

of several complex variables \( \vec{z} \), which generalize the traditional polylogarithms \( \text{Li}_n(z) \) of a single variable (the case

---

Email address: panzer@mathematik.hu-berlin.de (Erik Panzer)

Preprint submitted to Computer Physics Communications

March 14, 2014

References
[1] Maple 16[1] Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario.
\( r = 1 \) studied for example in [3]. Many properties and relations of these multivalued functions can be formulated and studied conveniently in terms of combinatorial structures, which renders them suitable for symbolic algorithms that can be implemented on a computer.

This is mainly a consequence of their representation as a special class of iterated integrals [4] and our preferred basis are the classic hyperlogarithms [5] of

**Definition 1.1.** Given a finite set \( \Sigma \subseteq C \) containing \( 0 \in \Sigma \), each word \( w = \omega_1 \ldots \omega_n \in \Sigma^* \) (\( \omega_\sigma \) denotes the letter for \( \sigma \in \Sigma \)) defines the hyperlogarithm \( L_w \) by setting \( L_w^0(z) := \frac{\log^n z}{n!} \) and otherwise recursively applying

\[
L_{w_{\sigma} w'} := \int_0^z \frac{dz'}{z' - \sigma} L_w(z'). \tag{1.2}
\]

We also abbreviate \( L_{\sigma_1 \ldots \sigma_n} := L_{\omega_{\sigma_1} \ldots \omega_{\sigma_n}} \) and write \( \sigma^{(n)} \) for a sequence \( \sigma, \ldots, \sigma \) of \( n \) letters \( \sigma \).

These functions are also referred to as generalized harmonic polylogarithms (with linear weights) or Goncharov polylogarithms, since they relate to (1.1) via

\[
(-1)^\overline{n} \text{Li}_n(z) = L_{0^{(n_1 - 1)}, \sigma_1, \ldots, 0^{(n_r - 1)}, \sigma_r, 0^{(n_1 - 1)}, \sigma_1}(z) \tag{1.3}
\]

where \( \overline{n} = (n_1, \ldots, n_r) \in \mathbb{N}^r \) and \( \sigma_1, \ldots, \sigma_r \neq 0 \) are such that \( z' = \left( \frac{2}{\sigma_1}, \frac{2}{\sigma_2}, \ldots, \frac{2}{\sigma_r} \right) \), equivalently \( \sigma_1 = z \prod_{i=1}^r z_i^{-1} \). Particle physicists observed special classes of hyperlogarithms in results of Feynman integral calculations. The most famous example is the case when \( \Sigma \subseteq \{-1, 0, 1\} \), called harmonic polylogarithms in [6], and practical tools to compute with these are available like [7] [8]. Some algorithms for general hyperlogarithms are also implemented in [9]. However, the full power of definition 1.1 can be used not only to express the result of Feynman integrals, but actually to compute them in the first place.

Namely, the study [10] of periods of moduli spaces of curves of genus zero computed multiple integrals

\[
f_k := \int_0^\infty f_{k-1}(z_k) \, dz_k = \int_0^\infty dz_1 \cdots \int_0^\infty dz_k \, f_0 \tag{1.4}
\]
of certain polylogarithms \( f_0(z) \) such that each of the partial integrals \( f_k \) is a hyperlogarithm in the next integration variable \( z_{k+1} \). This criterion on \( f_0 \) is called **linear reducibility** in [11], where the symbolic integration algorithm of such functions is explained and applied theoretically to some finite scalar single-scale Feynman integrals (massless propagators). In [12] it was further shown that linear reducibility is actually fulfilled for an infinite family of non-trivial Feynman integrals, but still explicit results were missing.

This technique has then practically been used in [13] to compute off-shell three-point functions and in [14][15] to calculate operator insertions into propagator graphs containing a single non-zero mass scale. A further application to phase-space integrals related to Higgs production can be found in [17].

Unfortunately, none of these programs was made publicly available so far. This might partly be due to the fact that the exposition in [11] does not provide a simple method to obtain certain integration constants in a crucial intermediate step of the algorithm. In fact, [17] resorts to numeric evaluations to guess these constants and a similar approach is common to many applications of the symbol- and coproduct-calculus [15][20]. Also within the method of differential equations [21], boundary conditions occur that must be obtained separately, e.g. through physical reasoning or separate computations of expansions in certain limits.

We close this gap and provide a complete implementation of the method [11] of symbolic integration using hyperlogarithms in the computer algebra system Maple [22]. This program was used in [23][24] to compute several non-trivial Feynman integrals (including divergences and complicated kinematics) and we hope that it will prove helpful in further applications by physicists and mathematicians alike.

Since our foremost goal was to supply a tool for the computation of Feynman integrals, we did not aim for a most general computer algebra framework to handle hyperlogarithms but instead focused on this particular application. Still, the algorithms were implemented for very general situations and may be used for different problems as well.

For completeness let us mention that while we focus on polylogarithms as iterated integrals, the representation (1.1) as nested sums opens the door to completely different strategies like [25] with implementations readily available [26][27]. A lot of progress is being made on symbolic manipulation of sums and we like to point out [9] and the numerous references therein. However, we will not pursue this approach in our work.

### 1.1. Plan of the paper

In section 2 we present our algorithms to symbolically manipulate hyperlogarithms in sufficient detail so as to make an implementation straightforward. We follow the ideas of [11] where the reader might find illuminating examples and details. Our main original contribution is section 2.5 where we solve the problem of determination of integration constants mentioned above.

The Maple implementation HyperInt is presented in section 3 and includes examples of its application to integration problems and for transformations of arguments of polylogarithms.

To apply these methods to multiple integrals (1.4), we review the property of linear reducibility in section 4 and explain how to exploit the polynomial reduction algorithm contained in HyperInt.

Section 5 is devoted to our original motivation and main application: the calculation of Feynman integrals. In HyperInt we supply a couple of commands to facilitate the work with Feynman graphs. Detailed examples
and demonstrations are contained in the attached Maple worksheet Manual.mw.

To ensure correctness of our program, we performed plenty of tests. Some of them are summarized in Appendix A and provided in the file HyperTests.mpl.

Some combinatorial proofs were delegated to Appendix B and we supply a short reference of functions and options provided by HyperInt in Appendix C.

2. Algorithms for hyperlogarithms

We already mentioned references on hyperlogarithms, multiple polylogarithms and iterated integrals. In section 2.1 we collect some standard results and fix our notations.

Afterwards we follow the ideas of [11] for the integration of hyperlogarithms and explain in detail how each step can be implemented combinatorially. In short, to compute \( f_k = \int_0^\infty dz_k f_{k-1}(z_k) \) for some polylogarithm \( f_{k-1} \), we proceed in three steps:

1. Express \( f_{k-1} \) as a hyperlogarithm in \( z_k \).
2. Find a primitive \( F_{k-1}(z_k) \) such that \( \partial_{z_k} F_{k-1} = f_{k-1} \).
3. Evaluate the limits \( f_k = \lim_{z_k \to \infty} F_k - \lim_{z_k \to 0} F_k \).

Many of these operations are straightforward or explained with examples in [11]. All the work actually lies in step 1 above, which is the subject of the reviewing section 2.4 and our additional algorithm of section 2.5 to symbolically compute constants of integration.

The original work [10] in the setting of moduli spaces contains a lot of details, worked examples and geometric interpretations of the ideas employed. In particular we recommend sections 5 and 6 therein which develop the logarithmic regularization in detail.

Let us remark that instead of tracking a sequence \([1, 4]\) of one-dimensional iterated integrals, the natural approach would be to consider every \( f_k \) as an iterated integral in several variables \( z_{k+1}, z_{k+2}, \ldots \) simultaneously. This idea is pursued in [28] and their authors are currently finalizing an implementation of this method as well.

2.1. The tensor algebra and iterated integrals

The algebraic avatar of iterated integrals is the shuffle algebra

\[
T(\Sigma) := \bigoplus_{w \in \Sigma^*} \mathbb{Q} w = \bigoplus_{n=0}^\infty T_n(\Sigma), \quad T_n(\Sigma) := (\mathbb{Q} \Sigma)^\otimes n
\] (2.1)

spanned by all words over the alphabet \( \Sigma \); some references for this Hopf algebra are [29] [30]. It is graded by the weight \( n = |w| \) counting the number of letters in a word \( w = \omega_1 \ldots \omega_n \in \Sigma^n \). Apart from the non-commutative concatenation product, it is equipped with the commutative shuffle product defined recursively by

\[
(\omega_\sigma w) \shuffle (\omega_\tau w') := \omega_\sigma (w \shuffle w') + \omega_\tau (\omega_\sigma w \shuffle w')
\] (2.2)

until \( 1 \shuffle w = w \shuffle 1 = w \) where 1 denotes the empty word which is the identity of \( T(\Sigma) \). The coproduct \( \Delta : T(\Sigma) \to T(\Sigma) \otimes T(\Sigma) \) of interest is the deconcatenation

\[
\Delta(\omega_1 \ldots \omega_n) := \sum_{k=0}^n \omega_1 \ldots \omega_k \otimes \omega_{k+1} \ldots \omega_n
\] (2.3)

These combinatorial structures precisely capture the analytic properties of the iterated integrals [4, 31]

\[
\int_\gamma \omega_1 \ldots \omega_n := \int_0^1 \frac{d\gamma(z)}{\gamma(z) - \sigma_1} \int_{\gamma[z, z]} \omega_2 \ldots \omega_n
\] (2.4)

differential one-forms \( \omega_\sigma = \frac{dz_\sigma}{z_\sigma} \in \Omega^1(\mathbb{C} \setminus \Sigma) \) of a single variable \( z \). With \( \int_1 := 1 \), (2.4) defines homotopy invariant functions of a path \( \gamma : [0, 1] \to \mathbb{C} \setminus \Sigma \) of integration which are often identified with the resulting multivalued analytic functions of the endpoint \( z = \gamma(1) \in \mathbb{C} \setminus \Sigma \).

Observe that in (2.2) we chose the singular base point \( \gamma(0) = 0 \in \Sigma \) which is the reason why we had to define \( L_{\varphi}(z) := \log z \) specially and not by the divergent integral \( \int_0^z 1/z dz \).

We extend the maps \( w \mapsto \int w \) and \( w \mapsto L_w \) linearly to the whole shuffle algebra \( T(\Sigma) \). Then the fundamental properties of iterated integrals become

1. \( \int_\gamma w \cdot \int_\gamma w' = \int_\gamma (w \shuffle w') \), i.e. \( \int_\gamma \) is multiplicative,
2. By Chen’s lemma, concatenation \( \gamma \star \eta \) of two paths with \( \eta(1) = \gamma(0) \) gives for every word \( w = \omega_\sigma w_n \)

\[
\int_{\gamma \star \eta} w = \sum_{i=0}^n \int_{\gamma} \omega_{\sigma i} \ldots \omega_{\sigma n} \cdot \int_{\eta} \omega_{\sigma i+1} \ldots \omega_{\sigma n}
\] (2.5)

3. \( \mathbb{Q}(z) \otimes_{\mathbb{Q}} T(\Sigma) \ni f \otimes w \mapsto \left[ \gamma \mapsto f(\gamma(1)) \cdot \int_\gamma w \right] \) is injective, so iterated integrals associated to different words are linearly independent with respect to rational (actually even for algebraic) prefactors \( f \).

The analogous properties hold for the hyperlogarithms \( w \mapsto L_w \) of (1.4). These functions \( L_w : \mathbb{C} \setminus \Sigma \to \mathbb{C} \) are single-valued once we restrict to the simply connected domain where \( 0 < |z| < \min \{|\sigma| : \sigma \notin \Sigma\} \) and \( z \notin (-\infty, 0) \), after fixing log to the principal branch with \( \log 1 = 0 \). In the sequel we will only consider such hyperlogarithms \( f(z) = L_w(z) \) that allow for an analytic continuation to all of \((0, \infty)\). This is necessary to give the integrals \( \int_0^\infty f(z) dz \) we want to compute a well-defined value.

2.2. Integration and differentiation

We consider the algebra \( L(\Sigma) := \mathcal{O}_\Sigma[ L_w(z) : w \in \Sigma^* ] \) spanned by hyperlogarithms with rational prefactors whose denominators factor linearly with zeros in \( \Sigma \) only:

\[
\mathcal{O}_\Sigma := \mathbb{Q}\left[ z, \frac{1}{z-\sigma} : \sigma \in \Sigma \right].
\] (2.6)
By construction we have $\partial_z L_{\omega_1 \ldots, \omega_r}(z) = \frac{1}{\sigma - \tau} L_{\omega_1 \ldots, \omega_r}(z)$ such that $L(\Sigma)$ is closed under $\partial_z$, while for any $f \in L(\Sigma)$ we can find primitives $F \in L^+(\Sigma)$, $\partial_z F(z) = f(z)$, in the enlarged algebra $L^+(\Sigma) := \mathcal{O}^+ \cup \{ L_w : w \in \Sigma^+ \}$ where

$$\mathcal{O}^+ := \mathcal{O}_\Sigma \cup \left\{ \frac{1}{\sigma - \tau} : \sigma, \tau \in \Sigma \text{ and } \sigma \neq \tau \right\}. \quad (2.7)$$

Namely, a primitive for $g(z)L_w(z)$ can be constructed by partial fractioning the rational prefactor

$$g(z) = \sum_{\sigma \in \Sigma} \sum_{n \in \mathbb{N}} \frac{A_{\sigma, n}}{(z - \sigma)^n} + \sum_{n \in \mathbb{N}_0} A_n z^n \in \mathcal{O}_\Sigma,$$  

setting $F = L_w(z)$ as a primitive of $\frac{L_w(z)}{z - \sigma}$ and repeated use of the partial integration formula

$$\int \frac{dz}{L_w(z)} \left( \frac{1}{z - \sigma} \right) = -\log \left( \frac{L_w(z)}{n(z - \sigma)} \right) + \int \frac{dz}{L_w(z)} \left( \frac{1}{n(z - \sigma)} \right),$$

$$\int dz z^n L_w(z) = \frac{z^{n+1}}{n+1} L_w(z) - \int dz z^n \frac{\partial_z L_w(z)}{n+1} \quad (2.9)$$

(2.10)

to reduce the problem of finding a primitive to the case where the hyperlogarithm $\partial_z L_{\omega_1 \ldots, \omega_r}(z) = \frac{L_{\omega_1 \ldots, \omega_r}(z)}{z - \sigma}$ is of lower weight. This recursion terminates when $w$ becomes the empty word. Hence computation of a convergent integral $\int_0^\infty f(z) \, dz$ for $f \in L(\Sigma)$ reduces to obtaining a primitive $F \in L^+(\Sigma)$ of $f$ as described and evaluating the limits

$$\int_0^\infty f(z) \, dz \to \lim_{z \to 0} F(z) - \lim_{z \to +\infty} F(z). \quad (2.11)$$

2.3. Divergences and logarithmic regularization

The singularities of $L_w(z)$ at $z \to \tau \in \Sigma \cup \{ \infty \}$ are at worst logarithmic, namely for any $w \in T(\Sigma)$ there is a decomposition

$$L_w(z) = \mathcal{O}_w \left( \begin{array}{c} z \\ \log z \\ \log(\tau - z) \end{array} \right), \quad \tau = \infty, \quad \tau \neq \infty \quad (2.12)$$

with functions $f^{(i)}_{w, \tau}(z)$ uniquely defined upon the requirement of being holomorphic at $z \to \tau$; for $t = \infty$ this means holomorphy of $f^{(0)}_{w, \tau}(z)$ at $z \to 0$. Note that $L_w(z)$ is finite for $z \to \tau \notin \{ 0, \infty \}$ whenever $w$ does not begin with the letter $\omega_r$.

The **regularized limits** are defined for any $\tau$ as

$$\text{Reg}_\tau L_w(z) := f^{(0)}_{w, \tau}(\tau), \quad (2.13)$$

such that $\lim_{z \to \tau} L_w(z) = \text{Reg}_\tau L_w(z)$ whenever this limit is finite. The advantage is then that by linearity,

$$\lim_{z \to \tau} f(z) = \sum_{w \in \Sigma^*} \lambda_w \text{Reg}_\tau L_w(z) \quad \text{for} \quad f(z) = \sum_{w \in \Sigma^*} \lambda_w L_w(z)$$

can be computed for each word $w$ separately and is thus well suited for an implementation, even though the limits $\lim_{z \to \tau} L_w(z)$ might diverge individually.

**Definition 2.1.** For disjoint sets $A, B \subset \Sigma$ the projection $\text{reg}^A_B : T(\Sigma) \to T(\Sigma)$ is determined by the requirements

1. $\text{reg}^A_B(w \cup w') = \text{reg}^A_B(w) \cup \text{reg}^B_B(w') \forall w, w' \in \Sigma^*$,  
2. $\text{reg}^A_B(w) = w = \omega_{\sigma_1} \ldots \omega_{\sigma_n}$ if $\sigma_1 \notin B$ and $\sigma_n \notin A$,  
3. $\text{reg}^A_B(w) = 0$ if $\forall 1 \neq w \in A^* \cup B^*$.

We write $\text{reg}^\tau_{\sigma} := \text{reg}^\tau_{\sigma} B$ and suppress empty sets in the notation, e.g. $\text{reg}_{\sigma} = \text{reg}^\tau_{\sigma_0}$ and $\text{reg}^\tau_{\sigma} = \text{reg}^\tau_{\sigma_0}$.

This shuffle-regularization is a combinatorial operation that projects onto words that neither begin with a letter in $A$ nor end with a letter from $B$. Every word $w \in T(\Sigma)$ decomposes uniquely as

$$w = \sum_{w \in A^*} \sum_{w \in B^*} a \cup b \cup w^{(a,b)} \quad (2.14)$$

into such $A$-$B$-regularized words $w^{(a,b)} \in \text{im} \text{reg}^B_A$ and thus $\text{reg}^B_A(w) = w^{(1,1)}$. To compute $\text{reg}^B_A$, we can use

**Lemma 2.2.** For $w = \omega_0 \alpha$ with $\alpha = \omega_1 \ldots \omega_{\alpha_n}$,

$$w = \sum_{i=0}^n [\omega_\alpha \ldots (-\omega_0)] \omega \alpha \omega_\alpha i+1 \ldots \omega_\alpha. \quad (2.15)$$

So when $\sigma \notin A$ is the last letter of $w$ not in $A$, thus $\sigma \in \Sigma^*$, we deduce $\text{reg}^A_B(w) = (-1)^n (\omega \omega_\alpha i+1 \ldots \omega_\alpha).$

Analogously $\text{reg}^B_B(b, \omega_\beta u) = \omega_{\beta} (u \omega S(b))$ for $b \in B^*$ and $\sigma \notin B$, setting $S(b_1 \ldots b_k) := (-b_1) \ldots (-b_1)$. Finally note $\text{reg}^B_A = \text{reg}^A \circ \text{reg}^B \circ \text{reg}^A_A$.

For $A = \{ 0 \}$ and $B = \emptyset$, $(2.14)$ reads $w = \sum_i \omega_i \cup w_i$ where $w_i$ do not end in $\omega_i$. Since $L_{w_i}(z)$ is holomorphic at $z \to 0$ and $L_{w_i}(z) = \sum \log^i z \cdot L_{w_i}(z)$ reveal $f_{w_i}(z) = L_{w_i}(z)$ from $(2.13)$, we can compute the limit

$$\text{Reg}_{z \to 0} L_{w_i}(z) = \text{Reg}_{z \to 0} L_{w_i}(0) = 0$$

In fact our definition $(2.12)$ is deliberately tuned such that the empty word $w = 1 \to L_w(z) = 1$ is the only word in $\Sigma^*$ with non-vanishing $\text{Reg}_{z \to 0} L_w(z)$.

**Lemma 2.3.** Let $w = \sum_i \omega_i \cup w_i \in \Sigma^*$ for $\text{reg}^0_\emptyset$, $w_i$ not ending on $\omega_0$. Then $L(w) = \sum \log^n z \cdot L_{w_i}(z)$ and $L_{w_i}(z)$ are holomorphic at $z \to 0$ and their series expansion $L_{w_i}(z) = \sum_{n \geq 0} a_n z^n$ can be directly computed (recursively) from the iterated integral representation: Starting with the empty word $L_1(z) = 1$, let $L_w(z) = \sum_{n \geq 0} a_n z^n$. Then

$$L_w(0) = \sum_{n=1}^{\infty} a_n \, z^n \quad \text{and for any} \quad \sigma \in \Sigma \setminus \{ 0 \}. \quad (2.17)$$

$$L_w(0) = \frac{1}{(\sigma - \tau)^n} \sum_{m=0}^{\infty} \frac{a_n}{\sigma^m (n + m + 1)} z^{n+m+1}. \quad (2.18)$$
Lemma 2.6.  

\[ L_w(z) = \sum_{k=0}^{n-1} \frac{\log (\omega_{1,k} \cdots \omega_{k,n})}{z} \cdot \frac{\log (\omega_{k+1,n+1} \cdots \omega_{n,n})}{z} \]

(2.19)

**Definition 2.4.** For a word \( w = \omega_1 \ldots \omega_n \), let

\[ \text{reg}^\infty(w) := \sum_{k=1}^{n} (\omega_{k,n} - \omega_{-1}) \left[ (-\omega_{-1})^{k-1} + \omega_{k,n+1} \cdots \omega_{n,n} \right] \]

(2.20)

denote the projection of \( T(\Sigma) \) on words beginning with differences \( (\omega_n - \omega_{-1}) \) that annihilates \( n \log(z) \) for any \( n > 0 \). Further set \( \text{reg}^\infty(z) := \text{reg}^\infty \circ \text{reg}^\infty_0 \).

If \( w = (\omega_{-1}, \omega_n) \), then

\[ L_w(z) = \int_{0}^{z} \frac{(1 + \sigma_1/z)}{z - \sigma_1} d\sigma \]

reveals that \( L_w(z) = \text{Reg}_z(z) \) is finite as an absolutely convergent integral since \( L_w(z) \) grows at worst logarithmically by (2.12). We therefore conclude

\[ L_w(z) = \text{Reg}_z(z, \infty) \](2.22)

from \( \text{Reg}_z(z) = 0 \) and \( \text{Reg}_z^\infty \).

**Lemma 2.5.** For any words \( w, w' \in L(\Sigma) \), \( \text{reg}^\infty(w \cup w') = \text{reg}^\infty(w) \cup \text{reg}^\infty(w') \) is multiplicative and for any word \( w = \omega_1 \ldots \omega_n \), the following identity holds:

\[ w = \sum_{k=0}^{n} \omega_{k,n} \cup \text{reg}^\infty(\omega_{1,k} \cdots \omega_{k,n}) \cdot \omega_{k+1,n+1} \cdots \omega_{n,n} \]

(2.23)

The second ingredient to compute (2.19) lies in

**Lemma 2.6.** For any Möbius transform \( f(z) = \frac{az + b}{cz + d} \),

\[ \frac{df^{-1}(z)}{f(z)} = \frac{dz}{z - f^{-1}(z)} - \frac{dz}{z - f^{-1}(\infty)} \]

where \( f(w) = \sum_{k=1}^{n} \Phi_k \cdot \Phi_{k+1} \cdot \Phi_{k+2} \cdot \ldots \cdot \Phi_n \) with the linear (and multiplicative) map \( \Phi_k \) that replaces any letter \( \omega_{1,k} \) by \( \Phi_k(\omega_{1,k}) := \omega_{1,k} \cdot \omega_{1,k+1} \cdot \omega_{1,k+2} \), dropping any \( \omega_{1,k} := 0 \).

(2.24)

We apply this to \( f(w) = f_{1/w} \Phi_1(\omega_{1,k}) w \) and recall that \( \lim_{u \to 1} f_{1/u} w = L_w(\frac{1}{u}) \) is finite for \( w \in \text{reg}^\infty_0 \) not ending in \( \omega_0 \). Furthermore,

\[ \text{Reg}_u \lim_{u \to 1} f_{1/u} w = \text{Reg} \log \frac{u}{z} = \log \frac{1}{z} = L_w(\frac{1}{z}) \]

completes (2.19) to a combinatorial algorithm of the form

\[ L_w(z) = \sum L_{\Phi_1(w)}(\frac{1}{z}) \cdot \text{Reg}^\infty_0(z) \]

(2.26)

Employing (2.18), this equation can be used to expand \( L_w(z) \) at \( z \to \infty \) as a polynomial in \( z \) and a power series in \( \frac{1}{z} \). This suffices to compute \( \lim_{z \to \infty} F(z) \) in (2.11).

2.4. Regularized limits as hyperlogarithms

When we follow (2.14), after taking the limits (2.11) we will from (2.26) have a representation of the partial integral \( F \) in terms of expressions \( L_{\text{reg}^\infty_0}(w)(\infty) \) that depend on the next integration variable \( t := \frac{1}{2} \cdot \text{lim}^\infty \) through the letters in the word \( w \). To proceed with the integration process, we must rewrite \( F \) as a hyperlogarithm in \( t \).

So let \( w = \omega_1 \ldots \omega_n \) with letters \( \sigma_i(t) \) depending on a parameter \( t \), then we can take the derivative \( \partial_t L_w(z) \) in the integrand of the iterated integral \( L_w \). Partial fractioning and partial integration suffice to prove

\[ \partial_t L_w(\sigma_i) = \sum_{k=1}^{n} \left[ \frac{\partial_t \sigma_i}{\sigma_i} \right] L_{\omega_{i,k+1} \cdots \omega_{i,n}}(\zeta) \]

(2.27)

where \( \mu_{\sigma_i} \) means to delete the letter \( \omega_{i} \) from \( w \). Applying \( \text{Reg}_z \) and exploiting \( \text{Reg}_z \to \infty \) yields

\[ \partial_t \text{Reg}_z(z) = \sum_{i=1}^{n} \left[ \partial_t \text{Reg}_z(z) \cdot L_{\omega_{i,k+1} \cdots \omega_{i,n}}(\zeta) \right] \]

(2.28)

We assume that all \( \sigma_i = 1 \) are rational, such that any \( \sigma_i - \sigma_j = \frac{c}{\tau - \zeta} \).

(2.29)

Namely, equation (2.27) presents an effective recursive algorithm to compute (2.28). First translate all shorter words \( \text{Reg}_z \to \infty \) \( L_{\omega_{i,k+1} \cdots \omega_{i,n}}(\zeta) \) such (2.27) reads

\[ \partial_t \text{Reg}_z(z) = \sum_{u \in \Sigma^*_{\tau} \cdots \partial_t} \lambda_{\tau,u} L_u(t) \cdot c_u \]

(2.30)

for some multiplicities \( \lambda_{\tau,u} \in \mathbb{Z} \) and constants \( c_u \). So

\[ \text{Reg}_z(z) = C + \sum_{u \in \Sigma^*_{\tau} \cdots \partial_t} \lambda_{\tau,u} L_u(t) \cdot c_u \]

(2.31)

determined up to the integration constant

\[ C = \text{Reg} \text{Reg}_z(z) \]

(2.32)

\[ ^2 \text{with zeros } \tau \in \mathbb{W} \text{ in the algebraic closure and integer multiplicities } \lambda \in \mathbb{Z} \]
2.5. Regularized limits of regularized limits at infinity

We now explain how to compute the regularized limits (2.32) symbolically, without a need for numeric evaluations (which are for example used in [17] as explained in its appendix D). The ideas we present in the first half of this section were very recently also sketched in [16].

Note that the limits $C$ of (2.32) are constant only with respect to the variable $t$, while in our applications these will in general still depend on further variables.

Let $w \in \Sigma^*$ be a word with letters $\Sigma \subset Q(t) \setminus \{0, \infty\}$ depending rationally on $t$. We can restrict to $w = \text{reg}_0(w)$ not ending with $\omega_0$ since Reg$_{z \to \infty} L_{\omega_0}(z) = 0$. The simplest possible case is

**Lemma 2.8.** When $\lim_{t \to 0}(\Sigma) \subset Q \setminus \{0, \infty\}$ is finite and $w = \omega_1 \ldots \omega_n$ ends in a letter with $\lim_{t \to 0}(\sigma_0) \neq 0$, then

$$\text{Reg } L_{\omega}(z) = \text{Reg } L_{\text{lim } w}(z).$$  

(2.33)

**Proof.** Using (2.22) it suffices to investigate differences $w = (\omega_1 - \omega_0) \omega_2 \ldots \omega_n$ and consider $L_{\omega}(\infty)$ as the absolutely convergent integral (2.21) with integrand

$$\frac{1 + \sigma_1}{(z_1 + 1)(z_1 - \sigma_1(t))} \frac{1}{2z_1 - \sigma_2(t)} \ldots \frac{1}{z_n - \sigma_n(t)}.$$  

(2.34)

We can apply the theorem of dominated convergence to show $\lim_{t \to 0} L_{\omega}(\infty) = L_{\lim w}(\infty)$, essentially because the limiting integral is absolutely integrable itself. So

$$\text{Reg } L_{\omega}(z) = \text{Reg } L_{\lim w}(\infty) = \lim_{t \to 0} \text{Reg } L_{\omega}(\infty)$$

holds for all $w$ as allowed in (2.8). Looking at (2.20), we may swap $\text{reg } \infty$ and $\lim_{t \to 0}$ since the latter just substitutes letters $\omega_i \mapsto \omega_{i(0)}$. Finally apply (2.22) again. □

This naive method can fail for three different reasons:

I Some $\sigma(t) \in \Sigma$ diverges in the limit $t \to 0$.

II $\sigma_0(0) = 0$, because the limiting integrand (2.34) at $t = 0$ is not integrable: $0 \frac{1}{z_n - \sigma_n(t)}$ diverges.

III Some letter has a limit $\sigma_i(0) \neq 0$ on the positive real axis, wherefore (2.34) acquires a singularity at $z = \sigma_i(0)$ inside the domain of integration.

We consider our main contribution as the algorithm to deal with these cases, which we present below. First note

**Lemma 2.9** (Scaling invariance). Given some $\alpha \in \mathbb{Z}$ and a word $w = \omega_1 \ldots \omega_n$ in letters $\sigma_i \in Q(t)$, let $w' := \omega_1' \ldots \omega_n'$ for $\sigma_i'(t) := t^\alpha \cdot \sigma_i(t)$. Then

$$\text{Reg } L_{\omega}(z) = \text{Reg } L_{w'}(z).$$  

(2.35)

**Proof.** Since $\text{Reg } L_{\omega}(z) = \text{Reg } L_{\omega}(w)$ we can restrict to $\sigma_0 \neq 0$ and rescale $z' = z^{t/\alpha}$ in (1.12) to conclude that $L_{\omega}(z) = L_{\omega}(zt^\alpha)$. In regard of the regularization (2.12) at $\tau = \infty$, this shows that

$$f^{(j)}_{w,\infty}(z) = \sum_{j \geq 1} \binom{j}{i} \log^{j-i}(t^\alpha) \cdot f^{(i)}_{w,\infty}(zt^\alpha).$$

Therefore we can conclude that indeed,

$$\text{Reg } f^{(j)}_{w,\infty}(\infty) = \text{Reg } \sum_{i = 0}^{j} \frac{\log^{j-i}(t^\alpha)}{i!} \cdot f^{(i)}_{w,\infty}(\infty)$$

$$= \text{Reg } f^{(j)}_{w',\infty}(\infty) = \text{Reg } \text{Reg } L_{w'}(z).$$

A suitable such rescaling ensures finiteness of all $\sigma_i'(t)$ at $t \to 0$ and thus resolves problem I above.

**Example 2.10.** Applying lemmata 2.2, 2.8, equation (2.22) and a Möbius transformation (2.6) with $f(z) = \frac{1}{t+z}$ shows

$$\text{Reg } L_{\omega;w-1}(z) = \text{Reg } L_{\omega;w-1}(z)$$

$$= \lim_{z \to \infty} \text{Reg } L_{\omega;w-1}(z) = \text{Reg } L_{\omega;w-1}(\infty) = -\int_0^1 \omega_0 \omega_1 = \zeta_2.$$

To address the issue II when $\sigma_0(0) = 0$, we make

**Definition 2.11.** For any $0 \neq \sigma(t) \in Q(t)$, the Laurent series $\sigma(t) = \sum_{n=0}^\infty t^na_n$ at $t \to 0$ with $a_N \neq 0$ defines a vanishing degree $\text{deg}_v(\sigma) := N \in \mathbb{Z}$ and leading coefficient $\text{lead}(\sigma) := a_N = \lim_{t \to 0}(\sigma(t) \cdot t^{-N})$. For a word $w = \omega_1 \ldots \omega_n$ set $\text{deg}_v(w) := \min\{\text{deg}_v(\sigma_i) : i \leq n\}$.

Whenever the final letter of a word $w = \omega_1 \ldots \omega_n$ is of smallest vanishing degree $\sigma_i = \text{deg}_v(w)$, rescaling $\sigma_i(t) := t^{-\text{deg}_v(w)} \sigma_i(t)$ ensures $\sigma_i(0) = \text{lead}(\sigma_i) \neq 0$ and makes Lemma 2.8 becomes applicable. Hence let us define

$$\text{reg } w := \lim_{t \to 0} \omega_{i_1} \ldots \omega_{i_n} \text{ if } \sigma_i = \text{deg}_v(w),$$

(2.36)

e.g. $\text{reg } \omega_{1-\omega_{-1}}(t) = \omega_{1-\omega_{-1}}$ in example 2.10.

But if $\text{deg}_v(\sigma_i) > \text{deg}_v(w)$, the rescaled $\sigma_i(t)$ will vanish at $t \to 0$. In this case let

$$k := \max\{i : \deg_v(\sigma_i) = \deg_v(w)\} < n$$

denote the last place in $w$ with minimal vanishing degree. Using (2.15) we can rewrite $w = \sum w_i \omega_i a_i$ such that each $w_i$ ends in $\omega_\alpha$ and $a_i$ is a suffix of $\omega_{i+k} \ldots \omega_n$. Thus $|a_i| \leq n - k < n$. Applying this procedure recursively to each $a_i$ finally results in a representation

$$w = \sum_{i} (w_{i,1} \omega_1 \ldots \omega_{i,r_i})$$

(2.37)

of $w$ in the shuffle algebra into elements $w_{i,j}$ each ending in some $\sigma_{i,j}$ with minimum vanishing degree $\text{deg}_v(\sigma_{i,j}) = \text{deg}_v(w_{i,j}).$

**Example 2.12.** For $w = \omega_{-1} \omega_{-1}$ this decomposition reads $w = \omega_{-1} \omega_{-1} - \omega_{-1} \omega_{-1}$. So with $\text{Reg } L_{\omega_{-1} \omega_{-1}}(z) = 0$,

$$\text{Reg } L_{\omega_{-1} \omega_{-1}}(z) = -\text{Reg } L_{\omega_{0} \omega_{-1}}(z).$$

**Definition 2.13.** For any alphabet $\Sigma \subset Q(t)$ let

$$\text{lead}_{\Sigma}(\sigma) := \{0\} \cup \{\text{lead}(\sigma) : \sigma \in \Sigma \setminus \{0\}\}.$$  

(2.38)

Further we denote by $\text{reg}_{\text{lead}}(\Sigma) : T(\Sigma) \to T(\text{lead}_{\Sigma}(\sigma))$ the unique morphism of shuffle algebras that extends (2.36) with $\text{reg}_{\text{lead}}(\omega_0) = 0$. 


This combinatorial regularized limit of words is a projection.

\[
\text{reg } w = \sum_{t \to 0} \left( \text{reg } w_{\ell,t} \cup \ldots \cup \text{reg } w_{0,t} \right).
\]

(2.39)

Putting together the lemmata and with the linearity and multiplicativity of \(\text{Reg}_{t \to 0}\), \(\text{Reg}_{z \to \infty}\), and \(w \mapsto L_w(z)\) we can compute regularized limits combinatorially using

**Corollary 2.14.** For an alphabet \(\Sigma \subset \mathbb{C}(t) \setminus (0, \infty)\) such that \(\text{lead}_t(\Sigma) \subset \mathbb{C} \setminus (0, \infty)\), any \(w \in T(\Sigma)\) fulfills

\[
\text{Reg } L_w(z) = \text{Reg } L_{\text{reg}_{t \to 0}(w)}(z).
\]

(2.40)

**Example 2.15.** For \(a, b, c \in \mathbb{C} \setminus [0, \infty)\), the decomposition of the form \((2.37)\) for the word \(w = \omega_{i,t} \omega_{j,t} \omega_{k,t}\) is

\[
w = \omega_{a,t} + \omega_{b,t} \omega_{c,t} - \omega_{d,t} + \omega_{e,t} \omega_{f,t} \omega_{g,t} + \omega_{h,t} \omega_{i,t} \omega_{j,t}
\]

and shows \(\text{Reg}_{t \to 0} \text{Reg}_{z \to \infty} L_w(z) = \text{Reg}_{z \to \infty} L_w(z)\) for

\[
u = \omega_{a,t} \omega_{b,t} \omega_{c,t} - \omega_{d,t} \omega_{e,t} \omega_{f,t} \omega_{g,t} + \omega_{h,t} \omega_{i,t} \omega_{j,t}.
\]

We still need to address problem III on our agenda: What happens when some \(\text{lead}_t(\sigma) \subset (0, \infty)\) approaches a positive value? In this situation, \(\text{Reg}_{z \to \infty} \text{Reg}_{t \to 0} L_w(z)\) can have a discontinuity along some \(t \in (0, \infty)\). For example,

\[
\text{Reg } L_w(z) = \text{Reg } L_{\text{reg}_{z \to \infty}(w)}(z).
\]

(2.41)

\[\text{Reg } L_w(z) = \text{Reg } L_{\text{reg}_{t \to 0}(w)}(z).
\]

(2.42)

\[\text{Reg } L_w(z) = \text{Reg } L_{\text{reg}_{z \to \infty}(w)}(z).
\]

(2.43)

**Definition 2.16.** Choosing \(t \in \mathbb{R}^+\) partitions the alphabet

\[
\Sigma = \Sigma^+ \cup \Sigma^0 \cup \Sigma^- \subset \mathbb{C}(t) \setminus (0, \infty)
\]

(2.44)

\[\text{into non-positive } \text{lead}_t(\Sigma) \cap (0, \infty) = \emptyset \text{ and letters with } \text{lead}_t(\Sigma^\pm) \subset \mathbb{R}^+ \text{ for sufficiently small } \text{Re } t \text{ and infinitesimal } \text{Im } t > 0. \text{ In particular we note that whenever } \text{lead}_t(\sigma) \subset (0, \infty),
\]

\[\text{deg}_t(\sigma) < 0 \Rightarrow \sigma \in \Sigma^- \text{ and } \text{deg}_t(\sigma) > 0 \Rightarrow \sigma \in \Sigma^+.
\]

(2.45)

We denote the finite positive limits by

\[\Sigma_0^+ := \left\{ \lim_{t \to 0} \sigma(t) : \sigma \in \Sigma^+ \text{ and } \text{deg}_t(\sigma) = 0 \right\}.
\]

(2.46)

**Example 2.17.** For \(\Sigma = \{-1 + it, t, 3t, 1 + it, 2t - t, -1 + it, 1 + it, 2t, -1 + it\}\), the limit \(t \to 0 + it\) is shown in figure 2 and (2.41) reads

\[
\Sigma = \{-1 + it, 0 - t\}, \Sigma^+ = \{1 + it, 3t\}, \Sigma^- = \left\{ 2 - t, -1 + it \right\}.
\]

(2.47)

Figure 1: This graph shows the limits of \(\Sigma\) in example 2.17 when \(t \to 0\) with positive real part and small positive imaginary part.

Figure 2: The letters \(\{1 + it, 2 - t\} \subset \Sigma\) in example 2.17 induce a deformation of the real integration path \([0, \infty)\) towards \(\gamma\), which avoids the positive limits in passing below \(\Sigma_0^+ = \{1\}\) and above \(\Sigma_0^- = \{2\}\).

Now consider a word \(w = \omega_{i,t} \ldots \omega_{a,t}\) with \(\text{deg}_t(\sigma_n) = \text{deg}_t(w)\). Only the letters \(\Sigma_{i,t} := \{\sigma_r: \text{deg}_t(\sigma_r) = \text{deg}_t(w)\}\) play a role since after rescaling \(\sigma_n'(t) = \sigma_n(t) \cdot t^{-\text{deg}_t(w)}\) by lemma 2.9 all other letters have \(\text{deg}_t'(\sigma_r') = \text{deg}_t(\sigma_r) - \text{deg}_t(w) > 0\) and therefore approach \(0 \notin (0, \infty)\) in the limit \(t \to 0\) of \((2.33)\). By homotopy invariance,

\[
\text{Reg } L_w(z) = \int_{\gamma} \text{reg } w(t) \gamma(t) \to 0
\]

is the iterated integral along a smooth deformation \(\gamma\) of the originally real integration contour \([0, \infty)\). It avoids the positive letters among \(\text{lead}_t(\Sigma)\) as follows:

\[\text{deg}_t(w) < 0: \Sigma_w \setminus \Sigma^- \subset \Sigma^-, \gamma \text{ passes above all } \text{lead}_t(\Sigma_w),
\]

\[\text{deg}_t(w) > 0: \Sigma_w \setminus \Sigma^- \subset \Sigma^+, \gamma \text{ passes below all } \text{lead}_t(\Sigma_w),
\]

\[\text{deg}_t(w) = 0: \lim_{t \to 0} (\text{reg } \Sigma_w \setminus \Sigma^-) \subset \Sigma_0^+ \cup \Sigma_0^\pm \text{ and } \gamma \text{ passes above } \Sigma^- \text{ and below } \Sigma_0^+ \text{ as illustrated in figure 2 for example 2.17. We must require that } \Sigma_0^+ \cap \Sigma_0^- = \emptyset \text{ as otherwise } \gamma \text{ is pinched between letters from } \Sigma^+ \text{ and } \Sigma^- \text{ in the limit } t \to 0. \text{ This situation did not occur in our applications but could be incorporated in the future.}
\]

In order to keep the implementation simple, we express (2.44) again in the form \(\text{Reg}_{z \to \infty} L_w(z)\) with \(v \not= \text{containing any positive letters (such that } L_w(z)\text{ is single-valued on } z \in (0, \infty)\) and does not need additional specification of the contour \(\gamma\)). This is achieved by splitting up the contour \(\gamma = \eta_u \star \gamma_u\) at \(u > 0\) with the straight path \(\eta_u\) from 0 to \(u\).
So for $w = \omega_{\sigma_1} \ldots \omega_{\sigma_n}$ with $\sigma_n \neq 0$, Chen’s lemma (2.5) takes the form

$$I_w = \sum_{k=0}^{n} \int_{\gamma} \omega_{\sigma_1} \ldots \omega_{\sigma_k} \cdot L_{\sigma_{k+1}, \ldots, \sigma_n}(u)$$

(2.45)

where $u < \tau := \min(\{\sigma_1, \ldots, \sigma_n\} \cap (0, \infty))$ approaches the first potential branch point of $L_w(z)$ on the positive real axis, see figure 3. In the limit $u \to \tau$ we obtain

$$L_w(z) = \sum_{k=0}^{n} \int_{\gamma} \omega_{\sigma_1} \ldots \omega_{\sigma_k} \cdot L_{\sigma_{k+1}, \ldots, \sigma_n}(\tau)$$

(2.46)

where $\int_{\gamma}$ denotes the iterated integral (2.4) whenever $\sigma_k \neq \tau$. It is extended to all words by imposition of $\int_{\gamma} \omega(wv) = \int_{\gamma} \omega(w) \cdot \int_{\gamma} \omega(v)$ and letting $\gamma$ determine the branch of

$$\int_{\gamma} \omega(\tau) := \int_{\gamma} \omega = \pm i\pi + \log \frac{z - \tau}{\tau} \quad \text{for} \quad z > \tau.$$  

(2.47)

Example 2.19. Take the dilogarithm $L_{\omega \omega_1}(z) = -L_i(z)$ and the path $\gamma$ passing below $\tau = 1$ shown in figure 3. Since

$$\text{reg}_1(\omega_1) = 0 \quad \text{and} \quad L_{\omega \omega_1}(1) = -\zeta_2$$

(2.46) reduces to

$$L_i(z) - \zeta_2 = -\int_{0}^{z} (\omega \omega_1) := -\int_{0}^{z} \omega_1 \cdot \int_{0}^{z} \omega_1 + \int_{0}^{z} (\omega_1 \omega_0).$$

Inserting $f^1_{\omega_1}$ we obtain $\zeta_2 = \int_{0}^{z} \omega_1 = i\pi + \log(z - 1)$ and $z = 1 + \Im(z)$, we obtain for $\Im(z) = \Im(z') > 1$ the representation

$$L_i(z) = \zeta_2 + L_{\omega \omega_1}(z') - (i\pi + \log(z')) L_{\omega \omega_1}(z') = \z_2 - L_i(z') - (i\pi + \log(z')) \log.$$

In (2.46) we can resolve $\int_{\gamma}$ into explicit factors $\pm i\pi$ from (2.47) as dictated by $\gamma$ and the iterated integrals $\int_{\gamma}$.

Recursive splitting of these at the next positive left $\tau' := \min(\{\sigma_1, \ldots, \sigma_n\} \cap (0, \infty))$ finally expresses

$$\int_{\gamma}(w) \quad \text{in powers of} \quad \pm i\pi \quad \text{and} \quad \int_{\gamma}(v), \int_{\gamma}(v'), \ldots$$

which are simply defined by the straight line integration paths $0 \to \tau, \tau \to \tau'$ and so on. Through Möbius transformations (2.25) these may all be transformed to $0 \to \infty$. 

3. The implementation HyperInt

3.1. General remarks

We implemented the algorithms of section 2 in the computer algebra system Maple [22]. Even though these procedures are very flexible, we did not intend to provide a general purpose package supporing arbitrary symbolic calculations with hyper- and polylogarithms.

Instead, we were driven by our aim to compute Feynman integrals as we comment on in section 3. Therefore other applications are not as well supported, but we will give examples showing how HyperInt can be used for quite general calculations with polylogarithms.

Note that we did not include facilities for numeric evaluations of hyper- and polylogarithms, because first of all these are not necessary for the algorithms and secondly there are already established programs available for this task, e.g. [32, 33].

The program uses the remember option of Maple, which creates lookup tables to avoid recomputation of functions. But some of these functions depend on global parameters as explained for instance in section 3.6. Therefore, whenever such a parameter is changed, the function forgetAll() must be called to invalidate those lookup tables. Otherwise the program might behave inconsistently.

3.2. Installation and files

The program requires no installation. It is enough to load it during a Maple-session by invoking

```maple
> read "HyperInt.mpl";
```

if the file HyperInt.mpl is located in the current directory or another place in the search paths of Maple. If periodLookups.m can be found, it will be loaded automatically which is of great benefit as explained in section 3.4.

All together, we supply the following main files:

HyperInt.mpl

Contains our implementation of the algorithms in section 2 as well as supplementary procedures to handle Feynman graphs and Feynman integrals.

periodLookups.m

This table stores a reduction of multiple zeta values up to weight 12 to a (conjectured) basis and similarly for alternating Euler sums up to weight 8. It is not required to run the program, but necessary for efficient calculations involving high weights. Details follow in section 3.4.

Manual.mw

This Maple worksheet explains the practical usage of HyperInt. In particular it includes plenty of explicit Feynman integral computations. Many explanations, details and comments are provided here.

HyperTests.mpl

A series of various tests of the program. Calling
Maple with maple HyperTests.mpl has to run without any error messages. Please report immediately if errors occur. Note that these tests only work when periodLookups.m can be found by HyperInt.

Due to the many different tests, including some Feynman integrals, the reader might find it instructive to read this file. See also section Appendix A.

3.3. Representation of polylogarithms and conversions

Internally, polylogarithms are represented as lists
\[ f = [[g_1, [w_{1,1}, \ldots, w_{1,r_1}]],[g_2, [w_{2,1}, \ldots, w_{2,r_2}]], \ldots] \] (3.1)

of pairs of rational prefactors \( g_i \) and lists of words \( w_{i,j} = \text{reg}_0(w_{i,j}) \) not ending on \( \omega_0 \). These encode the function
\[ f = \sum_i g_i \cdot \prod_{j=1}^{r_i} L_{\text{reg}^\infty(w_{i,j})}(\infty) \] (3.2)

and we decided not to use \( \text{2.2} \) to combine those words into the linear combination \( \omega_j w_{i,j} \) for two reasons:

1. Empirically, this expansion of shuffle products tends to increase the number of terms considerably.
2. Our algorithm of section 2.5 to compute \( \text{Reg}^\infty \) produces products of words with different sets of letters. Mixing these letters due to a shuffle introduces spurious letters in following integration steps which we want to avoid.

To encode hyperlogarithms of a particular variable \( z \), we use a list notation without explicit products:
\[ f = [[g_1, w_1],[g_2, w_2], \ldots] := \sum_i g_i(z)\text{L}_{w_i}(z). \] (3.3)

These representations make the implementation of the algorithms of section 2 straightforward, but for easier, human-readable input and output we allow the notations

\[ \text{Hlog}(z, [\sigma_1, \ldots, \sigma_j]) := L_{\sigma_1, \ldots, \sigma_j}(z) \quad \text{and} \quad \text{Mpl}([n_1, \ldots, n_r, [z_1, \ldots, z_r]]) := L_{n_1, \ldots, n_r}(z_1, \ldots, z_r) \]

for hyperlogarithms (1.2) and multiple polylogarithms (1.1). HyperInt extends the native function convert(f,form) to transform an expression \( f \) containing any of the functions

\{log, ln, polylog, dilog, Hlog, Mpl, HpI\}

into one of the possible target formats

form = HlogRegInf;
form \in \{Hlog, Mpl\};
form = HpI;

transforms \( f \) into the list representation (3.1).
expresses \( f \) in terms of \( L \) or \( Li \), using (1.3).
translates hyperlogarithms \( \text{Hlog}(z, w) \) with words \( w \in \{-1, 0, 1\}^r \) into the compressed notation of harmonic polylogarithms that was introduced in [6]. Concretely, \( \text{HpI}_{n_1, \ldots, n_r}(z) := L_{n_1, \ldots, n_r}(z) \) where \( 0 := \omega_0 \) and for any \( n \in \mathbb{N}, \pm n := \mp \omega_0^{-1} \omega_{\pm 1} \).

form = i

same as form = Hlog, but produces the notation
\[ i[0, \sigma_n, \ldots, \sigma_1, z] := \text{Hlog}(z, [\sigma_1, \ldots, \sigma_n]) \]

which is used in zeta_procedures [3]. The result can then be evaluated numerically in that program, e.g. using evalz(\( \cdot \)).

Example 3.1. The dilogarithm \( \text{Li}_2(z) \) has representations
\[
> \text{convert(polylog}(2,z), \text{Hlog});
\]
\[ \text{Hlog}(1, [0, 1/z]) \]
\[
> \text{convert(polylog}(2,z), \text{HlogRegInf});
\]
\[ ([1, [-1 + z, -1]], [-1, [[-1, -1, -1]]) \]

Due to the many functional relations, a general polylogarithm \( f(z) \) has many different representations. In particular, the representation (3.2) is far from being unique.

It is therefore crucial to be able to express polylogarithms in a basis in order to simplify results and to detect relations. As was demonstrated in [11], lemma 2.7 provides such a basis through

Corollary 3.2. Let \( f(z) = \text{Reg}_{z \to \infty} L_w(z) \) for \( w \in L(\Sigma) \) with rational letters \( \Sigma \subset \mathbb{Q}(z_{i+1}, \ldots, z_n) \), which may only depend on the following variables. The factors \( c_i \) in (3.4) are constants (with respect to \( z \)), namely
\[
c_i \in \text{Reg} \ldots \text{Reg} \text{Reg} L(\Sigma)(z).
\]

Its implementation constitutes the essential function

fibrationBasis(f, [z_1, \ldots, z_r], F),

which writes a polylogarithm \( f \) (preferably in the list notation (3.1), otherwise it will be converted first) in the form (3.4) with respect to the order \( \vec{z} = (z_1, \ldots, z_n) \) of variables (when \( \vec{z} \) is omitted, \( \vec{z} = [] \) is used). If the optional table \( F \) is supplied, the result will be stored as \( F_{w_{i,1}, \ldots, w_{i,n}} = c_i \).

Example 3.3. This function can be used to obtain functional relations between polylogarithms. For example,
\[
> \text{fibrationBasis(polylog}(2,1-z), [z]);
\]
\[
\text{Hlog}(z, [1, 0]) + \zeta_2
\]
\[
- \text{Mpl}(2, [z]) + \ln(z) \text{Mpl}(1, [z]) + \zeta_2
\]

reproduces the classic identity \( \text{Li}_2(1-z) = \zeta_2 - \text{Li}_2(z) - \log \log(1-z) \). Similarly, we obtain the inversion relation for \( \text{Li}_5 \): \( (-1) \frac{1}{120} \ln^5 x + \frac{1}{6} \ln^5 x + \frac{7}{180} \ln^3 x \ln x + \text{Li}_5(x) - x:\)
fibrationBasis(polylog(5, -1/x), [x]):
> convert(%, Mpl);
> fibrationBasis(%, [x, y]);
Hlog(y, [0, 1]) + Hlog(x, [0, 1]) − Hlog(x, [1]) Hlog(y, [1])

Note that for more than one variable, each choice of order defines a different basis and a function may take a much simpler form in one basis than in another. For example, Li_{1,2}(y, x) + Li_{2,1}(\frac{y}{x}, y)

f := Mpl([1,2], [y,x]) + Mpl([1,2], [1/y,y*x]):
fibrationBasis(polylog(5, -1/x), [x]):
> fibrationBasis(Mpl([3], [1/2]));

fibrationBasis(f, z);
Hlog(x, [0, 1, 1/y]) + Hlog([0, 1, 1/y])

We like to emphasize that every order describes a true basis without relations. In particular this means that f = 0 if and only if fibrationBasis(f, z) returns 0, no matter which order z was chosen.

Analytic continuation in a variable z is performed along a straight path, therefore the result can be ambiguous when this line contains a point where the function is not analytic. In this case, an auxiliary variable
\[ δ_z = \begin{cases} +1 & \text{when } z \in \mathbb{H}^+, \\ -1 & \text{when } z \in \mathbb{H}^- \end{cases} \] (3.6)

will appear to distinguish the branches above and below the real axis. From example 2.19 consider

\[ fibrationBasis(polylog(2, i+z), [z]); Iπδ_z Hlog(z, [-1]) − Hlog(z, [-1, 0]) + ζ_2 \]

3.4. Periods

Our algorithms express constants like (3.5) through iterated integrals \( R_{Q0→\infty} L_u(z) \) of words \( u \in \mathbb{Q}^\infty \) with algebraic letters. These are transformed into iterated integrals \( L_u(1) \) by \( u = \text{zeroInfPeriod}(w) \). Such special values of multiple polylogarithms satisfy a huge number of relations and it is clearly highly desirable to express them in a basis over \( \mathbb{Q} \).

The case \( u \in \{0, 1\}^\infty \) of multiple zeta values (MZV) is by now perfectly understood on the motivic level [35], such that conjectural \( \mathbb{Q} \)-bases are available at arbitrary weight and [36] even provides a reduction algorithm that was implemented in [34]. Similar results can also be found for some cases of \( u \in \{0, \mu: \mu^N = 1\}^\infty \) with \( N \)-th roots of unity \( \mu \), see [37].

HyperInt can load lookup tables to benefit from such relations and we supply the file periodLookups.m which provides the reductions that were proven in the data mine project [38] using standard relations. It includes multiple zeta values up to weight 12 and alternating Euler sums \( (u \in \{-1, 0, 1\}^\infty) \) up to weight 8 in the notionation
\[ ζ_{n_1, ..., n_r} := Li_{|n_1| - ... - |n_r|} \left( \frac{n_1}{|n_1|} ... \frac{n_r}{|n_r|} \right), \] (3.7)

with indices \( n_1, ..., n_r \in \mathbb{Z} \setminus \{0\}, n_r \neq 1 \). When \( u \in \{0, a, 2a\}^\infty \cup \{-a, 0, a\}^\infty \), Möbius transformations are used to express \( L_u(1) \) in terms of alternating Euler sums and \( \log a \).

Example 3.4. HyperInt automatically attempts to load periodLookups.m, but can run without it. With its help,

\[ \text{fibrationBasis}(\text{Mpl}([3], [1/2])); \]
\[ \frac{1}{6} \ln(2)^3 − \frac{1}{2} \ln(2)ζ_2 + \frac{7}{10} ζ_3 \]

is reduced to MZV and \( \ln 2 \). But if periodLookups.m is not available, we obtain merely

\[ \text{fibrationBasis}(\text{Mpl}([3], [1/2])); \]
\[ −ζ_3 − ζ_{2,1} − ζ_{1,2} + \frac{1}{6} \ln(2)^3 \]

The user can define a different basis reduction or provide bases for periods involving higher weights or additional letters. These must be defined as a table, zeroOnePeriods[u] := L_u(1), (3.8)

and saved to a file \( f \). To read it call loadPeriods(f).

Example 3.5. Polylogarithms \( Li_2(z) \) at fourth roots of unity \( z \in \{±1, ±i\}^\infty \) up to weight \( |n| \leq 2 \), like

\[ f := \text{Mpl}([1,1],[I,-1]) \text{Mpl}([1,1],[1,-1,1]): \]

\[ \text{fibrationBasis}(f); \]
\[ \text{Hlog}(1, [-I], I) + \text{Hlog}(1, [-I, I]) \]

are tabulated in periodLookups4thRoots.mpl in terms of \( \ln 2, i, π \) and Catalan’s constant \( \text{Im Li}_2(i) \):

\[ \text{loadPeriods("periodLookups4thRoots.mpl");} \]
\[ \text{fibrationBasis}(f); \]
\[ \frac{1}{8} ζ_2 + \frac{1}{4} \ln(2)^2 − \frac{1}{4} π \ln(2) + I \text{ Catalan} \]

\[ 3 \text{For MZV and alternating sums, [38] provides reductions up to weights 22 and 12, respectively.} \]
3.5. Integration of hyperlogarithms

The most important function provided by HyperInt is

\[ \text{integrationStep}(f, z) := \int_{0}^{\infty} f(z) \, dz \]  

(3.9)

and computes the integral of a polylogarithm \( f \), which must be supplied in the form (3.1). First it explicitly rewrites \( f(z) \in L(\Sigma)(z) \) following lemma 2.7 as a hyperlogarithm in \( z \). Then a primitive \( F = \text{integrate}(f, z) \) is constructed as explained in section 2.2 and finally expanded at the boundaries \( z \to 0, \infty \).

**Example 3.6.** To compute \( \int_{0}^{\infty} \frac{1}{z(y + f(y))} \, dy \), type

\begin{verbatim}
> convert(Mpl([1,1], [-x/y, -y]))/y/(y+1), HyperRegInt): fibrationBasis(%);
\end{verbatim}

\( \zeta_3 \log(x) + \log(x, [1, 0, 1]) - \log(x, [0, 0, 1]) \)

A more convenient and flexible form is the function

\[ \text{hyperInt}(f, [z_1 = a_1, \ldots, z_r = a_r : b_r]) \]

\[ := \int_{a_1}^{b_1} \cdots \int_{a_r}^{b_r} f(z_1, \ldots, z_r) \, dz_1 \cdots dz_r \]  

(3.10)

which computes multi-dimensional integrals by repeated application of (3.3) in the order \( z_1, \ldots, z_r \) as specified. It automatically transforms the domains \( (a_k, b_k) \) of integration to \( (0, \infty) \) and furthermore, \( f \) can be given in any form that is understood by \text{convert} (\cdot, \text{HyperRegInt}).

**Example 3.7.** A typical integral studied in the origin [10] of the algorithm is \( I_2 \) of equation (8.6) therein:

\[ I_2 := \int \frac{1}{t_2}(\frac{1}{t_2} - t_3) \, dt_2 \]

\[ \text{hyperInt}(I_2, [t_2=0 \ldots t_2, t_2=0 \ldots t_3, t_3=0 \ldots 1]): \]

\[ \text{fibrationBasis}(%) \]

\[ 2\zeta_3 \]

**Example 3.8.** The “Ising-class” integrals \( E_n \) were defined in [39]: For \( n \geq 2 \) let \( u_k := \sum_{i=1}^{k} t_i, u_1 := 1 \) and set

\[ E_n := 2 \int_{0}^{1} dt_2 \cdots \int_{0}^{1} dt_n \left( \prod_{1 \leq j < k \leq n} \frac{u_j - u_k}{u_j + u_k} \right)^2. \]

(3.11)

Because the denominators \( u_j + u_k \) have very simple factors, it is easy to prove linear reducibility along the sequence \( t_2, \ldots, t_n \) and to show that all \( E_n \) are rational linear combinations of alternating Euler sums.

We included a simple procedure \text{IsingE}(n) to evaluate them in the attached manual. In particular we can confirm the conjecture on \( E_5 \) made in [39]:

\[ > \text{IsingE}(5); \]

\[ 2\zeta_3 \left( -37 + 232 \log(2) \right) - 4\zeta_2 \left( 31 - 20 \log(2) + 64 \log^2(2) \right) \]

\[ - \frac{318}{5} \zeta_3^2 + 42 - 992\zeta_{1,-3} - 40 \log(2) + 464 \log^2(2) + \frac{512}{3} \log^4(2) \]

For illustration further exact results for \( E_n \) up to \( n = 8 \) can be found in \text{IsingE.mpl}. Time- and memory-requirements of these computations are summarized in Table 7.

3.5.1. Singularities in the domain of integration

The integration (3.9) requires that \( f(z) \in L(\Sigma)(z) \) is a hyperlogarithm without any letters \( \Sigma_+ := \Sigma \cap (0, \infty) = \emptyset \) inside the domain of integration, which ensures that \( f(z) \) is analytic on \( (0, \infty) \).

Otherwise \( f(z) \) can have poles or branch points on \( \Sigma_+ \) and the integration is then performed along a deformed contour \( \gamma \) as discussed in section 2.5. The dependence on \( \gamma \) (see figure 3) is encoded in the variables

\[ \delta_{z, \sigma} = \begin{cases} +1 & \text{when } \gamma \text{ passes below } \sigma, \\ -1 & \text{when } \gamma \text{ passes above } \sigma. \end{cases} \]  

(3.12)

Example 3.9. The integrand \( f(z) = \frac{1}{z^2} \) has a simple pole at \( z \to 1 \) and is not integrable over \((0, \infty)\). Instead, HyperInt computes the contour integrals

\begin{verbatim}
> hyperInt(1/(1-z^2), z): fibrationBasis(%);
\end{verbatim}

Warning, Contour was deformed to avoid potential singularities at \{1\}.

\[ -\frac{1}{2} \cdot I\pi \delta_{z_1} \]

Note even when positive letters \( \Sigma_+ \) occur, \( f(z) \) can be analytic on \((0, \infty)\) nonetheless. In this case the dependence on any \( \delta_{z, \sigma} \) drops out in the result.

Example 3.10. The integrand \( f(z) = -\frac{\ln(z)}{1+z^2} \) is analytic at \( z \to 1 \) and thus on all of \((0, \infty)\). It integrates to

\begin{verbatim}
> hyperInt(ln(z)/(1+z^2), z): fibrationBasis(%);
\end{verbatim}

Warning, Contour was deformed to avoid potential singularities at \{1\}.

\[ -\frac{3}{2} \zeta_2 \]

3.5.2. Detection of divergences

By default, the option \text{_hyper_check_divergences} = true is activated and triggers, after each integration, a test of convergence. The primitive \( F(z) \) is expanded as

\[ F(z) = \sum_{i=0}^{N} \log^i z \sum_{j=-M}^{\infty} z^j F_{i,j} \quad \text{at} \quad z \to 0 \]

(3.13)

and all polylogarithms \( F_{i,j} \) with \( i > 0 \) or \( j < 0 \) are explicitly checked to vanish \( F_{i,j} = 0 \) using \text{fibrationBasis}: the limit \( z \to \infty \) is treated analogously. This method is time-consuming and we recommend to deactivate this option for any involved calculations, expecting that the convergence is granted by the problem at hand.

Example 3.11. An endpoint divergence at \( z \to \infty \) is detected for \( \int_{0}^{\infty} \frac{\ln z}{z} \, dz = \lim_{z \to \infty} \log_{\omega_{-1}, \omega_0}(z) \);

\begin{verbatim}
> hyperInt(ln(z)/(1+z), z);
\end{verbatim}

Error, (in integrationStep) Divergence at \( z = \text{infinity} \) of type \( \ln(z)^{-2} \)
The expansions (3.13) are only performed up to \(_{\text{hyper_max_pole_order}}\) (default value is 10). If higher order expansions are needed, an error is reported and this variable must be increased.

Note that the expansion (3.13) is only computed at the endpoints \(z \to 0, \infty\). Polar singularities inside \((0, \infty)\) are not detected, e.g. \(_{\text{hyperInt}}\left(\frac{1}{1-z^2}, z\right)\) at such critical points \(\Sigma_+ = \{\tau_1 < \ldots < \tau_k\}\) with \(\tau_0 := 0, \tau_{k+1} := \infty\) with the effect that all singularities now lie at endpoints and will be properly analyzed by the program.

A problem arises if calculations involve periods for which no basis reduction is known to \(_{\text{HyperInt}}\), because the vanishing \(\mathcal{F}_{i,j} = 0\) of a potential divergence might not be detected. One can then set \(_{\text{hyper_abort_on_divergence}} := \text{false}\) to continue with the integration. All \(\mathcal{F}_{i,j}\) of (3.13) are stored in the table \(_{\text{hyper_divergences}}\).

Example 3.12. When periodLookups.m is not loaded,

```maple
> _hyper_splitting_field := \{I\}; > integrationStep([[1/(1+z^2), []]], z);
```

fails because factorization is initially only attempted over the rationals \(\mathbb{K} = \mathbb{Q}\). Instead we can allow for an algebraic extension \(\mathbb{K} = \mathbb{Q}(\mathbb{R})\) by specification of a set \(\mathbb{R} = \ _{\text{hyper_splitting_field}}\) of radicals:

```maple
> _hyper_splitting_field := \{I\}; > integrationStep([[1/(1+z^2), []]], z);
```

We can also go further and factorize over the full algebraic closure \(\mathbb{K} = \mathbb{Q}(\mathbb{Z})\) by setting \(_{\text{hyper_algebraic_roots}} := \text{true}\). Over \(\mathbb{K}\), all rational functions \(\mathbb{Q}(\mathbb{Z})\) factor linearly such that we can integrate any \(f \in \text{Res}_{0,\infty}(\mathbb{Z}(\mathbb{Z})(t))\) as long as we start with rational letters \(\Sigma \subset \mathbb{Q}(\mathbb{Z})\).

This feature is to be considered experimental and only applied in \(_{\text{transformWord}}\) which implements lemma 2.7. Given an irreducible polynomial \(P \in \mathbb{Q}(z)\) and a distinguished variable \(z\), the symbolic notation

\[
\omega_{\text{Root}(P,z)} := \sum \{\omega_{\xi_0}: P|_{z=\xi_0} = 0\}
\]

sums the letters corresponding to all the roots of \(P\).

Example 3.13. A typical situation looks like this:

```maple
> f,g := \text{Hlog}(x,-z,x+x^2), \text{Hlog}(x,[x+x^2,-z]); > fibrationBasis(f,g, [x, z]);
```

To express \(f + g\) as a hyperlogarithm in \(x\), the roots \(R = \text{Root}(P,x) = \{-\frac{1}{2} \pm \frac{\sqrt{3}}{2}\}\) of \(P = z + x + x^2\) seem necessary. After allowing for such algebraic letters, we obtain:

```maple
> _hyper_algebraic_roots := \text{true}; > fibrationBasis(f+g, [x, z]);
```

Since this result actually does not involve \(\omega_R\) at all one might wonder why it was necessary in the first place. The reason is that the individual contributions \(f\) and \(g\) indeed need \(\omega_R\). Only in their sum this letter drops out.\(^4\)

---

\(^4\)In this extremely simple example this is clear since by (2.49), \(f + g = \text{Log}_{-\frac{1}{2}}(z) \cdot \text{Log}_{\frac{1}{2}}(z)\) factorizes into \(\log \frac{1}{1+z} \cdot \log \frac{1}{1+z}\). We thus see why our representation (5.1) is preferable to one where all products of words are multiplied out (as shuffles).

---

| \(n\) | \(1\) | \(2\) | \(3\) | \(4\) | \(5\) | \(6\) | \(7\) | \(8\) |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| time | 10 ms | 41 ms | 52 ms | 235 ms | 2.0 s | 40.6 s | 29.3 min | 28 h |
| RAM  | 35 MiB | 51 MiB | 51 MiB | 76 MiB | 359 MiB | 1.6 GiB | 1.9 GiB | 30 GiB |

Table 1: Resources consumed during computation of the Ising-type integrals \(E_n\) of (3.11) running on Intel® Core™ i7-3770 CPU @ 3.40 GHz. The column with \(n = 1\) (when \(E_n := 1\)) requires no actual computation and shows the time and memory needed to load periodLookups.m.

3.6. Factorization of polynomials

Since we are working with hyperlogarithms throughout, it is crucial that all polynomials occurring in the calculation factor linearly with respect to the integration variable \(z\). For example,

```maple
> integrationStep([[1/(1+z^2), []]], z);
```

Error, (in partialFractions) \(1+z^{-2}\) is not linear in \(z\)

---

4In this extremely simple example this is clear since by (2.49), \(f + g = \text{Log}_{-\frac{1}{2}}(z) \cdot \text{Log}_{\frac{1}{2}}(z)\) factorizes into \(\log \frac{1}{1+z} \cdot \log \frac{1}{1+z}\). We thus see why our representation (5.1) is preferable to one where all products of words are multiplied out (as shuffles).
Note that further processing of functions with such algebraic letters\footnote{These are sometimes referred to as generalized harmonic polylogarithms with nonlinear weights.} is not supported by HyperInt, because their integrals are in general not hyperlogarithms anymore. However, the case of example 3.13 occurs frequently wherefore the option \texttt{hyper\_ignore\_nonlinear\_polynomials} (default value is \texttt{false}) is available to ignore all algebraic letters in the first place. That is, all words containing such a letter are immediately dropped when it is set to \texttt{true}.

In the example above this gives the correct result for $f+g$, but will provoke false answers when \texttt{fibrationBasis} is applied to $f$ or $g$ alone. Hence this option should only be used when linear reducibility is granted; preferably using the methods of section 4.

3.7. Additional functions

In the manual we describe some further procedures provided by HyperInt (note that all algorithms of section 2 are were implemented), like the extension of the commands \texttt{diff} and \texttt{series} to compute differentials and series expansions of hyperlogarithms.

3.8. Performance

During programming we focussed on correctness and we are aware of considerable room for improvement of the efficiency of HyperInt. But we hope that our code and the details provided in section 2 will inspire further, streamlined implementations, even outside the regime of computer algebra systems. This is possible since apart from the factorization of polynomials (which can be performed before the actual integration, see the next section), all operations boil down to elementary manipulations of words (lists) and computations with rational functions.

Ironically, often just decomposing into partial fractions becomes a severe bottleneck practice, as was also noted in \cite{18}. This happens when an integrand contains denominator factors to high powers or very large polynomials in the numerator.

We observed that Maple consumes a lot of main memory, in very challenging calculations the demand grew beyond 100 GIB. Often this turns out to be the main limitation in practice.

Our program uses some functions that are not thread-safe and can therefore not be parallelized automatically. However, since the integration procedure considers every hyperlogarithm individually, a manual parallelization is straightforward: Multiple instances of Maple can each compute a different piece of an integral whose results can be added up afterwards. Some example scripts are provided and discussed in the manual.

Also note that the product representation (3.1) inherently allows for different representations of the same words, because a product can either be represented symbolically or as the corresponding sum of shuffles. We argued that shuffling out every product is not desirable, so a better solution could be to choose an order on the alphabet $\Sigma$, which then gives rise to a polynomial basis of the shuffle algebra $T(\Sigma)$ in terms of Lyndon words \cite{10}.

4. Polynomial reduction and linear reducibility

In order to compute multi-dimensional integrals (1.4) by iterated integration using the algorithms of section 2 we must require that for each $k$, the partial integral

$$f_k \in L(\Sigma_k) (\omega_{k+1}) \quad \text{where} \quad \Sigma_k \subset \mathbb{C} (z_{k+2}, \ldots, z_n)$$

is a hyperlogarithm in the next integration variable $z_{k+1}$. The alphabet $\Sigma_k$ is restricted to rational functions of the remaining variables, in particular $\Sigma_k \subset \mathbb{C} (z_{k+2})$, because only then lemma 2.7 guarantees that its integral $f_{k+1} \in L((\Sigma_k)z_{k+2}) (\omega_{k+2})$ is a hyperlogarithm in $z_{k+2}$.

**Definition 4.1.** We call $f_0(z)$ linearly reducible if for some ordering $z_1, \ldots, z_n$ of its variables, sets $\Sigma_k$ exist such that (4.1) holds for all $0 \leq k < n$.

For illustration let us suppose we want to integrate

$$f_0(x, y, z) := \frac{1}{((1 + x)^2 + y)(y + z^2)}$$

over $x$ and $y$. To integrate $x$, we must include in $\Sigma_x$ the algebraic zeros $1 \pm \sqrt{y}$ to get a hyperlogarithm $f_0'(x) \in L(\Sigma_x) (x)$ in $x$. But then the integral

$$\int_0^\infty f_0 \, dx = \arctan \frac{\sqrt{y}}{\sqrt{y} + z^2}$$

is not a hyperlogarithm in $y$ at all\footnote{But it is a hyperlogarithm in $t := \sqrt{y}$, so in this simple case a change of variables would help us out.}. On the other hand, since $f_0(y) \in L (\{-1, -z^2\}) (y)$ for letters rational in $x$, integration of $y$ results in a hyperlogarithm

$$\int_0^\infty f_0 \, dy = 2 \frac{\log (1 + x) - \log z}{(x + 1 + z)(x + 1 - z)}$$

in $x$ over letters $\{-1, -1 \pm z\}$. So in the order $z_1 := y$, $z_2 := x$ linear reducibility is given and we can integrate

$$\int_0^\infty dx \int_0^\infty dy \ f_0 = \frac{L_{\omega_1} \omega_0 (z) - L_{\omega_1 - \omega_0} (z)}{z},$$

which is a harmonic polylogarithm in $z$.\hspace{1cm} \Box
In principle, we can try to integrate \( f_0 \) for some arbitrary order and verify, after each step, that \( \Sigma_k \) is rational (or otherwise abort and try a different order). But fortunately this is not necessary since there are means to analyse the singularities of the integrals \( f_k \) in advance.

Namely, polynomial reduction algorithms were presented in [11] and [12]. These compute, for each subset \( I \subseteq E := \{ z_1, \ldots, z_n \} \) of variables, a set \( S_I \subseteq \mathbb{Q}[E \setminus I] \) of irreducible polynomials that provide an upper bound of the Landau varieties as introduced in [12]. In particular this means that if there exists an ordering \( z_1, \ldots, z_n \) of the variables such that all \( p \in S_{I_k} \) are linear in \( z_{k+1} \), for any \( 0 \leq k < n \) and \( I_k := \{ z_1, \ldots, z_k \} \), then the linear reducibility (4.1) is granted with the rational alphabets

\[
\Sigma_k := \{ 0 \} \cup \bigcup_{p \in S_{I_k}} \{ \text{zeros of } p \text{ in w.r.t. } z_{k+1} \}. \quad (4.6)
\]

Readers familiar with the symbol calculus will realize that the polynomials \( S_{I_k} \) provide an upper bound of the entries of the symbol of \( f_k \). Explicit examples of such reductions are worked out in [11 12] and the appendix of [24].

### 4.1. Performance

A polynomial reduction can significantly speed up the calculation of integrals [13]. During the step when \( f_k \) is rewritten as a hyperlogarithm in \( z_{k+1} \) following section 2.4, all words that contain a letter not in \( \Sigma_k \) can be dropped, since the knowledge of (4.6) proves that all such contributions must in total add up to zero (see example 3.13, where the algebraic roots \( \omega_{\text{Root}}(p_{z_1}) \) drop out for \( f + g \)).

Note that the dimension of the space of hyperlogarithms over an alphabet \( \Sigma_k \) grows exponentially with the weight. Therefore, a polynomial reduction is absolutely crucial for problems of high complexity and cutting down the number of polynomials in \( \Sigma_k \) is highly desirable. In practice this means that after computation of a polynomial reduction, one should look for a sequence \( z_1, \ldots, z_n \) of variables not only ensuring that \( S_{I_k} \) are linear in \( z_{k+1} \), but also minimizing the number of \( z_{k+1} \)-dependent polynomials in \( S_{I_k} \).

### 4.2. Implementation in HyperInt

HyperInt implements the compatibility graph method [12] of polynomial reduction and provides it as the command \texttt{cgReduction}(L). The entries \( L_I = [S_I, C_I] \) of the table \( L \) are pairs of polynomials \( S_I \) and edges \( C_I \subset (S_I^2) \) between them.

**Example 4.2.** The reduction of the integrand \( f_{2} \) starts with the complete graph on the factors of its denominator:

\[
\begin{align*}
S &= \{ x^2 + 2 x + 1, y, y + z^2 \}; \\
L &= \{ \{ \} \}; \quad \text{(4.2)}
\end{align*}
\]

\[
L_{\{x\}_1} = \{ x + 1, x + 1 + z, x + 1 - z \};
\]

\[
\{ 1 + z, z - 1 \}
\]

We see that the results for \( S_{\{x\}} \) and \( S_{\{x,y\}} \) match with the letters of (4.4) and (4.5), but \( S_{\{z\}} \) is not computed because \( S_{\{z\}} \) is not linear in \( x \).

Our implementation can use the knowledge of such reductions in two places (examples are given in the manual):

- When a table \( S \) is supplied as the (optional) fourth parameter to \texttt{fibrationBasis}, then all words \( w, k \) in \( S \) containing letters not in \( \Sigma_k \) of (4.6) are removed from the result.

- In the first step of integrating \( \int_0^\infty f \, dz \), the integrand \( f \) is rewritten as a hyperlogarithm in \( z \) using \texttt{transformWord}(f, z) = \sum w \cdot c_w \). Setting

\[
\begin{align*}
\_\text{hyper\_restrict\_singularities} &= \text{true}; \\
\_\text{hyper\_allowed\_singularities} &= S;
\end{align*}
\]

ensures that any word \( w \) containing a letter that is not a zero of some polynomial \( p(z) \in S \) is dropped.

### 4.3. Spurious polynomials and changes of variables

Bear in mind that the sets \( S_I \) only provide upper bounds on the alphabet. In course of our calculations we regularly observed that, with the number \( |I| \) of integrated variables increasing, more and more polynomials in \( S_I \) tend to be spurious. In extreme cases it happens that a reduction contains surplus non-linear polynomials in every variable, while \( f_0 \) actually is linearly reducible.

But even when linear reducibility strictly fails, it is sometimes possible to change variables such that the integrand becomes linearly reducible in these new variables. We explain this in [24] using the example of a divergent, massive four-point box integral. Similar transformations are also employed in [10] to calculate generating functions of operator insertions into finite one-scale integrals. Also note the discussion [41] of alphabets containing square root letters that are typical for applications in particle physics and can be rationalized through simple changes of variables.

### 5. Application to Feynman integrals

In section 2 we investigated hyperlogarithms on their own, but the algorithms were originally developed in [11] for the computation of Feynman integrals. Important results on their linear reducibility (including counterexamples) and the geometry of Feynman graph hypersurfaces were obtained in [12]. In [23 24] we successfully applied our implementation to compute many non-trivial examples, including massless propagators up to six loops and also divergent integrals depending on up to seven kinematic invariants. All results\(^7\) presented in these papers were computed using this program HyperInt.

\(^7\)These can be downloaded from \url{http://www.math.hu-berlin.de/~panzer/}
Laurent series the integrand in ε assuming that (5.1) is convergent ε.

5.2. ε-expansion

The popular method of Schwinger parameters [31] expresses Feynman integrals \( \Phi(G) \) associated to Feynman graphs \( G \) by

\[
\Phi(G) = \Gamma(\text{sd}) \prod_{e \in E} \int_0^\infty \frac{\alpha^\varepsilon_{e-1}}{\Gamma(\alpha_e)} \frac{\varphi^{\varepsilon_{\text{sd}}} \cdot |\cdot|^{2-\varepsilon_{\text{sd}}}}{\delta(1-\alpha_{EN})}
\]

in \( D \) space-time dimensions. To each edge \( e \in E \) of the graph corresponds a Schwinger variable \( \alpha_e \), and the corresponding scalar propagator may be raised to some power \( \alpha_e \). The superficial degree of divergence is \( \text{sd} := \sum_{e \in E} \alpha_e - |G| \cdot \frac{D}{2} \) for the loop number \( |G| \) of \( G \). The two graph polynomials \( \psi \) and \( \varphi \) are for example defined in [52], the \( \delta \)-distribution freezes an arbitrary \( \alpha_{EN} \).

5.2. ε-expansion

For calculations in dimensional regularization[8], we set \( D = 4 - 2\varepsilon \) and also the edge powers \( \alpha_e = A_e + \varepsilon \beta_e \) are \( \varepsilon \)-dependent and expanded near an integer \( A_e \in \mathbb{Z} \). Assuming that (5.1) is convergent[9] for \( \varepsilon = 0 \), we can expand the integrand in \( \varepsilon \) and obtain each coefficient \( c_n \) of the Laurent series \( \Phi(G) = \sum_n c_n \varepsilon^n \) as period integrals

\[
c_n = \Gamma(\text{sd}) \prod_{e \in E} \int_0^\infty \frac{d\alpha_e}{\Gamma(\alpha_e)} \cdot \frac{P^{(n)}}{Q^{(n)}} \cdot \delta(1-\alpha_{EN}) \quad (5.2)
\]

where \( P^{(n)}, Q^{(n)} \in \mathbb{Q}[\alpha] \) denote polynomials and \( f^{(n)} \in \mathbb{Q}[\alpha, \log \alpha, \log \varphi, \log \psi] \). In particular \( f^{(n)} \in L(\Sigma_e)(\alpha_e) \) is a hyperlogarithm in \( \alpha_e \) whenever \( \varphi \) and \( \psi \) are linear in \( \alpha_e \). If \( f^{(n)} \) even turns out to be linearly reducible, we can integrate it with HyperInt.

5.3. Additional functions in HyperInt

In Appendix C.4 we list the most important functions that support the calculation of Feynman integrals. These entail simple routines to construct the graph polynomials \( \psi \) and \( \varphi \).

For divergent integrals, the parametric integrands in the representation (5.1) can be divergent. Such a situation demands partial integrations, which effectively implement the analytic (dimensional) regularization and produce a convergent integral representation in the end. This procedure is defined and exemplified in [24] and implemented into HyperInt as described in the manual.

5.4. Examples

Plenty of examples are provided in the Maple worksheet Manual.mw, wherefore we only present a very brief case of a four-loop massless propagator here.

First we define the graph of figure 4 by its edges \( E \) and specify two external momenta of magnitude one entering the graph at the vertices 1 and 3. The polynomials \( \psi \) and \( \varphi \) can be computed with

\[
E := [[1, 2], [2, 3], [3, 4], [4, 1], [5, 1], [5, 2], [5, 3], [5, 4]]:
\]

\[
\psi := \text{graphPolynomial}(E):
\]

\[
\phi := \text{secondPolynomial}(E, [[1, 1], [3, 1]]):
\]

This graph has vertex-width three [12] and is therefore linearly reducible. Still let us calculate a polynomial reduction to verify this claim:

\[
L := \text{table}(): S := \text{irreducibles}(\psi, \phi):
\]

\[
L(\{}\{\}:=[S, \{\}]):= \text{cgReduction}(L):
\]

Afterwards we can investigate the polynomial reduction (for example with the procedure reductionInfo(L)) and find a linearly reducible sequence \( \bar{Z} \) of variables. We recommend to always check this with

\[
z := [x[1], x[2], x[6], x[5], x[3], x[4], x[7], x[8]]:
\]

\[
\text{checkIntegrationOrder}(L, z[1..7]):
\]
1. \((x[1]): 2\) polynomials, 2 dependent
2. \((x[2]): 5\) polynomials, 4 dependent
3. \((x[6]): 8\) polynomials, 4 dependent
4. \((x[5]): 7\) polynomials, 4 dependent
5. \((x[3]): 6\) polynomials, 6 dependent
6. \((x[4]): 4\) polynomials, 3 dependent
7. \((x[7]): 1\) polynomials, 1 dependent

Final polynomials:

\[
\{\}
\]

The integrand is assembled according to (5.1) which in this case is already convergent as-is. We expand to second order in \(\varepsilon\) with

\[
> \text{sdd := nops(E)-(1/2)*4*(4-2*epsilon)};
> f := series(\text{psi}^\text{-}(1-2*epsilon+\text{sdd})*\text{phi}^\text{-}(-\text{sdd}), \text{epsilon}=0);
> \text{f:=add(coeff(f,epsilon,n)*epsilon^n,n=0..2)};
> \text{f := series(psi^(-2+epsilon+sdd)*phi^(-sdd), epsilon=0)};
\]

Now we integrate out all but the last Schwinger parameter

\[
> \text{hyperInt(f, z[1..-2])}:
\]

and reduce the result into a basis of MZV:

\[
> \text{fibrationBasis(f)*z[1..-1]}:
> \text{collect(%, epsilon)};
\]

\[
\left(254\zeta_3 + 780\zeta_5 - 200\zeta_3\zeta_5 - 196\zeta_2^3 + 80\zeta_2^3 - \frac{168}{5}\zeta_2^3\zeta_3\right) \varepsilon^2 \\
+ \left(-28\zeta_2^3 + 140\zeta_5 + \frac{80}{7}\zeta_2^3\right) \varepsilon + 20\zeta_5.
\]

Examples containing more external momenta, massive propagators and also divergences are included in Manual.mw.

Acknowledgments. I thank Francis Brown for his beautiful articles and Dirk Kreimer for continuous encouragement. Oliver Schnetz kept me interested into graphical functions and kindly verified many of my computations with his very own methods, thereby providing a strong cross-check. Also Johannes Henn provided some \(\varepsilon\)-expansions to me for tests, and he motivated the study of divergent integrals in the parametric representation. Many discussions with Christian Bogner of concrete examples and problems greatly improved my understanding of iterated integrals. Figures were generated with JaxoDraw [56].

Appendix A. Tests of the implementation

We extensively tested our implementation with a variety of examples. Most of these are supplied in the file HyperTests.mpl which must run without any errors. Since it contains many diverse applications of HyperInt, it might also be useful as a supplement to the manual.

Plenty of functional and integral equations of polylogarithms, taken from the books [3, 57], are checked with HyperInt. These tests revealed a few misprints in [3):

- Equation (7.93): \(-\frac{9}{4}\pi^2 \log^2(\zeta)\) must be \(-\frac{9}{4}\pi^2 \log^2(\xi)\).
- Equation (7.99), repeated as (44) in appendix A.2.7: The second term \(-\frac{9}{4}\pi^2 \log^2(\zeta)\) of the last line must be replaced with \(-\frac{9}{4}\pi^2 \log^2(\zeta)\).
- Equation A.3.5. (9): The terms \(-2\text{Li}_3(1/x) + 2\text{Li}_3(1)\) should read \(\text{Li}_3(1/x) - \text{Li}_3(1)\) instead.
- In equation (7.132), a factor \(\frac{1}{4}\) in front of the second summand \(D^a_{p=1} \{\cdots\}\) is missing (it is correctly given in 7.131).
- Equation (8.80): \((1 - v)\) inside the argument of the fourth \(\text{Li}_2\)-summand must be replaced by \((1 + v)\), so that after including the corrections mentioned in the following paragraph, the correct identity reads

\[
0 = \text{Li}_2 \left(\frac{(1 + v)w}{1 + w}\right) + \text{Li}_2 \left(\frac{-(1 - v)w}{1 - w}\right) \\
+ \text{Li}_2 \left(\frac{(1 - v)w}{1 + w}\right) + \text{Li}_2 \left(\frac{-(1 + v)w}{1 - w}\right)
\]

(A.1)

- Equation (16.46) of [57]: \(x^4\) must read \(x^2\).
- Equation (16.57) of [57]: \(x^4\) must read \(x^2\).

Some tests are constructed by calculation of parametric integrals with known results in terms of polylogarithms and MZV. We used the expansion of Euler’s beta function in the form

\[
\exp \left[ \sum_{n=2}^{\infty} \frac{\zeta(n)}{n} (x^n + y^n - (x + y)^n) \right] = \int_0^\infty \frac{z^{-x}}{(1 + z)^{2-x-y}} dz
\]

and also checked the identity \((z \geq 0)\)

\[
\int_0^\infty \left[ \left( \frac{1}{x - z} - \frac{1}{x + z} \right) \text{Li}_n(-x - z) - \frac{1}{x} \text{Li}_n\left( -\frac{z}{x + 1} \right) \right] dx = n \text{Li}_{n+1}(-z),
\]

(A.2)

which is easily derived inductively for any \(n \in \mathbb{N}\).

The two families of “bubble graphs” shown in figure A.10 can be calculated with standard techniques in momentum-space. Following the forest formula, we get

\[
\mathcal{P}(B_{n,m}) := \left. \frac{\partial}{\partial q^2} \right|_{q^2=1} \Phi_R(B_{n,m}) = (n + m)! \quad (A.3)
\]

for the derivative of the Feynman integrals \(\Phi_R\) renormalized by subtraction at external momentum \(q^2 = 1\). The second family has generating function

\[
\ln \left\{ (1 - x - y) \sum_{n,m \geq 0} \frac{n^m}{n!m!} \mathcal{P}(B_{n,m}) \right\} \\
= \sum_{r \geq 1} \frac{2r+1}{2r+1} \left[ x^{2r+1} + y^{2r+1} - (x + y)^{2r+1} \right] \quad (A.4)
\]
and we used (A.3) and (A.4) to verify our results obtained from the parametric integral representations for these periods derived in [68].

We furthermore tested some simple period integrals of [10] and transformations of polynomials into hyperlogarithms given in [59]. Our results for the integrals $E_n$ of (3.11) match the analytic results up to $n = 4$ given in [39] and the numeric values obtained therein for $E_5$ and $E_6$ agree with our exact results.

Probably the strongest tests of our implementation are the computations of $\varepsilon$-expansions of various single-scale [25] and multi-scale [24] Feynman integrals. We cross-checked these results with many different references, verified that they obey the symmetries of the associated Feynman graphs and in some cases used established programs [60, 61] to obtain numeric evaluations to confirm our analytic formulas.

Also we verified the operator matrix elements $I_{1a}$, $I_{1b}$, $I_{2a}$, $I_{2b}$, $I_4$ of ladder graphs computed in [14] and checked the Benz graphs $I_1$, $I_2$ and $I_3$ of [10]. The examples of $I_4$ and $I_1$ are part of our manual, where we also correct mistakes in the equations (3.18) and (3.1) loc.cit.

Another check was done with the massless hexagon integral [62, 63], which is also included in the manual.

Appendix B. Proofs

Lemma 2.3. The statement is trivial for $n = 1$ and for $n > 1$ we apply (2.2) to (2.1) such that the right-hand side becomes

$$\sum_{0 \leq k < r \leq n} \left( \omega_{\sigma_1} - \omega_{-1} \right) \left[ \omega^k_{\sigma_1} \omega (-\omega_{-1})^{r-k-1} \omega \omega_{\sigma_{r+1}} \ldots \omega_{\sigma_n} \right]$$

$$+ \omega_{-1} \sum_{k=1}^n \left[ \omega^{k-1} \omega \operatorname{reg} \infty \left( \omega_{\sigma_{r+1}} \ldots \omega_{\sigma_n} \right) \right].$$

The second contribution is $\omega_{-1} \omega_{\sigma_2} \ldots \omega_{\sigma_n}$ by the induction hypothesis, while the sum $\sum_{k=0}^{r-1} \omega^k_{\sigma_1} \omega (-\omega_{-1})^{r-k-1} = (\omega_{-1} - \omega_{-1})^{r-1} = \delta_{r1}$ reduces the first contribution to $(\omega_{\sigma_1} - \omega_{-1}) \omega_{\sigma_3} \ldots \omega_{\sigma_n}$. \qed

Lemma 2.18. Consider a sequence $\omega^a_r$ in a word $w = u(\omega^a_r)v$ where $u \in \text{im}(\operatorname{reg}_\tau)$ does not end in $\omega_\tau$ and $v \in \text{im}(\operatorname{reg}_\tau)$ does not begin with $\omega_\tau$. The contributions to (2.45) that split $w$ between $u$ and $v$ are

$$\sum_{k=0}^n \int_{\gamma_u} \left( \omega^a_r \right) \cdot \int_{\gamma_v} \left( \omega^a_n - k \right) \cdot \int_{\eta_u} \left( \omega^a_n - k \right) \cdot \int_{\eta_v} \left( \omega^a_n - k \right).$$

From lemma 2.2 we obtain the identities

$$u(\omega^a_n) = \sum_{k=0}^n \left( \omega^a_n - k \right) \cdot \int_{\eta_u} \left( \omega^a_n - k \right) \cdot \int_{\eta_v} \left( \omega^a_n - k \right).$$

which allow us to rewrite (2.5) as

$$\sum_{\mu + v + a = n} \int_{\gamma_u} \left( \omega^a_n \right) \cdot \int_{\gamma_v} \left( \omega^a_n - k \right) \cdot \int_{\eta_u} \left( \omega^a_n - k \right) \cdot \int_{\eta_v} \left( \omega^a_n - k \right).$$

The sum over $a + b = n$ is $\mu - \nu$ of the last two terms combined to $\int_{\gamma_u} \omega^a_n \cdot \omega^b_n$. Now the limit $u \rightarrow \tau$ in the remaining two factors is finite, such that (2.5) then becomes

$$\sum_{\mu + v \leq n} \int_{\tau} \left( \omega^a_n \right) \cdot \int_{\tau} \left( \omega^a_n - k \right) \cdot \int_{\tau} \left( \omega^a_n - k \right) \cdot \int_{\tau} \left( \omega^a_n - k \right).$$

$$= \sum_{\mu=0}^n \int_{\tau} \left( \omega^a_n \right) \cdot \int_{\tau} \left( \omega^a_n - k \right) \cdot \int_{\tau} \left( \omega^a_n - k \right).$$

$$= \sum_{\mu=0}^n \int_{\tau} \left( \omega^a_n \right) \cdot \int_{\tau} \left( \omega^a_n - k \right) \cdot \int_{\tau} \left( \omega^a_n - k \right).$$

$$= \sum_{\mu=0}^n \int_{\tau} \left( \omega^a_n \right) \cdot \int_{\tau} \left( \omega^a_n - k \right) \cdot \int_{\tau} \left( \omega^a_n - k \right).$$

$$= \sum_{\mu=0}^n \int_{\tau} \left( \omega^a_n \right) \cdot \int_{\tau} \left( \omega^a_n - k \right) \cdot \int_{\tau} \left( \omega^a_n - k \right).$$
We used \( \text{B.1} \) again and the definition \( \int_{r}^{\infty} \omega_{r} := \int_{r}^{\infty} \omega_{r} \). \[ \square \]

Appendix C. List of functions and options provided by HyperInt

Appendix C.1. Options and global variables

**_hyper_vercity** (default: 1)
The higher this integer, the more progress information is printed during calculations. The value zero means no such output at all.

**_hyper_verbose_frequency** (default: 10)
Sets how often progress output is produced during integration or polynomial reduction.

**_hyper_return_tables** (default: false)
When true, integrationStep returns a table instead of a list. This is useful for huge calculations, because Maple can not work with long lists.

**_hyper_check_divergences** (default: true)
When active, endpoint singularities at \( z \to 0, \infty \) are detected in the computation of integrals \( \int_{0}^{\infty} f(z) \, dz \).

**_hyper_max_pole_order** (default: 10)
Sets the maximum values of \( i \) and \( j \) in (3.13) for which the functions \( f_{i,j} \) are computed to check for potential divergences \( f_{i,j} \neq 0 \).

**_hyper_abort_on_divergence** (default: true)
This option is useful when divergences are detected erroneously, as happens when periods occur for which no basis is supplied to the program.

**_hyper_divergences**
A table collecting all divergences that occurred.

**_hyper_splitting_field** (default: \( \emptyset \))
This set \( R \) of radicals defines the field \( k = \mathbb{Q}(R) \) over which all factorizations are performed.

**_hyper_ignore_nonlinear_polynomials** (default: false)
Set to true, all non-linear polynomials (that would result in algebraic zeros as letters) will be dropped during integration. This is permissible when linear reducibility is granted.

**_hyper_restrict_singularity** (default: false)
When true, the rewriting of \( f \) as a hyperlogarithm in \( z \) (performed during integration) projects onto the algebra \( L(\Sigma) \) of letters \( \Sigma \) specified by the roots of the set \_hyper_allowed_singularity (default: \( \emptyset \)) of irreducible polynomials. This can speed up the integration.

**_hyper_algebraic_roots** (default: false)
When true, all polynomials will be factored linearly which can introduces algebraic functions. Further computations with such functions are not supported.

**Appendix C.2. Maple functions extended by HyperInt**

**convert(f, form)** with \( form \in \{ \text{Hlog, Mpl, HlogRegInf} \} \)
Rewrites polylogarithms \( f \) in terms of hyper-or polylogarithms using \( \text{[1,3]} \). Choosing \( form = \text{HlogRegInf} \) transforms \( f \) into the list representation (3.1).

**diff(f, z)**
Computes the partial derivative \( \partial_{z} f \) of polylogarithms \( \text{Hlog}(g(t), w(t)) \) or polylogarithms \( \text{Mpl}(\vec{n}, \vec{z}(t)) \) that occur in \( f \). This works completely generally, i.e. also when a word \( w(t) \) depends on \( t \).

**series(f, z = 0)**
Implements the expansion of \( f = L_{w}(z) \) at \( z \to 0 \).
To expand at different points, use \_fibrationBasis first as explained in the manual.

**Appendix C.3. Some new functions provided by HyperInt**

Note that there are further functions in the package, cf. the manual.

**integrationStep(f, z)**
Computes \( \int_{0}^{\infty} f(z) \, dz \) for \( f \) in the form (3.1).

**hyperInt(f, \vec{z})** with a list \( \vec{z} = [z_{1}, \ldots, z_{r}] \) or single \( \vec{z} = z_{1} \)
Computes \( \int_{0}^{\infty} dz_{r} \cdots \int_{0}^{\infty} dz_{1} f \) from right to left. Any variable can also specify the bounds \( z_{i} = a_{i..b_{i}} \) to compute \( \int_{a_{i}}^{b_{i}} dz_{i} \) instead.

**fibrationBasis(f, [z_{1}, \ldots, z_{r}], F, S)**
Rewrites \( f \) as an element of \( L(\Sigma_{1})(z_{1}) \otimes \cdots \otimes L(\Sigma_{r})(z_{r}) \). Note that \( \Sigma_{i} \subset \mathbb{C}(z_{i+1}, \ldots, z_{r}) \) in general are algebraic functions of the following variables. A table \( F \) (with indexing function \_sparseReduced) can be supplied to store the result in compact form, otherwise \_Hlog-expressions are returned.

For each defined key \( z_{i} \) of \( S \), the result is projected from \( L(\Sigma_{1})(z_{1}) \otimes \cdots \otimes L(\Sigma_{r})(z_{r}) \) restricting to letters \( \Sigma_{i}' := \{ \text{zeros of } p(z_{i}) \mid p \in S_{z_{i}} \} \). All words including other letters are dropped in the computation.

**index/sparseReduced**
This indexing function corresponds to Maple\’s \_sparse, but entries with value zero are removed. It is used to collect coefficients of hyperlogarithms.

**forgetAll()**
Clears cache tables for internal functions and should be called when options were changed.

**transformWord(w, t)**
Given a word \( w = [\sigma_{1}, \ldots, \sigma_{n}] \) as a list, this function returns a list \( \left[ [u_{1}, u_{2}], \ldots \right] \) of pairs such that

\[
\text{Reg} \sum_{z \to \infty} L_{w}(z) = \sum_{i} L_{u_{i}}(t) \cdot \text{Reg} L_{u_{i}}(z)
\]

and implements the algorithm of section \( \text{2.4} \). Note that \( u_{i} \) is given in the product form (3.1).
integrates $f(z)$

Takes a hyperlogarithm $f(z)$ in the form (3.3) and computes a primitive $F(\partial_z F(z) = f(z))$ following section 2.2.

cgReduction($L$, todo, d)

Computes compatibility graphs (stored in the table $L$) for the variable sets in the list $\text{todo}$, only taking reductions into account where every polynomial is of degree $d$ or less (default $d = 1$).

When $\text{todo}$ is a set, then all reductions of variables not overlapping this set are computed.

checkIntegrationOrder($L$, $\vec{z}$)

Tests whether for $\vec{z} = [z_1, \ldots, z_l]$, all polynomials in the reduction $L$ are linear in the corresponding $z_i$ and prints the number of polynomials.

Appendix C.4. Functions related to Feynman integrals

graphPolynomial($E$)

Computes $\psi$ for the graph with edges $E = [e_1, \ldots]$ given as a list of pairs of vertices $e_i = [v_{1i}, v_{2i}]$. The vertices $V = \{1, \ldots, |V|\}$ must be numbered consecutively.

forestPolynomial($E$, $P$)

The spanning forest polynomial $\Phi^P$ of $|E|$ of the graph $E$, $P$ is a partition of a subset of vertices.

secondPolynomial($E$, $P$, $M$)

Computes $\varphi$ for the graph with edges $E$ that denote scalar propagators of masses $M$ (optional). $P = [[v_{11}, p_{11}^2], \ldots]$ lists the vertices $v_i$ to which external momenta $p_i$ are attached.

graphicalFunction($E$, $V_{\text{ext}}$)

Constructs the parametric integrand for a graphical function as defined in (3.3) in $D = 4$. The edge list $E = [e_1, \ldots]$ can contain lists $e_i = [v_{1i}, v_{2i}]$ to denote propagators and sets $e_i = \{v_{1i}, v_{2i}\}$ to denote inverse (numerator) propagators. The external vertices are $V_{\text{ext}} = [v_2, v_0, v_1, v_\infty]$ with $v_\infty$ optional.

drawGraph($E$, $P$, $M$, $s$)

Draws the graph defined by the edge list $E$. The remaining parameters are optional: $P$ and $M$ are as for secondPolynomial while $s$ sets the style of the drawing (see GraphTheory[DrawGraph]).

References

[1] A. B. Goncharov, Multiple polylogarithms, cyclotomy and modular complexes, Math.Res.Lett. 5 (1998) 497–516. arXiv:1106.2076.

[2] J. M. Borwein, D. M. Bradley, D. J. Broadhurst, P. L. Zvengrowski, Complementary sequences of multisummable $\ln$-harmonic series, J. Phys. A 35 (2002) 3957–3988.

[3] J. Blümlein, C. Schneider, Analytic and algorithmic aspects of generalized harmonic sums and polylogarithms, J. Phys. A 44 (2011) 105204.

[4] J. Blümlein, C. Schneider, Multiple polylogarithms and the Lerch transcendent, in: J. Blümlein (Ed.), Multiple Polylogarithms and Their Applications, Fields Inst. Commun. 50 (2006) 33–72.

[5] J. M. Borwein, D. M. Bradley, D. J. Broadhurst, P. L. Zvengrowski, Complementary sequences of multisummable $\ln$-harmonic series, J. Phys. A 35 (2002) 3957–3988.

[6] J. A. Lappo-Danilevsky, Mémoires sur la théorie des systèmes des équations différentielles linéaires, Chelsea, 1953.

[7] D. Maître, HPL, a mathematica implementation of the harmonic polylogarithms, Comput.Phys.Commun. 183 (3) (2012) 846–865. arXiv:0801.0378.

[8] D. Maître, Extension of HPL to complex arguments, Comput.Phys.Commun. 183 (3) (2012) 846. arXiv:0801.0378.

[9] J. Ablinger, J. Blümlein, C. Schneider, Analytic and algorithmic aspects of generalized harmonic sums and polylogarithms, J. Phys. A 44 (2011) 105204.

[10] J. M. Borwein, D. M. Bradley, D. J. Broadhurst, P. L. Zvengrowski, Complementary sequences of multisummable $\ln$-harmonic series, J. Phys. A 35 (2002) 3957–3988.

[11] F. Brown, The Massless Higher-Loop Two-Point Function, Communications in Mathematical Physics 287 (2009) 925–958.

[12] F. Brown, On the periods of some Feynman integrals, arXiv:1207.0114.

[13] F. Chavez, C. Duhr, Three-mass triangle integrals and single-valued polylogarithms, Journal of High Energy Physics 11 (2012) 114. arXiv:1209.2722.

[14] J. Ablinger, J. Blümlein, A. Hussain, S. Klein, C. Schneider, F. Wübbroch, Massless 3-loop ladder diagrams for quarkonic local
[53] O. Schnetz, Graphical functions and single-valued multiple polylogarithms, ArXiv e-prints arXiv:1302.6445.

[54] D. Binosi, L. Theußl, Jaxodraw: A graphical user interface for drawing Feynman diagrams, Comput. Phys. Commun. 161 (1–2) (2004) 76–86. arXiv:hep-ph/0309015 doi:10.1016/j.cpc.2004.05.001
URL http://www.sciencedirect.com/science/article/pii/S0010465504002115

[55] L. Lewin, Structural Properties of Polylogarithms, Vol. 37 of Mathematical Surveys and Monographs, American Mathematical Society, 1991. doi:10.1090/surv/037

[56] D. Kreimer, E. Panzer, Renormalization and Mellin transforms, in: C. Schneider, J. Blümlein (Eds.), Computer Algebra in Quantum Field Theory, Vol. XII of Texts & Monographs in Symbolic Computation, Springer Wien, 2013, pp. 195–223. arXiv:1207.6321 doi:10.1007/978-3-7091-1616-6_8

[57] J. Broedel, O. Schlotterer, S. Stieberger, Polylogarithms, multiple zeta values and superstring amplitudes, Fortsch. Phys. 61 (9) (2013) 812–870. arXiv:1304.7267 doi:10.1002/prop.201300019

[58] C. Bogner, S. Weinzierl, Resolution of singularities for multi-loop integrals, Comput. Phys. Commun. 178 (2008) 596–610. arXiv:0709.4092 doi:10.1016/j.cpc.2007.11.012

[59] V. Del Duca, C. Duhr, V. A. Smirnov, The massless hexagon integral in \( D = 6 \) dimensions, Phys. Lett. B703 (3) (2011) 363–365. arXiv:1104.2781 doi:10.1016/j.physletb.2011.07.079
URL http://www.sciencedirect.com/science/article/pii/S0370269311008987

[60] L. J. Dixon, J. M. Drummond, J. M. Henn, The one-loop six-dimensional hexagon integral and its relation to MHV amplitudes in \( N = 4 \) SYM, JHEP 2011 (6) (2011) 100. arXiv:1104.2787 doi:10.1007/JHEP06(2011)100