Discovering New *L*-Function Relations Using Algebraic Sieving

Hadrien Barral[1], Éric Brier, Rémi Géraud-Stewart[1,2], Arthur Léonard[1], David Naccache[1], Quentin Vermande[1], and Samuel Vivien[1]

[1] Département d'informatique de l'ÉNS, École normale supérieure, CNRS, PSL Research University, Paris, France.
[2] QPSI, Qualcomm Inc., San Diego CA, USA.

Abstract. We report the discovery of new results relating *L*-functions, which typically encode interesting information about mathematical objects, obtained in a *semi-automated* fashion using an algebraic sieving technique.
Algebraic sieving initially comes from cryptanalysis, where it is used to solve factorization, discrete logarithms, or to produce signature forgeries in cryptosystems such as RSA. We repurpose the technique here to provide candidate identities, which can be tested and ultimately formally proven.
A limitation of our technique is the need for human intervention in the post-processing phase, to determine the most general form of conjectured identities, and to provide a proof for them. Nevertheless we report 29 identities that hitherto never appeared in the literature, 9 of which we could completely prove, the remainder being numerically valid over all tested values.
This work complements other instances in the literature where this type of automated symbolic computation has served as a productive step toward theorem proving; it can be extremely helpful in figuring out what it is that one should attempt to prove.

Keywords: *L*-functions, algebraic sieving, conjectures

Introduction

Dirichlet famously introduced *L*-functions, which amongst other tools proved instrumental in establishing results in the distribution of prime numbers in infinite sequences [Dir89]. *L*-functions and their countless generalizations can be constructed for many objects, including characters[3], modular forms, or elliptic curves where they are notably used to formulate the celebrated Birch–Swinnerton–Dyer conjecture [HR15].

This paper focuses on *L*-functions constructed from multiplicative functions (Section 1.1). The Dirichlet sums of such functions features a particularly nice property: they can be expressed as an infinite product over the primes, in a manner reminiscent of Euler's product [Eul37] for Riemann's ζ function (Section 1.3), and are accordingly called *the L-function's Euler product*. At the same time, the Dirichlet sum can yield a known function, such as Riemann's ζ or η functions. This raises the following question:

> *"Can we find* remarkable *relationships between special functions (e.g., ζ, logarithms etc.), or at the very least between Dirichlet sums, through the study of their Euler product?"*

[3] Dirichlet's original motivation

Our approach consists in adapting algebraic sieving algorithms, initially designed to factor composite integers or compute discrete logarithms, to reduce the question of detecting new theorems to the finding of "smoothness" relationships followed by a linear algebraic processing which can be fully automated.

This method is heuristic, but the candidate identities can be tested automatically, and if they succeed, spend some time formally proving them. In doing so, we found many relations relating special functions, or at the very least Dirichlet sums. The simplest of such relations are already well-known — see e.g., [GS08] — but we find several new, non obvious results, which may prove useful in the study of L-functions. Nontrivial examples found by our algorithm are identities such as:

$$\sum_{n=1}^{\infty} \frac{\lambda(n)\tau(n)\sigma_2'(n)}{n^6} = \frac{\zeta(4)^2\zeta(10)\zeta(12)^2}{\zeta(6)^2\zeta(20)} = \frac{154226363\pi^{10}}{12741871041900},$$

where the functions λ, τ, σ' are given hereafter.

1 Preliminaries

Notations. We denote by $\mathbb{P}$ the set of all prime numbers and $\mathbb{N}^*$ is the set of naturals numbers without 0.

1.1 Multiplicative functions

Definition 1 (Multiplicative Function). *A function $f : \mathbb{N}^* \to \mathbb{C}$ is* multiplicative *if for any coprime integers x, y, $f(xy) = f(x)f(y)$. We denote by $\mathcal{M}$ the set of multiplicative functions.*

Example 1. The functions given in Table 1 are well known to be multiplicative and are used throughout this paper. Additional multiplicative functions can be found in [GS08].

1.2 Dirichlet *L*-functions

L-functions were formally defined, and given this name, by Dirichlet [Dir89, pp. 313–342], whose original aim was to prove that there are infinitely many primes in any (primitive) arithmetic progression.

Definition 2 (Dirichlet L-functions). *If $f \in \mathcal{M}$, we define the corresponding formal series called the* L-function *associated to f:*

$$L(f, s) = \sum_{n=1}^{\infty} \frac{f(n)}{n^s}. \tag{1}$$

The well-definedness, convergence properties and analytical continuation of such sums have been extensively studied. For our purposes it is sufficient to say that if f doesn't grow

$\mathbb{1} : n \mapsto 1$
$\epsilon : n \mapsto \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{otherwise} \end{cases}$ *Kronecker* $\delta_{1,n}$
$\mathrm{Id} : n \mapsto n$ *Identity function*
$\varphi : n \mapsto \#\{1 \le i \le n : i \wedge n = 1\}$ *Euler's totient function*
$\sigma_k : n \mapsto \sum_{i \mid n} i^k$ *The* k^{th} *divisor function*
τ : σ_0, the number of divisors
$\tau_k : n \mapsto \#\{(i_1, \ldots, i_k) \in \mathbb{N}^{*k} : \prod_{\ell=1}^{k} i_\ell = n\}$ *The number of ways to express* n *as a product of* k *positive factors* *Note that* $\tau = \tau_2$
$\mu : n \mapsto \begin{cases} (-1)^s & \text{if } n = \prod_{i=1}^{s} p_i \text{ with } p_1, \ldots, p_s \in \mathbb{P} \\ 0 & \text{otherwise} \end{cases}$ *Möbius' function*
$\mu_k : n \mapsto \begin{cases} (-1)^s & \text{if } n = \prod_{i=1}^{s} p_i^k \text{ with } p_1, \ldots, p_s \in \mathbb{P} \\ 0 & \text{otherwise} \end{cases}$, thus $\mu_1 = \mu$ *One of the possible generalizations of Möbius' function*
$J_k : n \mapsto \#\{(a_1, \ldots, a_k) \in \mathbb{N}^{*k} : a_i \le n \text{ and } (a_1, \ldots, a_k, n) \text{ are coprime}\}$ *Jordan's totient function (we have* $J_k(n) = \mu(n) * n^k$*)*
$\lambda : n \mapsto (-1)^r$, where $r = \#\{(p, k) \in \mathbb{P} \times \mathbb{N}^* : p^k \mid n\}$ *Liouville's function*
$\zeta_k : n \mapsto n^k$, where k is non-negative.
$\nu_k : n \mapsto \begin{cases} 1 & \text{if } n \text{ is a } k^{\text{th}} \text{ power} \\ 0 & \text{otherwise} \end{cases}$
$\xi_k : n \mapsto \begin{cases} 1 & \text{if } n \text{ is } k\text{-free} \\ 0 & \text{otherwise} \end{cases}$ *where "*n *is* k*-free" means* $\forall p \in \mathbb{P}, p^k \nmid n$
$\theta : n \mapsto \#\{(a, b) \in \mathbb{N}^{*2} : ab = n \text{ and } \gcd(a, b) = 1\}$
$\sigma'_k : n \mapsto \sum_{d \mid n} \lambda(d) d^k$, where k is non-negative
$\psi_k : n \mapsto \sum_{d \mid n} d^k \left\lvert \mu\left(\frac{n}{d}\right) \right\rvert$ *where* ψ_1 *is known as Dedekind's function*

Table 1. Examples of multiplicative functions.

too fast, the corresponding L-functions is convergent as soon as the real part of s is large enough. In particular,

$$L(\mathbb{1}, s) = \sum_{n=1}^{\infty} \frac{\mathbb{1}(n)}{n^s} = \sum_{n=1}^{\infty} \frac{1}{n^s} = \zeta(s),$$

where $\zeta(s)$ is Riemann's celebrated zeta function [Rie59].

Remark 1 (About convergence). In the rest of this paper, we do not discuss in detail the convergence of L-functions, and assume that the formal manipulations are valid throughout. The result is that some terms in the relations we obtain may be divergent. This is only of consequence if such divergent terms end up in the final identities, at which point they are easily spotted. It is of the nature of a heuristic that it sometimes produces correct outputs through a reasoning that is not valid throughout, and this is why we insist that the identities found by our methods, even if numerically credible, often need an independent proof to become theorems.

1.3 Euler products and Bell series

Let $f \in \mathcal{M}$. Under classical convergence hypotheses, we can write an L-functions as its Euler product:

$$L(f, s) = \prod_{p \in \mathbb{P}} \left(\sum_{k=0}^{\infty} \frac{f(p^k)}{p^{ks}} \right) = \prod_{p \in \mathbb{P}} R_p(f, s).$$

The quantity $R_p(f, s)$ is called the *Bell series* associated to f at p and s [Apo98, p. 42–45].

Definition 3 (Good functions and R-fractions). *Assuming $\exists R(f, s) \in \mathbb{C}(X)$ such that $R_p(f, s) = R(f, s)(p)$ we say that f is* good*, and call R its R-fraction.*

Example 2. φ is good since:

$$\begin{aligned} R_p(\varphi, s) &= \sum_{k=0}^{\infty} \frac{\varphi(p^k)}{p^{ks}} = 1 + \sum_{k=1}^{\infty} \frac{p^k - p^{k-1}}{p^{ks}} \\ &= 1 + \frac{p-1}{p^s} \sum_{k=0}^{\infty} \frac{1}{p^{k(s-1)}} = 1 + \frac{p-1}{p^s - p} \\ &= \frac{p^s - 1}{p^s - p}. \end{aligned}$$

Example 3. $\mathbb{1}$ is good since:

$$R_p(\mathbb{1}, s) = \sum_{k=0}^{\infty} \frac{1}{p^{ks}} = \frac{p^s}{p^s - 1}.$$

The key observation is that for such functions, any multiplicative relation *between R-fractions* gives a relation *between the corresponding L-functions* [Apo98, Theorem 2.25], which was one of the motivations for Bell's introduction of the eponymous series in the 1930s.

Example 4. For s large enough to guarantee convergence:

$$\frac{R(\mathbb{1}, s-1)}{R(\mathbb{1}, s)} = \frac{X^{s-1}}{X^{s-1}-1}\frac{X^s-1}{X^s} = \frac{X^s-1}{X^s-X} = R(\varphi, s).$$

Hence, we recover the well-known relation:

$$L(\varphi, s) = \frac{\zeta(s-1)}{\zeta(s)}.$$

2 Adapting algebraic sieving to *L*-functions

In light of the previous sections' observations, we may seek multiplicative relations between R-fractions. We do this by adapting the algebraic sieving technique to the context of rational fractions.

2.1 Pseudo-linear functions

As a preliminary step, we need to obtain the R-fractions associated to (good) multiplicative functions. It turns out that many interesting multiplicative functions belong to a subset which is more amenable to algorithmic treatment:

Definition 4 (Pseudo-linear function). *$f \in \mathcal{M}$ is* pseudo-linear *if $\exists n \in \mathbb{N}^*$, a column vector $u \in \mathbb{C}(X)^n$ and a matrix $A \in M_n(\mathbb{C}(X))$ such that for every prime p,*

$$f(p^k) = \pi(A^k u)(p).$$

Where π is the projection on the first component. We will represent such a pseudo-linear function as a pair (A, u) and denote by $\mathcal{P}$ the set of pseudo-linear functions.

Example 5. For the function τ (number of divisors) we have the matrix $\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}$ and the vector $\begin{bmatrix}1\\ 1\end{bmatrix}$. We can thus see that $\tau(p^k) = k+1$.

Let $f \in \mathcal{P}$ be represented as (A, u). We have, under invertibility hypotheses:

$$R(f, 0) = \sum_{k=0}^{\infty} \pi(A^k u) = \pi\left(\left(\sum_{k=0}^{\infty} A^k\right) u\right) = \pi((I_n - A)^{-1} u).$$

We know that $R(f, s) = R(f \times \mathrm{Id}^{-s}, 0)$.

2.2 Operations between R-fractions

Multiplication Let $f, g \in \mathcal{P}$, represented respectively as (A, u) and (B, v). We have:

$$\begin{aligned} fg(p^k) &= f(p^k)g(p^k) = \pi(A^k u)\pi(B^k v) \\ &= \pi\left((A^k \otimes B^k)(u \otimes v)\right) \\ &= \pi((A \otimes B)^k (u \otimes v)) \end{aligned}$$

where $\otimes$ is the tensor product. Thus, fg can be represented by $(A \otimes B, u \otimes v)$.

Dirichlet convolution If $f, g \in \mathcal{P}$, we denote by:

$$f * g : n \in \mathbb{N}^* \mapsto \sum_{d \mid n} f(n) g\left(\frac{n}{d}\right) \in \mathbb{C}$$

their *Dirichlet convolution*. It is well-known that the result $f * g$ is again a multiplicative function. Let (A, u) and (B, v) be representations of respectively f and g, we would like to compute a representation of $f * g$.

We have, under inversion hypotheses:

$$\begin{aligned} (f * g)(p^k) &= \sum_{i=0}^{k} f(p^i) g(p^{k-i}) = \sum_{i=0}^{k} \pi(A^i u)\pi(B^{k-i} v) \\ &= \sum_{i=0}^{k} \pi\left((A^i \otimes B^{k-i})(u \otimes v)\right) = \pi\left(\sum_{i=0}^{k} (A^i \otimes B^{k-i})(u \otimes v)\right) \\ &= \pi((A^{k+1} \otimes I - I \otimes B^{k+1})(A \otimes I - I \otimes B)^{-1}(u \otimes v)) \\ &= \pi((A \otimes I)^k u') + \pi((I \otimes B)^k v') \\ &= \pi(((A \otimes I)^k \oplus (I \otimes B)^k)(u' \otimes v')) \\ &= \pi((A \otimes I \oplus I \otimes B)^k (u' \otimes v' \oplus u' \otimes v')) \end{aligned}$$

where

$$\begin{aligned} u' &:= (A \otimes I)(A \otimes I - I \otimes B)^{-1}(u \otimes v) \\ v' &:= -(I \otimes B)(A \otimes I - I \otimes B)^{-1}(u \otimes v). \end{aligned}$$

Thus, $f * g$ can be represented by $(A \otimes I \oplus I \otimes B, u' \otimes v' \oplus u' \otimes v')$.

Remark 2. With this setup, it is easy to generate the rational fractions of many pseudo-linear functions: we can compute the representation of simple known multiplicative functions, then compose these representations using the product and the Dirichlet convolution, and finally compute the rational fractions.

Reduction of representations For performance, we would like, given a representation (A, u) of a pseudo-linear function f, to find a representation (B, v) of f of smaller dimension. This can be done easily by finding a linear relation between the rows of the matrix $(A_1, \ldots, A_d, u)$, then removing one of the rows using this relation.

2.3 Generating Relations

We can now turn to finding multiplicative relations between R-fractions.

Holding space basis. We introduce the following definition:

Definition 5 (Holding space). *A set V of non-zero rational fractions is a* holding space *if it is a finitely generated subgroup for the multiplication.*

From a set $P_1, \ldots, P_k$ of polynomials we can construct a set $\mathcal{B}$ which generates the same holding space, but with the property that polynomials in $\mathcal{B}$ are pairwise coprime. Indeed, we can proceed using the following `Insert` algorithm on $P = P_1, \ldots, P_k$:

Case	**Condition**	**Perform the operation**
Case 0	$P = 1$	`Discard it`
Case 1	$\exists Q \in \mathcal{B}$ such that $Q \mid P$	`Insert`(P/Q)
Case 2	$\exists Q \in \mathcal{B}$ such that $\gcd(Q, P) \neq 1$	$\mathcal{B} \leftarrow \mathcal{B} - Q$ $Q' \leftarrow Q/\gcd(Q, P)$ $\mathcal{B} \leftarrow \mathcal{B} \cup Q'$ `Insert`$(\gcd(Q, P))$ `Insert`$(P/\gcd(Q, P))$
Case 3	otherwise	$\mathcal{B} \leftarrow \mathcal{B} \cup P$

Table 2. `Insert` algorithm

The process is somewhat reminiscent of Buchberger's algorithm [Buc76], that transforms a given set of generators for a polynomial ideal into a Gröbner basis with respect to some monomial order.

We convene that $\mathcal{B}$ is sorted so that the polynomials appearing the most frequently in the P_i appear the first in $\mathcal{B}$. We use this algorithm on the polynomials appearing (as numerator or denominator) in a collection of R-fraction, and call the resulting set $\mathcal{B}$ our *holding space basis*.

Composition matrix. By construction, our R-fractions can be written as a product or ratio of elements in $\mathcal{B} = (b_1, \ldots, b_r)$; furthermore this decomposition is unique up to the order of terms. To each R-fraction we can therefore associate a vector whose i^{th} coefficient is the exponent of b_i in this unique decomposition. Stacking these row vectors together, all our R-fractions give a matrix M.

The multiplication of two R-fractions corresponds to the addition of two matrix rows, and more generally a multiplicative relation between R-fractions corresponds to a linear

combination of the rows of M which gives 0 — in other terms, we are interested in finding elements in the kernel of M.

3 Implementation

In this section we describe data structures and other specific choices made to implement our algorithm. The source code and all tools used for this paper are available under the GNU General Public License (`GPL-3.0-only`) license at https://github.com/CrazySumsTeam/CrazySums.

3.1 Generation

The algorithm starts by generating many R-fractions, from which relations will be sought. First of all, we start by giving constraints to bound the generated L-functions. In other words, we give to the generator the following input:

$$(f_0, \ldots, f_n),\ (a_0, \ldots, a_n),\ (b_0, \ldots, b_n),\ \text{as well as } (\min{}_s, \max{}_s, \max{}_{\text{score}})$$

Afterward, the generator computes all the the L-functions satisfying those constraints. This means for every $(j_i)_{0 \leqslant i < n}$ such that $\forall i, a_i \leqslant j_i \leqslant b_i$, we compute $R(f, s)$ such that:

$$f = \prod_{i=0}^{n-1} f_i^{j_i}, \qquad s = s(f) + k, \text{ such that } \min{}_s \leqslant k \leqslant \max{}_s \qquad \max{}_{\text{score}} \geqslant \sum_{i=0}^{n-1} j_i$$

where $s(f)$ is the minimal integer such that $R(f, s)$ is defined. Note that $\max_{\text{score}}$ is introduced to prevent generating overly-complicated L-functions before simple ones. Not using $\max_{\text{score}}$ drastically increases computation time for a minimal gain in the numbers of interesting relations found.

One important thing to see is that if $f = \prod_i f_i^i$ has already been computed it much faster to compute $f \cdot f_k$ than starting back from the beginning. This pruning where we just remember the last R-fraction computed vastly improved the computation time needed to generate the R-fractions. Consequently, this part wasn't modified to enable multi-threading.

3.2 Multi-threading

To improve performances, some computation steps have been parallelized. Those steps are the decomposition matrix computation and the basis generation.

1. The decomposition matrix computation can be parallelized quite intuitively. As the basis has already been found, it is thus ready-only at this point. Decomposing each polynomial can then be decomposed independently of the others, making the decomposition step an embarrassingly parallel problem. We first create a polynomial queue, on which each worker repeatedly pops a polynomial, decomposes it and stores the result in the decomposition matrix. As the actual decomposition is by far the most time-consuming step, locking the queue with simple mutexes does not create lock-contention.

2. The basic generation step is quite trickier. We implemented this step as an iterative parallel version of the `Insert` algorithm presented in table 2. Each worker needs to modify the basis (either append a polynomial at the end or break an existing polynomial into two smaller ones). Careful use of atomic accesses, mutexes (e.g. using shared mutex locking when possible), as well as mathematical arguments (e.g. when a worker overtakes another one when iterating on the existing basis) were needed to avoid excessive lock contention.

3.3 Polynomials

Polynomials are represented over a finite field $\mathbb{F}_p$ rather than rationals. This introduces the possibility of an error but allows for much faster operation. Errors can be made less likely by increasing p; however in practice, no spurious relation have been found for $p = 997$.

Should spurious relations be found, running our algorithm over a range of different values of p and seeking out recurring relations would filter out most, if not all, erroneous relations.

3.4 Matrices

The composition matrix M is stored as a matrix of rationals (pairs of integers). In principle this matrix holds $n \times m$ coefficients where n is the size of the basis and m the number of polynomials. However it turns out that the matrix is sparse. Hence, rather than using a 2-D arrays the matrix is represented as an array of rows, where a rows is a vector of pairs holding the column number and the coefficient at this position. Thus only non-zero coefficients need to be stored.

This has been implemented with a linear complexity rather than a logarithmic as the matrices contain a huge amount of zeroes. Knowing this, we expected no gain in terms of performances from improving the complexity. Thus we kept the linear complexity.

3.5 Post-processing the results

Once the results have been computed, human intervention is necessary to turn them into mathematical statements. A typical run outputs thousands of relations, which makes this task quite labor-intensive. Indeed our implementation does not infer general symbolic relations, but specific instances of them, and we did seek displaying relations in a human-friendly manner. This human intervention and how we strove to ease it is discussed in more details in appendix B.

4 Results

4.1 New Symbolic Relations Discovered

We give an identifier C-`XX` to the relations found using our approach, that (to the best of our knowledge) were not known before. Known relations, and those which are special cases

of others, were removed from this list. For some of these results we could also directly prove the equality, which is indicated by a ✓in the table (as an example, we provide the proof for C-14 in appendix A).

ID	Relation Discovered	Remarks
✓ C-01	$L(\sigma_n\lvert\mu\rvert, 2n) = \zeta(n)\cdot\zeta(3n)^{-1}$	
✓ C-05	$L\left(f^2\mu, 2n\right) = L\left(f\mu, n\right)\cdot L\left(f\lvert\mu\rvert, n\right)$	for multiplicative f
C-06	$L\left(\theta^2\sigma_n\lvert\mu\rvert, 2n\right) = L\left(\theta\lvert\mu\rvert, n\right)^2$	
C-07	$L\left(\theta^2 J_n\mu, 2n\right) = L\left(\theta\mu, n\right)^2$	
C-09	$L\left(\phi^{n-2}\theta^\ell\lvert\mu\rvert, n\right) = L\left(\phi^{n-2}\theta^\ell J_2, n+2\right)$	part. case of C-10
C-10	$L\left(\phi^i\theta^j J_k^p\lvert\mu\rvert, i+pk\right) = L\left(\phi^i\theta^j J_k^{p+1}, i+(p+1)k\right)$	generalizes C-09
✓ C-11	$L(J_n\mu, 2n) = \zeta(2n)\cdot\zeta(3n)\cdot\zeta(n)^{-1}\cdot\zeta(6n)^{-1}$	
✓ C-13	$L\left(J_{2i}\sigma_{i+k}, 3i+2k\right)\cdot L\left(J_i\sigma_i, 2i+k\right)^{-1} = \zeta(i+2k)\cdot\zeta(2i+2k)^{-1}$	$i\geq 1$, $k\geq 2$
✓ C-14	$L\left(\theta\sigma_n, 2n\right)\cdot L\left(J_n^2, 4n\right)^{-1} = \zeta\left(n\right)^2$	
C-15	$L\left(J_n\sigma_n, 3n\right)\cdot L\left(J_n^2, 4n\right)\cdot L\left(J_n^2\sigma_n^2, 5n\right)^{-1} = \zeta\left(2n\right)\cdot\zeta\left(3n\right)^{-1}$	
C-16	$J_{2n}\mu = J_n\sigma_n\mu$ (Note: ✓ $\forall n, J_{2n}\neq J_n\sigma_n$)	✓ for $n=1$
C-18	$L\left(\tau\sigma_k, n\right) = \zeta(n-k)^2\cdot\zeta(n)^2\cdot\zeta(2n-k)^{-1}$	
C-19	$L\left(\lambda\tau\sigma_k, n\right) = \zeta\left(2n-2k\right)^2\cdot\zeta\left(2n\right)^2\cdot\zeta\left(n-k\right)^{-2}\cdot\zeta\left(n\right)^{-2}\cdot\zeta\left(2n-k\right)^{-1}$	
C-20	$L\left(\tau J_n, 2n\right)\cdot L\left(J_n^2, 3n\right)^{-1} = \zeta(n)$	$n\geq 3$
C-21	$L\left(\tau\theta, n\right)\cdot L\left(\tau J_n^2, 3n\right)^{-1} = \zeta(n)^2$	$n\geq 3$
C-22	$L\left(\lambda\tau_k, n\right) = \zeta(2n)^k\cdot\zeta(n)^{-k}$	✓ for $k=2$
✓ C-23a	$\lambda\nu_{2k} = 1$	trivial
✓ C-23b	$\lambda\nu_{2k+1} = \lambda$	trivial
C-24	$L\left(\tau\sigma'_k, n\right) = \zeta\left(n\right)^2\cdot\zeta\left(2n-2k\right)^2\cdot\zeta\left(2n-k\right)\cdot\zeta\left(n-k\right)^{-2}\cdot\zeta\left(4n-2k\right)^{-1}$	
C-25	$L\left(\lambda\tau\sigma'_k, n\right) = \zeta\left(2n\right)^2\cdot\zeta\left(n-k\right)^2\cdot\zeta\left(2n-k\right)\cdot\zeta\left(n\right)^{-2}\cdot\zeta\left(4n-2k\right)^{-1}$	
✓ C-26	$\xi_k\xi_\ell = \xi_{\min(k,\ell)}$	trivial
C-27	$L\left(\lambda\xi_k, n\right) = \zeta(2n)\cdot\zeta(n)^{-1}\cdot\zeta(kn)^{-1}$	k odd
C-28	$L\left(\lambda\xi_k, n\right) = \zeta(2n)\cdot\zeta(kn)\cdot\zeta(n)^{-1}\cdot\zeta(2kn)^{-1}$	k even
C-29	$L\left(m\mapsto J_k(m^\ell), n\ell\right) = \zeta\left((n-k)\ell\right)\cdot\zeta\left((n-k)\ell+k\right)^{-1}$	
C-30	$L\left(m\mapsto\theta(m^\ell), n\ell\right) = \zeta\left(\ell n\right)^2\cdot\zeta\left(2\ell n\right)^{-1}$	
✓ C-31	$L\left(m\mapsto\xi_k(m^\ell), n\ell\right) = 1$	$k\leq\ell$, trivial
C-32	$L\left(m\mapsto\xi_k(m^\ell), n\ell\right) = \zeta(\ell n)\cdot\zeta\left(\left\lceil\frac{k}{\ell}\right\rceil\ell n\right)^{-1}$	$k>\ell$
C-33	$L\left(\tau\xi_2, n\right) = L\left(\theta\xi_2, n\right)$	
C-34	$L\left(m\mapsto\left(\theta J_k\right)(m^\ell), n\ell\right) = L\left(\theta J_k, n\ell-k(\ell-1)\right)$	

4.2 Already-Known results

Unsurprisingly, amongst the many identities found through our algorithm, some were already known. In the relation catalog [GS08], our algorithm has been able to find the following known symbolic relations:

D-[2-6,9-13,15,18,21,22,24-28,30,37-43,46,47,49-53,55,58]

Remark 3. Relations D-27 and D-51 were found to be erroneous in [GS08], but the original source [Jos86] used by [GS08] is correct and coherent with our results.

Remark 4. Although our algorithm identifies relationships automatically, its output is not in mathematical form and human intervention was necessary to infer the general forms of the relations and formulate the discovered conjectured under a parameterized form. As a result, a (large) part of the program's output has not yet been processed. This task could maybe be automated, at least in part, and several other intriguing results still await to be uncovered.

Remark 5. Obtaining even more relations would be possible, by investing more computational power and/or modifying the implementation to explore different branches, with the same caveat as above that turning the output into mathematical results is a labor-intensive task.

4.3 Performance

It is unknown how many relations exist in the search space. We report on experimental results based on repeated runs, with different configurations to find different types of relations (see section 3.1 about generation). All runs were measured on a 8 cores Intel i7-9700 CPU with 8GB of RAM.

- The default *small* configuration of our tool runs in 1.5 seconds. It generates 1858 R-fractions, leading to a basis of 1517 polynomials. 659 relations are found.
- An extended, *large* configuration setup runs in 20 minutes. It generates 55198 R-fractions, leading to a basis of 52667 polynomials. 5590 relations are found.

We provide in Table 3 the breakdown of the relations found by those runs.

Classification	**Small configuration**	**Large configuration**
Known relations	513	1305
Unclassified relations	19	3923
New relations	127	362
Total	**659**	**5590**

Table 3. Breakdown of the number of relations found by our tool in different configurations

Remark 6. There is no precise sense in which the running time of these algorithms can be analyzed, in part because such an analysis would require having "density" results on the distribution of L-functions (in the usual cryptanalytic context of these methods, it is e.g. the distribution of prime numbers that matters, which is of course much better understood).

4.4 Future work: Towards new functions

The techniques described in this article call for deeper investigations and open new research horizons. We give here a couple research directions that we think could extend the breadth of automated theorem discovery.

The framework of sieving can be used beyond the use we make of it in this paper, to identify further properties of L-functions, and possibly other functions as well. Naturally, the holding space can be extended with more functions, but there is a limit to how many interesting multiplicative functions we can find, and it makes sense to instead extend the kind of relations that we look for.

Indeed, we may let go of the desire that $R_p(f, s)$ be expressed as a polynomial, which extends the realm of relations we find. We can also consider L-functions for which the Bell series are not rational fractions: if it is a product of a rational fraction and some other function, say a logarithm of a rational fraction, we could use algebraic sieving on that other functions.

A further visual optimization would consist in adding to our algorithm known "shortcut relationships" to eliminate before the linear algebra step terms such as:

$$\prod_{p \in \mathbb{P}} \left(\frac{p^{2s}+1}{p^{2s}-1}\right) = \frac{B_{2s}^2 \times (4s)!}{2B_{4s}(2s)!^2}.$$

This would result in avoiding a number of $\zeta(x)$ in the resulting expressions but will not modify fundamentally the discovered relations. Here B_n denotes the absolute value of the n^{th} Bernoulli number.

We also limited ourselves to the convergent setting, but there are conceivably some identities that involve non-convergent functions such as:

$$\lim_{m\to\infty} \frac{m^7}{\log m} \cdot \frac{\sum_{i=1}^m \sigma(i)^4\phi(i)^2 i^{-3}}{\sum_{i=1}^m \sigma(i)\phi(i)} = 1 \quad \text{or} \quad \lim_{m\to\infty} \frac{m\sum_{i=1}^m \sigma(i)^2\phi(i)^2 i^{-3}}{(\sum_{i=1}^m \sigma(i)\phi(i))(\sum_{i=1}^m \phi(i)^2 i^{-4})} = \frac{\zeta(3)}{\zeta(2)}.$$

How to derive such identities in a mathematically consistent (and automated?) way?

Alternatively, we can leave L-functions altogether and consider the automated exploration of relations such as the following: consider the strictly positive roots of the equation $\tan(x) = x^3$ and denote by r_i the sum of the inverses of the roots' $(2i)^{\text{th}}$ powers[4]. Then we found:

$$61r_1 - 35r_1^2 + 5r_1^3 + (30r_1 - 70)r_2 + 40r_3 - 10 = \frac{1}{21}$$

by using algebraic sieving on entire functions, made possible by leveraging Weierstraß' factorization theorem.

Acknowledgements

We would like to thank the anonymous reviewers of the MathCrypt 2021 conference for proofreading an early version of this work and providing useful feedback.

[4] i.e., r_2 is the sum of the inverses of the roots' 4^{th} powers. The numerical values of those constants are $r_1 = 0.189819242600\ldots$,$r_2 = 0.004855766439\ldots$ and $r_3 = 0.000194930492\ldots$

References

Apo98. Tom M. Apostol. *Introduction to Analytic Number Theory*. Springer Science & Business Media, 1998.

Buc76. Bruno Buchberger. Theoretical Basis for the Reduction of Polynomials to Canonical Forms. *ACM SIGSAM Bulletin*, 10(3):19–29, 1976.

Dir89. Johann Peter Gustav Lejeune Dirichlet. *Werke*, volume 1. Leopold Kronecker, Reimer, Berlin, 1889.

Eul37. Leonhard Euler. Variae observationes circa series infinitas. *Commentarii academiae scientiarum imperialis Petropolitanae*, 9(1737):160–188, 1737.

GS08. Henry Gould and Temba Shonhiwa. A Catalog of Interesting Dirichlet Series. *Missouri Journal of Mathematical Sciences*, 20:2–18, 2008.

HR15. Godfrey Harold Hardy and Marcel Riesz. *The General Theory of Dirichlet's Series*. Cambridge University Press, 1915.

Jos86. McCarthy Paul Joseph. *Introduction to arithmetical functions*. Universitext. Springer-Verlag, New York, 1986.

Rie59. Bernhard Riemann. Über die Anzahl der Primzahlen unter einer gegebenen Grösse. *Ges. Math. Werke und Wissenschaftlicher Nachlaß*, 2:145–155, 1859.

A Example proof of a relation

We have not proved all the relations, but we checked some of them by hand. This process could be automated in the future. Let us show how to prove a relation.

For example, we would like to prove (C-14) that:

$$\frac{L\left(\theta\sigma_n, 2n\right)}{L\left(J_n^2, 4n\right)} = \zeta^2\left(n\right)$$

We already proved that:

$$R(\mathbb{1}, n) = \frac{X^n}{X^n - 1}$$

We have:

$$\begin{aligned}
R_p(\theta\sigma_n, 2n) &= \sum_{k=0}^{\infty} \frac{\theta(p^k)\sigma_n(p^k)}{p^{2nk}} = 1 + \sum_{k=1}^{\infty} \frac{2\sum_{i=0}^{k} p^{in}}{p^{2nk}} = 1 + 2\sum_{k=1}^{\infty} \frac{p^{(k+1)n} - 1}{(p^n - 1)p^{2nk}} \\
&= 1 + \frac{2}{p^n - 1} \sum_{k=1}^{\infty} \frac{p^{(k+1)n} - 1}{p^{2nk}} = 1 + \frac{2}{p^n - 1} \sum_{k=1}^{\infty} \frac{1}{p^{(k-1)n}} - \frac{1}{p^{2nk}} \\
&= 1 + \frac{2p^n}{(p^n - 1)^2} - \frac{2}{(p^n - 1)(p^{2n} - 1)} \\
&= \frac{p^{4n} - 2p^n + 1}{(p^n - 1)^2(p^{2n} - 1)}
\end{aligned}$$

Thus:

$$R(\theta\sigma_n, 2n) = \frac{X^{4n} - 2X^n + 1}{(X^n - 1)^2(X^{2n} - 1)}$$

Also:

$$\begin{aligned} R_p(J_n^2, 4n) &= \sum_{k=0}^{\infty} \frac{J_n^2(p^k)}{p^{4nk}} = 1 + \sum_{k=1}^{\infty} \frac{(p^{nk} - p^{n(k-1)})^2}{p^{4nk}} \\ &= 1 + (1 - 2p^{-n} + p^{-2n}) \sum_{k=1}^{\infty} \frac{1}{p^{2nk}} \\ &= 1 + \frac{p^{2n} - 2p^n + 1}{p^{4n} - p^{2n}} \\ &= \frac{p^{4n} - 2p^n + 1}{p^{4n} - p^{2n}} \end{aligned}$$

Thus:

$$R(J_n^2, 4n) = \frac{X^{4n} - 2X^n + 1}{X^{4n} - X^{2n}}$$

Now:

$$\begin{aligned} \frac{R(\theta\sigma_n, 2n)}{R(J_n^2, 4n)} &= \frac{X^{4n} - 2X^n + 1}{(X^n - 1)^2(X^{2n} - 1)} \frac{X^{4n} - X^{2n}}{X^{4n} - 2X^n + 1} = \frac{X^{2n}}{(X^n - 1)^2} \\ &= R(\mathbb{1}, n)^2 \end{aligned}$$

Thus, under convergence hypotheses, we have the expected relation.

B From relations instances to general symbolic formulae

As mentioned in subsection 3.5, human intervention is necessary to turn the output of our tool into general symbolic mathematical statements.

A significant software engineering work was needed to automatically sort, classify and display relations to simplify human intervention. As a rough approximation, half of the source code is dedicated to this task.

B.1 Pretty-printer

We hence implemented a pretty-printer who generates two different output formats (both methods are implemented together to prevent code duplication):

- The first format uses Unicode characters for mathematical symbols (such as σ) and just dumps the formulae in the shell. This enables the user to see directly the result once computed and analyze them on the flight.
- The second format is LaTeX code.

B.2 Formula classifier

Moreover, a formula classifier has been implemented to isolate known finds from unknown ones. For example, $L\left(\lambda\sigma'_k, s\right) = \zeta\left(2s\right) \cdot \zeta\left(s-k\right) \cdot \zeta\left(s\right)^{-1}$ (D-25) is classified with the following definition:

```
vector<pair<HFormula, Rational>> d_25 {
  {HFormulaLFunction(HFormulaProduct(
    HFormulaLeaf(FormulaNode::LEAF_LIOUVILLE),
    HFormulaLeaf(FormulaNode::LEAF_SIGMA_PRIME,
      (FormulaNode::LeafExtraArg){.k = FormulaNode::Symbolic("k"), .l = 0})
   ), FormulaNode::Symbolic("s")), Rational(1)},
  {HFormulaLFunction(HFormulaOne(), FormulaNode::Symbolic("s")), Rational(1)},
  {HFormulaLFunction(HFormulaOne(), FormulaNode::Symbolic("s+-k")), Rational(-1)},
  {HFormulaLFunction(HFormulaOne(), FormulaNode::Symbolic("2*s")), Rational(-1)},
};
```

B.3 Practical example

As an example[5], let us generate formulae of the form $L\left(\lambda\tau^i\sigma'^j_2, s\right), 0 \leq i \leq 1, 0 \leq i \leq 2, s = s(f) + k, 0 \leq k \leq 2$. Having populated our tools with known formulae from [GS08], the output if the following:

```
[D-25] L(λ σ', 3) = (ζ(2) ζ(6)) / ζ(3)
[D-25] L(λ σ', 4) = (ζ(3) ζ(8)) / ζ(4)
[D-25] L(λ σ', 5) = (ζ(4) ζ(10)) / ζ(5)
[D-42] L(λ σ'^2, 4) = (ζ(3)^2 ζ(8)) / (ζ(2) ζ(6))
[D-42] L(λ σ'^2, 5) = (ζ(4)^2 ζ(6) ζ(10)) / (ζ(3) ζ(5) ζ(8))
[D-42] L(λ σ'^2, 6) = (ζ(5)^2 ζ(8) ζ(12)) / (ζ(4) ζ(6) ζ(10))
[D-53] L(λ, 2) = ζ(4) / ζ(2)
[D-53] L(λ, 3) = ζ(6) / ζ(3)
[D-53] L(λ, 4) = ζ(8) / ζ(4)
[!!!!] L(λ τ, 4) = ζ(8)^2 / ζ(4)^2
[!!!!] L(λ τ, 5) = ζ(10)^2 / ζ(5)^2
[!!!!] L(λ τ, 6) = ζ(12)^2 / ζ(6)^2
[!!!!] L(λ τ σ', 5) = (ζ(4)^2 ζ(9) ζ(10)^2) / (ζ(5)^2 ζ(18))
[!!!!] L(λ τ σ', 6) = (ζ(5)^2 ζ(11) ζ(12)^2) / (ζ(6)^2 ζ(22))
[!!!!] L(λ τ σ', 7) = (ζ(6)^2 ζ(13) ζ(14)^2) / (ζ(7)^2 ζ(26))
```

The human step is then to conjecture that the 3 last lines form a pattern, namely $L\left(\lambda\sigma'_k, s\right) = \zeta\left(2s\right) \cdot \zeta\left(s-k\right) \cdot \zeta\left(s\right)^{-1}$. To strengthen evidence of this conjecture, we re-run the tool with a tighter formulae generation, but with a larger $\max_s$.

```
[!!!!] L(λ τ σ', 5) = (ζ(4)^2 ζ(9) ζ(10)^2) / (ζ(5)^2 ζ(18))
[!!!!] L(λ τ σ', 6) = (ζ(5)^2 ζ(11) ζ(12)^2) / (ζ(6)^2 ζ(22))
[!!!!] L(λ τ σ', 7) = (ζ(6)^2 ζ(13) ζ(14)^2) / (ζ(7)^2 ζ(26))
[!!!!] L(λ τ σ', 8) = (ζ(7)^2 ζ(15) ζ(16)^2) / (ζ(8)^2 ζ(30))
```

[5] This example is artificial, but it was necessary to reduce the size of the tool output to ease the reader comprehension. Typical runs have 100 to 10000 output lines.

```
[!!!!] L(λ τ σ', 9) = (ζ(8)^2 ζ(17) ζ(18)^2) / (ζ(9)^2 ζ(34))
[!!!!] L(λ τ σ', 10) = (ζ(9)^2 ζ(19) ζ(20)^2) / (ζ(10)^2 ζ(38))
[!!!!] L(λ τ σ', 11) = (ζ(10)^2 ζ(21) ζ(22)^2) / (ζ(11)^2 ζ(42))
[!!!!] L(λ τ σ', 12) = (ζ(11)^2 ζ(23) ζ(24)^2) / (ζ(12)^2 ζ(46))
[!!!!] L(λ τ σ', 13) = (ζ(12)^2 ζ(25) ζ(26)^2) / (ζ(13)^2 ζ(50))
```

Now having sufficient evidence, we give a name to this conjecture (here C-25), and process to add it to our classifier.

The new output of our tool allows us to focus on the remaining unknown finds:

```
[D-25] L(λ σ', 3) = (ζ(2) ζ(6)) / ζ(3)
[D-25] L(λ σ', 4) = (ζ(3) ζ(8)) / ζ(4)
[D-25] L(λ σ', 5) = (ζ(4) ζ(10)) / ζ(5)
[D-42] L(λ σ'^2, 4) = (ζ(3)^2 ζ(8)) / (ζ(2) ζ(6))
[D-42] L(λ σ'^2, 5) = (ζ(4)^2 ζ(6) ζ(10)) / (ζ(3) ζ(5) ζ(8))
[D-42] L(λ σ'^2, 6) = (ζ(5)^2 ζ(8) ζ(12)) / (ζ(4) ζ(6) ζ(10))
[D-53] L(λ, 2) = ζ(4) / ζ(2)
[D-53] L(λ, 3) = ζ(6) / ζ(3)
[D-53] L(λ, 4) = ζ(8) / ζ(4)
[C-25] L(λ τ σ', 5) = (ζ(4)^2 ζ(9) ζ(10)^2) / (ζ(5)^2 ζ(18))
[C-25] L(λ τ σ', 6) = (ζ(5)^2 ζ(11) ζ(12)^2) / (ζ(6)^2 ζ(22))
[C-25] L(λ τ σ', 7) = (ζ(6)^2 ζ(13) ζ(14)^2) / (ζ(7)^2 ζ(26))
[!!!!] L(λ τ, 4) = ζ(8)^2 / ζ(4)^2
[!!!!] L(λ τ, 5) = ζ(10)^2 / ζ(5)^2
[!!!!] L(λ τ, 6) = ζ(12)^2 / ζ(6)^2
```

Here, the remaining unknown finds are instances of C-22.