Logical Methods in Computer Science
Volume 15, Issue 4, 2019, pp. 3:1–3:21
https://lmcs.episciences.org/

Submitted Nov. 28, 2018
Published Oct. 17, 2019

# ON THE ENUMERATION OF CLOSURES AND ENVIRONMENTS WITH AN APPLICATION TO RANDOM GENERATION

MACIEJ BENDKOWSKI AND PIERRE LESCANNE

Jagiellonian University, Faculty of Mathematics and Computer Science, Theoretical Computer Science Department, ul. Prof. Łojasiewicza 6, 30–348 Kraków, Poland
*e-mail address*: maciej.bendkowski@tcs.uj.edu.pl

University of Lyon, École normale supérieure de Lyon, LIP (UMR 5668 CNRS ENS Lyon UCBL), 46 allée d'Italie, 69364 Lyon, France
*e-mail address*: pierre.lescanne@ens-lyon.fr

Abstract. Environments and closures are two of the main ingredients of evaluation in lambda-calculus. A closure is a pair consisting of a lambda-term and an environment, whereas an environment is a list of lambda-terms assigned to free variables. In this paper we investigate some dynamic aspects of evaluation in lambda-calculus considering the quantitative, combinatorial properties of environments and closures. Focusing on two classes of environments and closures, namely the so-called plain and closed ones, we consider the problem of their asymptotic counting and effective random generation. We provide an asymptotic approximation of the number of both plain environments and closures of size $n$. Using the associated generating functions, we construct effective samplers for both classes of combinatorial structures. Finally, we discuss the related problem of asymptotic counting and random generation of closed environments and closures.

## 1. Introduction

Though, traditionally, computational complexity is investigated in the context of Turing machines since their initial development, evaluation complexity in various term rewriting systems, such as $\lambda$-calculus or combinatory logic, attracts increasing attention only quite recently. For instance, let us mention the worst-case analysis of evaluation, based on the invariance of unitary cost models [29, 3, 1] or transformation techniques proving termination of term rewriting systems [2].

Much like in classic computational complexity, the corresponding average-case analysis of evaluation in term rewriting systems follows a different, more combinatorial and quantitative approach, compared to its worst-case variant. In [13, 14] Choppy, Kaplan and Soria propose an average-case complexity analysis of normalisation in a general class of term rewriting

*Key words and phrases:* lambda-calculus, combinatorics, functional programming, mathematical analysis, complexity.

Maciej Bendkowski was partially supported within the Polish National Science Center grant 2016/21/N/ST6/01032.


DOI:10.23638/LMCS-15(4:3)2019

systems using generating functions, in particular techniques from analytic combinatorics [22]. Following a somewhat similar path, Bendkwoski, Grygiel and Zaionc investigated later the asymptotic properties of normal-order reduction in combinatory logic, in particular the normalisation cost of large random combinators [9, 5]. Alas, normalisation in $\lambda$-calculus has not yet been studied in such a combinatorial context. Nonetheless, static, quantitative properties of $\lambda$-terms, form an active stream of recent research. Let us mention, non-exhaustively, investigations into the asymptotic properties of large random $\lambda$-terms [18, 8] or their effective counting and random generation ensuring a uniform distribution among terms with equal size [11, 26, 25, 12].

In the current paper, we take a step towards the average-case analysis of reduction complexity in $\lambda$-calculus. Specifically, we offer a quantitative analysis of environments and closures — two types of structures frequently present at the core of abstract machines modelling $\lambda$-term evaluation, such as for instance the Krivine or U- machine [16, 31], presented in Section 4. In Section 3 we discuss the combinatorial representation of environments and closures, in particular the associated de Bruijn notation. In Section 5 we list the analytic combinatorics tools required for our analysis and we show in Section 7 how they can be used for random generation. In Section 6 and Section 8 we conduct our quantitative investigation into so-called plain and closed environments and closures, respectively, subsequently concluding the paper in Section 9.

## 2. A combinatoric approach to higher order rewriting systems

As said in the introduction, viewing the $\lambda$-calculus from the perspective of counting is new, especially in the scientific community of structures for computation and deduction and requires motivation to be detailed.

First, clearly a new perspective on $\lambda$-calculus enlightens the semantics and opens new directions, especially by adding a touch of efficiency and a discussion on how the size of structures with binders (like $\lambda$-terms) can be measured. However, despite advanced mathematical techniques are used, the goal is more practical and connected to operational semantics and implementation. Counting allows assigning a precise measure on how a specific algorithm performs. In [27][1] Knuth calls analysis of Type A an *analysis of a particular algorithm* and shows how important it is in computer science. He adds (p. 3): "Complexity analysis provides an interesting way to sharpen our tools for the more routine problems we face from day to day."

Furthermore, a notion of probabilistic distribution as used in the average-case analysis of algorithms, after Sedgewick and Flajolet [39], is deduced. In particular a notion of uniform distribution is inferred in order to evaluate the *average case efficiency* of algorithms w.r.t. this distribution. In this paper, the algorithms the authors have in mind are the several reduction machines for the $\lambda$-calculus, especially the Krivine machine and the U-machine, for which analyses of Type A and more specifically average case analyses are expected to be built. Another application is *random generation* of terms and several kinds of logical models for computation as used for instance in QuickCheck [15]. A fully and mathematically justified random generator can only be built using the kind of tools developed in this paper.

But average case analysis based on uniform distribution is not the only one. The so-called *smoothed analysis of algorithms* [40] is another family of tools which is based on

[1]This paper is part of the book "Selected Papers on Analysis of Algorithms" [28] dedicated to Professor N. G. de Bruijn.

measures of size. Here the distribution is no more uniform and this method has promising applications, hopefully in structures for computation.

## 3. Environments and closures

In this section we outline the de Bruijn notation and related concepts deriving from $\lambda$-calculus variants with explicit substitutions used in the subsequent sections.

3.1. **De Bruijn notation.** Though the classic variable notation for $\lambda$-terms is elegant and concise, it poses considerable implementation issues, especially in the context of substitution resolution and potential name clashes. In order to accommodate these problems, de Bruijn proposed an alternative name-free notation for $\lambda$-terms [19]. In this notation, each variable $x$ is replaced by an appropriate non-negative integer $\underline{n}$ (so-called *index*) intended to encode the distance between $x$ and its binding abstraction. Specifically, if $x$ is bound to the $(n+1)$st abstraction on its unique path to the term root in the associated $\lambda$-tree, then $x$ is replaced by the index $\underline{n}$. In this manner, each closed $\lambda$-term in the classic variable notation is representable in the de Bruijn notation.

**Example 3.1.** Consider the $\lambda$-term $T = (\lambda xyzu.x(\lambda yx.y))\ (\lambda z.(\lambda u.u)z)$. Figure 1 depicts three different representations of $T$ as tree-like structures. The first one uses explicit variables, the second one uses back pointers to represent the bound variables, whereas the third one uses De Bruijn indices.

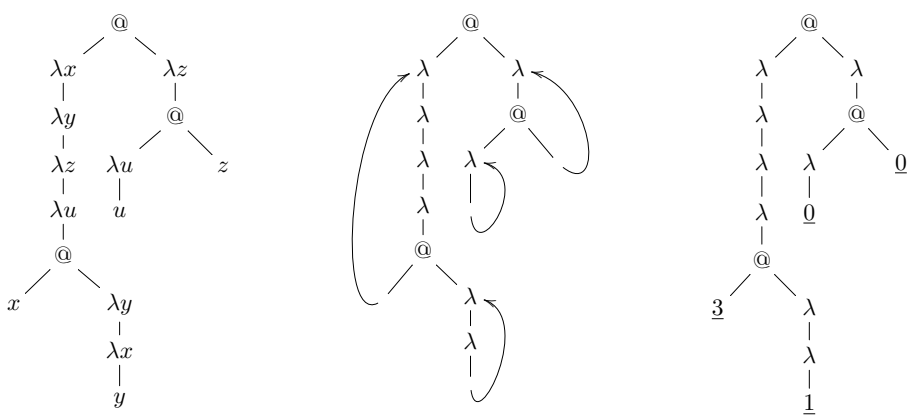

Figure 1: Three representations of the $\lambda$-term $T = (\lambda xyzu.x(\lambda yx.y))\ (\lambda z.(\lambda u.u)z)$.

In order to represent free occurrences of variables, one uses indices of values exceeding the number of abstractions crossed on respective paths to the term root. For instance, $\lambda x.yz$ can be represented as $\lambda\underline{12}$ since $\underline{1}$ and $\underline{2}$ correspond to two different variable occurrences.

Recall that in the classic variable notation a $\lambda$-term $M$ is said to be *closed* if each of its variables is bound. In the de Bruijn notation, it means that for each index occurrence $\underline{n}$ in $M$ one finds at least $n+1$ abstractions on the unique path from $\underline{n}$ to the term root of $M$. If a $\lambda$-term is not closed, it is said to be *open*. If heading $M$ with $m$ abstractions turns it into a closed $\lambda$-term, then $M$ is said to be *$m$-open*. In particular, closed $\lambda$-terms are 0-open.

**Example 3.2.** Note that $\lambda\lambda\lambda\lambda(\underline{3}(\lambda\lambda\underline{1}))\ (\lambda(\lambda\underline{0})\underline{0})$, actually the $T$ of Example 3.1 in De Bruijn notation, is closed. The $\lambda$-term $\underline{3}(\lambda\lambda\underline{1})$ is 4-open, however it is not 3-open. Indeed, $\lambda\lambda\lambda(\underline{3}(\lambda\lambda\underline{1}))$ is 1-open instead of being closed. Similarly, $\lambda(\underline{3}(\lambda\lambda\underline{1}))$ is 3-open, however it is not 2-open.

**Example 3.3.** Consider, on Figure 2, the term $\mathsf{S}\,\mathsf{K}$ and its two direct contractions

$$(\lambda\lambda\lambda\,\underline{2}\,\underline{0}\,(\underline{1}\,\underline{0}))\,(\lambda\lambda\underline{1}) \to \lambda\lambda((\lambda\lambda\underline{1})\,\underline{0}\,(\underline{1}\,\underline{0})) \to \lambda\lambda((\lambda\underline{1})\,(\underline{1}\,\underline{0}),$$

or, in notation with explicit names

$$(\lambda x.\lambda y.\lambda z.xz(yz))\,(\lambda x.\lambda y.x) \to \lambda y.\lambda z.(\lambda x.\lambda y.x)z(yz) \to \lambda y.\lambda z.(\lambda y.z)(yz).$$

It shows how $\beta$-contraction works in De Bruijn notation (cf. the next subsection). Moreover, it shows in $\lambda\lambda(\lambda\underline{1}\,(\lambda\underline{1}\,\underline{0})$ that the same variable namely $z$ may be associated with two De Bruijn indices, namely $\underline{1}$ and $\underline{0}$ and that the same De Bruijn index namely $\underline{1}$ may be associated with two variables namely $y$ and $z$. In the de Bruijn notation the value of an index associated with a variable depends of the context.

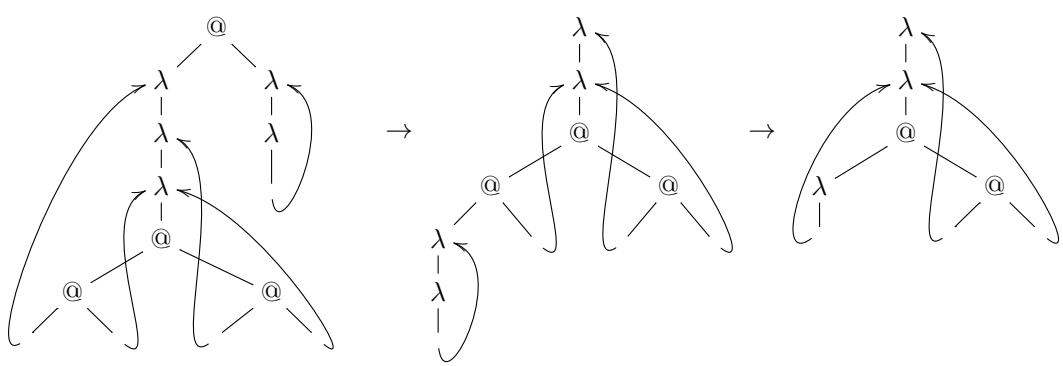

Figure 2: The term $\mathsf{S}\,\mathsf{K}$ and two contractions.

Certainly, the set $\mathcal{L}_m$ of $m$-open terms is a subset of the set of $(m+1)$-open terms. In other words, if $M$ is $m$-open, it is also $(m+1)$-open. The set of all $\lambda$-terms is called the set of *plain* terms. It is the union of the sets of $m$-open terms and is denoted as $\mathcal{L}_\infty$. Hence,

$$\mathcal{L}_0 \subseteq \mathcal{L}_1 \subseteq \cdots \subseteq \mathcal{L}_m \subseteq \mathcal{L}_{m+1} \cdots \subseteq \bigcup_{i=0}^{\infty} \mathcal{L}_i = \mathcal{L}_\infty\,. \tag{3.1}$$

Let us note that de Bruijn's name-free representation of $\lambda$-terms exhibits an important combinatorial benefit. Specifically, each $\lambda$-term in the de Bruijn notation represents an entire $\alpha$-equivalence class of $\lambda$-terms in the classical variable notation. Indeed, two variable occurrences bound by the same abstraction are assigned the same de Bruijn index. In consequence, counting $\lambda$-terms in the de Bruijn notation we are, in fact, counting entire $\alpha$-equivalence classes instead of their inhabitants.

3.2. **Closures and $\beta$-reduction.** Recall that the main rewriting rule of $\lambda$-calculus is *$\beta$-reduction*, see, e.g. [17]:

$$(\beta) \quad (\lambda M)\ N \quad \to \quad M\{\underline{0} \leftarrow N\} \tag{3.2}$$

where the operation $\{\underline{n} \leftarrow M\}$, i.e. substitution of $\lambda$-terms for de Bruijn indices, is defined inductively as follows:

$$\begin{aligned} (M\ N)\{\underline{n} \leftarrow P\} &= M\{\underline{n} \leftarrow P\}\ N\{\underline{n} \leftarrow P\} \\ (\lambda M)\{\underline{n} \leftarrow P\} &= \lambda(M\{\underline{(n+1)} \leftarrow P\}) \\ \underline{m}\{\underline{n} \leftarrow P\} &= \begin{cases} \underline{m-1} & \text{if } m > n \\ \tau_0^n(P) & \text{if } m = n \\ \underline{m} & \text{if } m < n\,. \end{cases} \end{aligned} \tag{3.3}$$

The first rule distributes the substitution in an application, the second rule pushes a substitution under an abstraction, and the third rule dictates how a substitution acts when the term is an index. Finally, $\tau_0^n(P)$ tells how to update the indices of a term which is substituted for an index. The operation $\tau_i^n(M)$ is defined by induction on $M$ as follows:

$$\begin{aligned} \tau_i^n(M\ N) &= \tau_i^n(M)\ \tau_i^n(N) \\ \tau_i^n(\lambda M) &= \lambda(\tau_{i+1}^n(M)) \\ \tau_i^n(\underline{m}) &= \begin{cases} \underline{m+n-1} & \text{if } m > i \\ \underline{m} & \text{if } m \leq i\,. \end{cases} \end{aligned} \tag{3.4}$$

A $\lambda$-term in the form of $(\lambda M)\ N$ is called a *$\beta$-redex* (or simply a *redex*). Lambda terms not containing $\beta$-redexes as subterms, are called ($\beta$-)normal forms. The computational process of rewriting (reducing) a $\lambda$-term to its $\beta$-normal form by successive elimination of $\beta$-redexes is called *normalisation*. There exists an abundant literature on normalisation in $\lambda$-calculus; let us mention, not exhaustively [30, 37, 33, 16, 34].

The central concepts present of formalisms dealing with normalisation in $\lambda$-calculus are environments and closures. An *environment* is a list of not yet evaluated closed terms meant to be assigned to indices $\underline{0}, \underline{1}, \underline{2}, \ldots, \underline{m-1}$ of an $m$-open $\lambda$-term. As lists, environments have two basic operations (two basic constructors), namely $\square$ for the empty environment and ":" for the *cons* operator, i.e., for the operator that put an item in front of an environment. Those not fully evaluated closed terms are represented by *closures*, where a *closure* is a couple consisting of an $m$-open $\lambda$-term and an environment. For instance, the closure $\langle M, \square\rangle$ consists of the $\lambda$-term $M$ evaluated in the context of an empty environment, denoted as $\square$, and represents simply $M$. The closure $\langle \underline{0}\,\underline{1}, \langle \lambda\lambda\underline{0}, \square\rangle : \langle \lambda\underline{0}, \square\rangle : \square\rangle$ represents the $\lambda$-term $(\underline{0}\,\underline{1})$ evaluated in the context of an environment $\langle \lambda\lambda\underline{0}, \square\rangle : \langle \lambda\underline{0}, \square\rangle : \square$. Here, intuitively, the index $\underline{0}$ receives the value $\lambda\lambda\underline{0}$ whereas the index $\underline{1}$ is assigned to $\lambda\underline{0}$. Finally, $\lambda\lambda\underline{0}$ is applied to $\lambda\underline{0}$. And so, reducing the closure $\langle \underline{0}\,\underline{1}, \langle \lambda\lambda\underline{0}, \square\rangle : \langle \lambda\underline{0}, \square\rangle : \square\rangle$, for instance using a Krivine abstract machine [16] (see Section 4.1), we obtain $\lambda\underline{0}$.

Let us notice that following the outlined description of environments and closures, we can provide a formal combinatorial specification for both using the following mutually recursive definitions:

$$\begin{aligned} \mathcal{C}los &::= \langle \Lambda, \mathcal{E}nv\rangle \\ \mathcal{E}nv &::= \square \mid \mathcal{C}los : \mathcal{E}nv \end{aligned} \tag{3.5}$$

In the above specification, $\Lambda$ denotes the set of all plain $\lambda$-terms. Moreover, we introduce two binary operators "$\langle \_, \_ \rangle$", i.e. the *coupling* operator, and ":", i.e. the *cons* operator, heading its left-hand side on the right-hand list. When applied to a $\lambda$-term and an environment, the coupling operator constructs a new closure. In other words, a *closure* is a couple of a $\lambda$-term and an environment whereas an environment is a list of closures, representing a list of assignments to free occurrences of de Bruijn indices.

Such a combinatorial specification for closures and environments plays an important rôle as it allows us to investigate, using methods of analytic combinatorics, the quantitative properties of both closures and environments.

## 4. Closures and abstract machines

Closures are one of the main ingredients of abstract machines performing reduction in $\lambda$-calculus. In the current section, we briefly mention two such machines and discuss how closures and environments relate to the evaluation dynamics of $\lambda$-terms.

4.1. **The Krivine machine.** The presentation of the Krivine machine we give here can be found in Curien's book [16, p. 66]. The state of the machine is a non-empty environment. Its transitions are:

$$\begin{array}{lcll}
\langle M\,N, e\rangle : e' & \to & \langle M, e\rangle : \langle N, e\rangle : e' & (App) \\
\langle \lambda M, e\rangle : \langle N, f\rangle : e' & \to & \langle M, \langle N, f\rangle : e\rangle : e' & (Abs) \\
\langle \underline{0}, \langle M, f\rangle : e\rangle : e' & \to & \langle M, f\rangle : e' & (Zero) \\
\langle \underline{n+1}, \langle M, f\rangle : e\rangle : e' & \to & \langle \underline{n}, e\rangle : e' & (Succ)
\end{array}$$

Interestingly, it is possible to optimise the above transition rules by merging the rules $(Zero)$ and $(Succ)$ into a single rule $(Fetch)$ given as

$$\langle \underline{i}, \langle M_0, f_0\rangle : \ldots : \langle M_i, f_i\rangle : \ldots\rangle : e' \quad \to \quad \langle M_i, f_i\rangle : e' \qquad (Fetch)$$

In words, when interpreting the index $\underline{i}$ we evaluate the $i$th closure of the environment associated with this index. Consequently, a sequence of $i+1$ transitions is replaced by a single one. The above Krivine machine performs head reductions and hence implements a *call-by-name* evaluation strategy. Strong normalisation can be implemented using, e.g. the U-machine.

4.2. **The U-machine.** The U-machine is an abstract machine derived from the calculus of explicit substitution $\lambda\upsilon$, see [31, 32, 4]. First, let us recall that a term of the $\lambda\upsilon$-calculus can contain *explicit substitutions* in form of $M[s]$ where $M$ is a *term* and $s$ a *substitution* as in the following grammar:

$$\begin{array}{rcl}
M, N & ::= & M\,N \mid \lambda M \mid \underline{n} \mid M[s] \\
s & ::= & M/ \mid \Uparrow(s) \mid \uparrow .
\end{array}$$

New operators corresponding to components of explicit substitutions admit the following, intuitive meaning. The *slash* operator / turns a given term into a substitution. Intuitively, it is meant to assign the given term $M$ to the index $\underline{0}$ as in $\underline{0}[M/] \to M$. The *shift* operator $\uparrow$ is a constant whose role is to increment de Bruijn indices, for instance $\underline{n}[\uparrow] \to \underline{n+1}$. Finally *lift*, denoted as $\Uparrow$, is meant to adjust the explicit substitution in the case when it is pushed under an abstraction. For instance, $(\lambda N)[s] \to \lambda(N[\Uparrow(s)])$.

Formally, the way $\beta$-reduction and explicit substitutions work together is given by the following rules of the $\lambda\upsilon$-calculus:

$$\begin{array}{lcll}
(\lambda M)\,N & \to & M[N/] & (Beta) \\
M\,N[s] & \to & M[s]\,N[s] & (App) \\
(\lambda M)[s] & \to & \lambda(M[\Uparrow(s)]) & (Abs) \\
\underline{0}[M/] & \to & M & (FVar) \\
\underline{n+1}[M/] & \to & \underline{n} & (RvAr) \\
\underline{0}[\Uparrow(s)] & \to & \underline{0} & (FVarLift) \\
\underline{n+1}[\Uparrow(s)] & \to & \underline{n}[s][\uparrow] & (RVarLift) \\
\underline{n}[\uparrow] & \to & \underline{n+1} & (VarShift)
\end{array}$$

In the U-machine *environments* are modified so to fit with the features of the $\lambda\upsilon$-calculus, especially with the shift and lift operators. Environments are still lists of *operations* to be performed on variables. These operations, in turn, are pairs in form of $(a, i)$ where $i$ is the number of lifts to be executed before basic actions are performed. Finally, basic actions are of two forms; either they are a *shift* $\uparrow$, or a closure $\langle M, e\rangle$. In other words, closures and environments of the U-machine are changed into:

$$\begin{array}{lcll}
e, f, g & ::= & (a, i)^* & \text{(lists of operations)} \\
a & ::= & \uparrow \mid \langle M, e\rangle & \text{(basic actions)} \\
i & ::= & 0 \mid i+1 & \text{(number of lifts)}
\end{array}$$

A state of the U-machine is a list $\langle M, e\rangle^*$ of pairs where $M$ is a *term* and $e$ is a *list of operations*. Let $(++)$ denote list concatenation and $\mathsf{LiftEnv}$ denote the map incrementing all the second arguments of given list of pairs, i.e. a coordinate-wise function $(a, i) \mapsto (a, i+1)$. Then, the transitions of the U-machine are given as follows:

$$\begin{array}{lcll}
\langle M\,N, f\rangle : e & \to & \langle M, f\rangle : \langle N, f\rangle : e & (APP) \\
\langle \lambda M, f\rangle : \langle N, g\rangle : e & \to & \langle M, \mathsf{LiftEnv}(f) + +[\langle N, g\rangle]\rangle : e & (LBA-BET) \\
\langle \underline{0}, (a, i+1) : f\rangle : e & \to & \langle \underline{0}, f\rangle : e & (FVARLIFT) \\
\langle \underline{n+1}, (a, i+1) : f\rangle : e & \to & \langle \underline{n}, (a, i) : (\uparrow, 0) : f\rangle : e & (RVARLIFT) \\
\langle \underline{0}, (\langle M, f\rangle, 0) : g\rangle : e & \to & \langle M, f + +g\rangle : e & (FVAR) \\
\langle \underline{n+1}, (\langle M, f\rangle, 0) : g\rangle : e & \to & \langle \underline{n}, g\rangle : e & (RVAR) \\
\langle \underline{n}, (\uparrow, 0) : g\rangle : e & \to & \langle \underline{n+1}, g\rangle : e & (VARSHIFT)
\end{array}$$

In the U-machine, two kind of states cannot be further reduced, i.e. states of the form $\langle \lambda N, f\rangle : \square$ (abstractions with empty stacks) and states of the form $\langle \underline{n}, \square\rangle : f$ (indices with nothing in their direct environment). It is possible to further reduce those states using strong normalisation. For that, we introduce the following inference rules which correspond to recursive calls of the U-machine. In there inference rules, $\xrightarrow[U]{}\!\!\!\!\rightarrowtail$ is a relation between list of pairs in form of $\langle M, e\rangle$ and corresponds to the reduction to normal form. Moreover, $\downarrow_{\mathsf{nf}}$ is a deterministic relation between a closure and a term. When we want to designate the result $N$ of the relation $\downarrow_{\mathsf{nf}}$ we write $\mathsf{nf}\langle M, e\rangle$ instead of $\langle M, e\rangle \downarrow_{\mathsf{nf}} N$.

$$\frac{\langle M, e\rangle : \square \xrightarrow[U]{}\!\!\!\!\rightarrowtail \langle \lambda N, f\rangle : \square}{\langle M, e\rangle \quad \downarrow_{\mathsf{nf}} \quad \lambda(\mathsf{nf}\langle N, \mathsf{LiftEnv}(f)\rangle)}$$

$$\frac{\langle M, e\rangle : \square \xrightarrow[U]{}\!\!\!\!\rightarrowtail \langle \underline{n}, \square\rangle : f}{\langle M, e\rangle \quad \downarrow_{\mathsf{nf}} \quad \underline{n}\ (\mathsf{map}\ \mathsf{nf}\ f)}$$

Actually, $\underline{n}$ (map nf $f$) is an abuse of notation for the successive applications of the list map nf $f$ on $\underline{n}$.

## 5. Analytic tools

In the following section we briefly[2] outline the main techniques and notions from the theory of generating functions and singularity analysis. We refer the curious reader to [22, 41, 24] for a thorough introduction.

Let $(f_n)_n$ be a sequence of non-negative integers. Then, the *generating function* $F(z)$ associated with $(f_n)_n$ is the formal power series $F(z) = \sum_{n\geq 0} f_n z^n$. Following standard notational conventions, we use $[z^n]F(z)$ to denote the coefficient standing by $z^n$ in the power series expansion of $F(z)$. Given two sequences $(a_n)_n$ and $(b_n)_n$ we write $a_n \sim b_n$ to denote the fact that both sequences admit the same asymptotic growth order, specifically $\lim_{n\to\infty} \frac{a_n}{b_n} = 1$. Finally, we write $\varphi \doteq c$ when the expression $\varphi$ is approximated by the number $c$.

Suppose that $F(z)$, viewed as a function of a single complex variable $z$, is defined in some region $\Omega$ of the complex plane centred at $z_0 \in \Omega$. Then, if $F(z)$ admits a convergent power series expansion in form of

$$F(z) = \sum_{n\geq 0} f_n (z - z_0)^n \tag{5.1}$$

it is said to be *analytic* at point $z_0$. Moreover, if $F(z)$ is analytic at each point $z \in \Omega$, then $F(z)$ is said to be *analytic in the region* $\Omega$. Suppose that there exists a function $G(z)$ analytic in a region $\Omega^*$ such that $\Omega \cap \Omega^* \neq \emptyset$ and both $F(z)$ and $G(z)$ agree on $\Omega \cap \Omega^*$, i.e. $F|_{\Omega\cap\Omega^*} = G|_{\Omega\cap\Omega^*}$, where $F|_A$ is the restriction of the function $F$ on the region $A$. Then, $G(z)$ is said to be an *analytic continuation* of $F(z)$ onto $\Omega^*$. If $F(z)$ defined in some region $\Omega \setminus \{z_0\}$ has no analytic continuation onto $\Omega$, then $z_0$ is said to be a *singularity* of $F(z)$. When a formal power series $F(z) = \sum_{n\geq 0} f_n z^n$ represents an analytic function in some neighbourhood of the complex plane origin, it becomes possible to link the location and type of singularities corresponding to $F(z)$, in particular so-called *dominating* singularities residing at the respective circle of convergence, with the asymptotic growth rate of its coefficients. This process of *singularity analysis* developed by Flajolet and Odlyzko [21] provides a general and systematic technique for establishing the quantitative aspects of a broad class of combinatorial structures.

While investigating environments and closures, a particular example of algebraic combinatorial structures, the respective generating functions turn out to be algebraic themselves. The following prominent tools provide the essential foundation underlying the process of *algebraic singularity analysis* based on *Newton-Puiseux expansions*, i.e. extensions of power series allowing fractional exponents.

**Theorem 5.1** Newton, Puiseux [22, Theorem VII.7]**.** *Let $F(z)$ be a branch of an algebraic equation $P(z, F(z)) = 0$. Then, in a circular neighbourhood of a singularity $\rho$ slit along a ray emanating from $\rho$, $F(z)$ admits a fractional Newton-Puiseux series expansion that is*

[2] In such a short presentation of a non-trivial theory, many terms, like "branch", "Newton-Puiseux series", "locally convergent" etc. are not defined. They are defined in the references [22, 41, 24].

*locally convergent and of the form*

$$F(z) = \sum_{k \geq k_0} c_k (z-\rho)^{k/\kappa} \tag{5.2}$$

*where* $k_0 \in \mathbb{Z}$ *and* $\kappa \geq 1$.

Let $F(z)$ be analytic at the origin. Note that $[z^n]F(z) = \rho^{-n}[z^n]F(\rho z)$. In consequence, following a proper rescaling we can focus on the type of singularities of $F(z)$ on the unit circle. The standard function scale provides then the asymptotic expansion of $[z^n]F(z)$.

**Theorem 5.2** Standard function scale [22, Theorem VI.1]**.** *Let* $\alpha \in \mathbb{C} \setminus \mathbb{Z}_{\leq 0}$. *Then,* $F(z) = (1-z)^{-\alpha}$ *admits for large* $n$ *a complete asymptotic expansion in form of*

$$[z^n]F(z) = \frac{n^{\alpha-1}}{\Gamma(\alpha)} \left(1 + \frac{\alpha(\alpha-1)}{2n} + \frac{\alpha(\alpha-1)(\alpha-2)(3\alpha-1)}{24n^2} + O\left(\frac{1}{n^3}\right)\right) \tag{5.3}$$

*where* $\Gamma\colon \mathbb{C} \setminus \mathbb{Z}_{\leq 0} \to \mathbb{C}$ *is the Euler Gamma function defined as*

$$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx \qquad \text{for } \Re(z) > 0 \tag{5.4}$$

*and by analytic continuation on all its domain.*

Given an analytic generating function $F(z)$ implicitly defined as a branch of an algebraic function satisfying $P(z, F(z)) = 0$, our task of establishing the asymptotic expansion of the corresponding sequence $([z^n]F(z))_n$ reduces to locating and studying the (dominating) singularities of $F(z)$. For generating functions analytic at the complex plane origin, this quest simplifies even further due to the following classic result.

**Theorem 5.3** Pringsheim [22, Theorem IV.6]**.** *If* $F(z)$ *is representable at the origin by a series expansion that has non-negative coefficients and radius of convergence* $R$*, then the point* $z = R$ *is a singularity of* $F(z)$.

We can therefore focus on the real line while searching for respective singularities. Since $\sqrt{z}$ cannot be unambiguously defined as an analytic function at $z = 0$ we primarily focus on roots of radicand expressions in the closed-form formulae of investigated generating functions.

**Counting $\lambda$-terms.** Let us outline the main quantitative results concerning $\lambda$-terms in the de Bruijn notation, see [7, 8, 25]. In this combinatorial model, indices are represented in a unary encoding using the successor operator $\mathsf{S}$ and 0. In the so-called *natural* size notion [8], assumed throughout the current paper, the size of $\lambda$-terms is defined recursively as follows:

$$\begin{array}{lcl} |0| & = & 1 \\ |\mathsf{S}\ n| & = & |\underline{n}| = |n|+1 \end{array} \qquad \begin{array}{lcl} |M\,N| & = & |M|+|N|+1 \\ |\lambda M| & = & |M|+1\,. \end{array}$$

And so, for example, $|\lambda\underline{12}| = 7$.

**Remark 5.4.** We briefly remark that different size notions in the de Bruijn representation, alternative to the assumed natural one, are considered in the literature. Among all of them, we choose to consider the above size notion in order to minimise the technical overhead of the overall presentation. Analytic methods employed in the current paper cover a broad range of possible size measures. We refer the curious reader to [26, 12, 25] for a detailed analysis of various size notions in the de Bruijn representation.

Let $l_n$ denote the number of plain $\lambda$-terms of size $n$. Consider the generating function $L_\infty(z) = \sum_{n\geq 0} l_n z^n$. Using symbolic methods, see [22, Part A. Symbolic Methods] we note that $L_\infty(z)$ satisfies

$$L_\infty(z) \;=\; zL_\infty(z) + zL_\infty(z)^2 + D(z) \qquad \text{where} \qquad D(z) = \frac{z}{1-z} = \sum_{n=0}^{\infty} z^{n+1}. \tag{5.5}$$

In words, a $\lambda$-term is either (a) an abstraction followed by another $\lambda$-term, accounting for the first summand, (b) an application of two $\lambda$-terms, accounting for the second summand, or finally, (c) a de Bruijn index which is, in turn, a sequence of successors applied to 0. Solving (5.5) for $L_\infty(z)$ we find that the generating function $L_\infty(z)$, taking into account that the coefficients $l_n$ are positive for all $n$, admits the following closed-form solution:

$$L_\infty(z) = \frac{1 - z - \sqrt{(1-z)^2 - \frac{4z}{1-z}}}{2z}\,. \tag{5.6}$$

The first values of the coefficients of $L_\infty$ are:

1, 3, 10, 40, 181, 884, 4539, 24142, 131821, 734577, 4160626 23881695,138610418, ...

This sequence is **A258973** in the *Online Encyclopedia of Integer Sequences*. In such a form, $L_\infty(z)$ is amenable to the standard techniques of singularity analysis. In consequence we have the following general asymptotic approximation of $l_n$.

**Theorem 5.5** Bendkowski, Grygiel, Lescanne, Zaionc [8]**.** *The sequence* $([z^n]L_\infty(z))_n$ *corresponding to plain $\lambda$-terms of size $n$ admits the following asymptotic approximation:*

$$[z^n]L_\infty(z) \sim C\rho_{L_\infty}^{-n} n^{-3/2} \tag{5.7}$$

*where*

$$\rho_{L_\infty} = \frac{1}{3}\left(\sqrt[3]{26+6\sqrt{33}} - \frac{4\;2^{2/3}}{\sqrt[3]{13+3\sqrt{33}}} - 1\right) \doteq 0.29559 \quad \textit{and} \quad C \doteq 0.60676. \tag{5.8}$$

In the context of evaluation, the arguably most interesting subclass of $\lambda$-terms are closed or, more generally, $m$-open $\lambda$-terms. Recall that an $m$-open $\lambda$-term takes one of the following forms. Either it is (a) an abstraction followed by an $(m+1)$-open $\lambda$-term, or (b) an application of two $m$-open $\lambda$-terms, or finally, (c) one of the indices $\underline{0}, \underline{1}, \ldots, \underline{m-1}$. Such a specification for $m$-open $\lambda$-terms yields the following functional equation defining the associated generating function $L_m(z)$:

$$L_m(z) = zL_{m+1}(z) + zL_m(z)^2 + \frac{1-z^m}{1-z}\,. \tag{5.9}$$

Since $L_m(z)$ depends on $L_{m+1}(z)$, solving (5.9) for $L_m(z)$ one finds that

$$L_m(z) = \frac{1 - \sqrt{1 - 4z^2\left(L_{m+1}(z) + \frac{1-z^m}{1-z}\right)}}{2z}\,. \tag{5.10}$$

For instance, the first coefficients of $L_0(z)$ are

$$0, 0, 1, 1, 3, 6, 17, 41, 116, 313, 895, 2550, 7450, 21881, 65168, \ldots$$

the first coefficients of $L_1(z)$ are

$$0, 1, 1, 3, 5, 15, 34, 98, 258, 743, 2098, 6142, 17988, 53614, 160619, \ldots$$

and the first coefficients of $L_2(z)$ are

$$0, 1, 2, 3, 8, 18, 49, 130, 364, 1032, 2987, 8758, 26000, 77937, 235677, \ldots$$

The presentation of $L_m(z)$ given in (5.9) poses considerable difficulties as $L_m(z)$ depends on $L_{m+1}(z)$ depending itself on $L_{m+2}(z)$, etc. If developed, the formula (5.10) for $L_m(z)$ consists of an infinite number of nested radicals. In consequence, standard analytic combinatorics tools do not provide the asymptotic expansion of $[z^n]L_m(z)$, in particular $[z^n]L_0(z)$ associated with closed $\lambda$-terms. In their recent breakthrough paper, Bodini, Gittenberger and Gołębiewski [12] propose a clever approximation of the infinite system associated with $L_m(z)$ and give the following asymptotic approximation for the number of $m$-open $\lambda$-terms.

**Theorem 5.6** Bodini, Gittenberger and Gołębiewski [12]**.** *The sequence* $([z^n]L_m(z))_n$ *corresponding to* $m$*-open* $\lambda$*-terms of size* $n$ *admits the following asymptotic approximation:*

$$[z^n]L_m(z) \sim C_m \rho_{L_\infty}^{-n} n^{-3/2} \tag{5.11}$$

*where* $\rho_{L_\infty}$ *is the dominant singularity corresponding to plain* $\lambda$*-terms, see* (5.8)*, and* $C_m$ *is a constant, depending solely on* $m$*.*

Let us remark that for closed $\lambda$-terms, the constant $C_0$ lies in between 0.07790995266 and 0.0779099823. In what follows, we use the above Theorem 5.6 in our investigations regarding what we call closed closures.

## 6. Counting plain closures and environments

In this section we start with counting *plain environments and closures*, i.e. members of $\mathcal{E}nv$ and $\mathcal{C}los$, see (3.5). We consider a simple model in which the size of environments and closures is equal to the total number of abstractions, applications and the sum of all the de Bruijn index sizes. Formally, we set

$$|\langle M, e\rangle| = |M| + |e| \qquad |\mathfrak{c} : e| = |\mathfrak{c}| + |e| \qquad |\Box| = 0\,.$$

**Example 6.1.** The following two tables list the first few plain environments and closures.

| size | environments | total |
|---|---|---|
| 0 | $\Box$ | 1 |
| 1 | $\langle \underline{0}, \Box\rangle : \Box$ | 1 |
| 2 | $\langle \underline{0}, \Box\rangle : \langle \underline{0}, \Box\rangle : \Box$<br>$\langle \underline{0}, \langle \underline{0}, \Box\rangle : \Box\rangle : \Box$<br>$\langle \lambda\underline{0}, \Box\rangle : \Box, \quad \langle \underline{1}, \Box\rangle : \Box$ | 4 |

| size | closures | total |
|---|---|---|
| 0 | | 0 |
| 1 | $\langle \underline{0}, \Box\rangle$ | 1 |
| 2 | $\langle \underline{0}, \langle \underline{0}, \Box\rangle\rangle$<br>$\langle \lambda\underline{0}, \Box\rangle \quad \langle \underline{1}, \Box\rangle$ | 3 |

By analogy with the notation $\mathcal{L}_\infty$ for the set of plain $\lambda$-terms, we write $\mathcal{E}_\infty$ and $\mathcal{C}_\infty$ to denote the class of plain environments and closures, respectively. Reformulating (3.5) we can now give a formal specification for both $\mathcal{E}_\infty$ and $\mathcal{C}_\infty$ as follows:

$$\begin{aligned} \mathcal{E}_\infty &= \mathcal{C}_\infty : \mathcal{E}_\infty \ \mid \ \Box \\ \mathcal{C}_\infty &= \langle \mathcal{L}_\infty, \mathcal{E}_\infty \rangle\,. \end{aligned} \tag{6.1}$$

In such a form, both classes $\mathcal{E}_\infty$ and $\mathcal{C}_\infty$ become amenable to the process of singularity analysis. In consequence, we obtain the following asymptotic approximation for the number of plain environments and closures.

**Theorem 6.2.** *The numbers $e_n$ and $c_n$ of plain environments and closures of size $n$, respectively, admit the following asymptotic approximations:*

$$e_n \sim C_e \cdot \rho^{-n} n^{-3/2} \quad \textit{and} \quad c_n \sim C_c \cdot \rho^{-n} n^{-3/2} \tag{6.2}$$

*where*

$$\begin{aligned} C_e &= \frac{\sqrt{\frac{5}{47}\left(109 + 35\sqrt{545}\right)}}{8\sqrt{\pi}} \doteq 0.699997, \\ C_c &= \frac{\sqrt{\frac{10\left(48069\sqrt{5} - 10295\sqrt{109}\right)}{65\sqrt{109} - 301\sqrt{5}}}}{\sqrt{\pi}\left(77 - 3\sqrt{545}\right)} \doteq 0.174999 \end{aligned} \tag{6.3}$$

*and*

$$\rho = \frac{1}{10}\left(25 - \sqrt{545}\right) \doteq 0.165476 \quad \textit{giving} \quad \rho^{-n} \doteq 6.04315^n. \tag{6.4}$$

*Proof.* Consider generating functions $E_\infty(z)$ and $C_\infty(z)$ associated with respective counting sequences, i.e. the sequence $(e_n)_n$ of plain environments of size $n$ and $(c_n)_n$ of plain closures of size $n$. Based on the specification (6.1) for $\mathcal{E}_\infty$ and $\mathcal{C}_\infty$ and the assumed size notion, we can write down the following system of functional equations satisfied by $E_\infty(z)$ and $C_\infty(z)$:

$$\begin{aligned} E_\infty(z) &= C_\infty(z) E_\infty(z) + 1 \\ C_\infty(z) &= L_\infty(z)\, E_\infty(z). \end{aligned} \tag{6.5}$$

Next, we solve (6.5) for $E_\infty(z)$ and $C_\infty(z)$. Though (6.5) has two formal solutions, the following one is the single one yielding analytic generating functions with non-negative coefficients:

$$E_\infty(z) = \frac{1 - \sqrt{1 - 4L_\infty(z)}}{2L_\infty(z)} \quad \text{and} \quad C_\infty(z) = \frac{1}{2}\left(1 - \sqrt{1 - 4L_\infty(z)}\right). \tag{6.6}$$

Since $L_\infty(z) > 0$ for $z \in (0, \rho_{L_\infty})$ there are two potential sources of singularities in (6.6). Specifically, the dominating singularity $\rho_{L_\infty}$ of $L_\infty(z)$, see (5.8), or roots of the radicand expression $1 - 4L_\infty(z)$. Therefore, we have to determine whether we fall into the so-called sub- or super-critical composition schema, see [22, Chapter VI. 9]. Solving $1 - 4L_\infty(z) = 0$ for $z$, we find that it admits a single solution $\rho$ equal to

$$\rho = \frac{1}{10}\left(25 - \sqrt{545}\right) \doteq 0.165476\,. \tag{6.7}$$

Since $\rho < \rho_{L_\infty}$ the outer radicand carries the dominant singularity $\rho$ of both $E_\infty(z)$ and $C_\infty(z)$. We fall therefore directly into the super-critical composition schema and in consequence know that near $\rho$ both $E_\infty(z)$ and $C_\infty(z)$ admit Newton-Puiseux expansions in form of

$$\begin{aligned} E_\infty(z) &= a_{E_\infty} + b_{E_\infty}\sqrt{1 - \frac{z}{\rho}} + O\left(\left|1 - \frac{z}{\rho}\right|\right) \\ &\quad \text{and} \\ C_\infty(z) &= a_{C_\infty} + b_{C_\infty}\sqrt{1 - \frac{z}{\rho}} + O\left(\left|1 - \frac{z}{\rho}\right|\right) \end{aligned} \tag{6.8}$$

with $a_{E_\infty}, a_{C_\infty} > 0$ and $b_{E_\infty}, b_{C_\infty} < 0$. At this point, we can apply the standard function scale, see Theorem 5.2, to the presentation of $E_\infty(z)$ and $C_\infty(z)$ in (6.8) and conclude that

$$[z^n]E_\infty(z) \sim C_{E_\infty}\rho^{-n}n^{-3/2} \quad \text{and} \quad [z^n]C_\infty(z) \sim C_{C_\infty}\rho^{-n}n^{-3/2} \tag{6.9}$$

where $C_{E_\infty} = \dfrac{b_{E_\infty}}{\Gamma(-\frac{1}{2})}$ and $C_{C_\infty} = \dfrac{b_{C_\infty}}{\Gamma(-\frac{1}{2})}$, respectively, with $\Gamma(-\frac{1}{2}) = 2\sqrt{\pi}$. In fact, reformulating (6.6) so to fit the Newton-Puiseux expansion forms (6.8) we find that

$$a_{E_\infty} = 2, \quad b_{E_\infty} = -\frac{1}{4}\sqrt{\frac{5}{47}\left(109 + 35\sqrt{545}\right)} \tag{6.10}$$

and

$$a_{C_\infty} = \frac{1}{2}, \quad b_{C_\infty} = \frac{2\sqrt{\frac{10\left(48069\sqrt{5}-10295\sqrt{109}\right)}{65\sqrt{109}-301\sqrt{5}}}}{3\sqrt{545}-77} \tag{6.11}$$

Numerical approximations of $C_{E_\infty} = \dfrac{b_{E_\infty}}{\Gamma(-\frac{1}{2})}$ and $C_{C_\infty} = \dfrac{b_{C_\infty}}{\Gamma(-\frac{1}{2})}$ yield the declared asymptotic behaviour of $(e_n)_n$ and $(c_n)_n$, see (6.2). □

Let us notice that as both generating functions $E_\infty(z)$ and $C_\infty(z)$ are algebraic, they are also *holonomic* (D-finite), i.e. satisfy differential equations with polynomial (in terms of $z$) coefficients. Using the powerful `gfun` library for `Maple` [38] one can automatically derive appropriate holonomic equations for $E_\infty(z)$ and $C_\infty(z)$, subsequently converting them into linear recurrences for sequences $(e_n)_n$ and $(c_n)_n$.

**Example 6.3.** We restrict the presentation to the linear recurrence for the number of plain environments, omitting for brevity the, likely verbose, respective recurrence for plain closures. Using `gfun` we find that $e_n$ satisfies the recurrence of Figure 3. Despite its appearance, this recurrence is an efficient way of computing $e_n$. Indeed, holonomic specifications for $C_\infty(z)$ and $E_\infty(z)$ allow computing the coefficients $[z^n]C_\infty(z)$ and $[z^n]E_\infty(z)$ using a linear number of arithmetic operations, as opposed to a quadratic number of operations as following their direct combinatorial specification. Let us remark that the involved computations operate on large integers, which have a linear in $n$ space representation. For instance, $e_{1000}$ has about 600 digits. In consequence, single arithmetic operations on such numbers cannot be performed in constant time.

## 7. Random generation of closures and environments

Effective counting methods for various discrete structures are among the most prominent and ubiquitous subjects in combinatorics. Although interesting in their own right, such counting methods (and in particular related algorithms) exhibit important benefits in the context of generating random instances of corresponding combinatorial structures. Let us mention, for instance, the successive use of random $\lambda$-terms used to disprove the correctness of eagerness optimisations of the salient Glasgow Haskell Compiler, see [36].

Given the fact that closures and environments are fundamental data structures used in different abstract machines related to the execution of $\lambda$-terms, random closures and environments can be used to model (in other words simulate) actual data encountered in the execution traces of abstract machines such as the Krivine or U-machines. In this context, random generation of closures and environments provide effective means of testing

$$
\begin{aligned}
&(125\,n^3 - 125\,n)\; e_n + \\
&(-475\,n^3 - 150\,n^2 + 325\,n)\; e_{n+1} + \\
&(-1625\,n^3 - 13650\,n^2 - 29125\,n - 17100)\; e_{n+2} + \\
&(5925\,n^3 + 65550\,n^2 + 204825\,n + 190800)\; e_{n+3} + \\
&(-10950\,n^3 - 149850\,n^2 - 609000\,n - 744300)\; e_{n+4} + \\
&(43599\,n^3 + 638460\,n^2 + 3028701\,n + 4633680)\; e_{n+5} + \\
&(-97781\,n^3 - 1680378\,n^2 - 9481237\,n - 17550960)\; e_{n+6} + \\
&(122749\,n^3 + 2388066\,n^2 + 15211685\,n + 31648968)\; e_{n+7} + \\
&(-184402\,n^3 - 3954630\,n^2 - 27717140\,n - 63149544)\; e_{n+8} + \\
&(280081\,n^3 + 6826380\,n^2 + 54868451\,n + 145130568)\; e_{n+9} + \\
&(-205649\,n^3 - 5654610\,n^2 - 51851989\,n - 158722620)\; e_{n+10} + \\
&(37439\,n^3 + 1339686\,n^2 + 16635271\,n + 70682784)\; e_{n+11} + \\
&(-68686\,n^3 - 3028038\,n^2 - 43616336\,n - 205972920)\; e_{n+12} + \\
&(222029\,n^3 + 9258780\,n^2 + 128417911\,n + 592399800)\; e_{n+13} + \\
&(-241115\,n^3 - 10519830\,n^2 - 152823475\,n - 739190880)\; e_{n+14} + \\
&(134151\,n^3 + 6201222\,n^2 + 95476551\,n + 489605640)\; e_{n+15} + \\
&(-42231\,n^3 - 2067834\,n^2 - 33729375\,n - 183277332)\; e_{n+16} + \\
&(7470\,n^3 + 386418\,n^2 + 6659316\,n + 38233296)\; e_{n+17} + \\
&(-678\,n^3 - 36972\,n^2 - 671670\,n - 4065240)\; e_{n+18} + \\
&(24\,n^3 + 1380\,n^2 + 26436\,n + 168720)\; e_{n+19} = 0.
\end{aligned}
$$

$$
\begin{aligned}
e_0 &= 1, & e_{10} &= 1233816,\\
e_1 &= 1, & e_{11} &= 6558106,\\
e_2 &= 4, & e_{12} &= 35202448,\\
e_3 &= 17, & e_{13} &= 190568779,\\
e_4 &= 77, & e_{14} &= 1039296373,\\
e_5 &= 364, & e_{15} &= 5704834700,\\
e_6 &= 1776, & e_{16} &= 31494550253,\\
e_7 &= 8881, & e_{17} &= 174759749005,\\
e_8 &= 45296, & e_{18} &= 974155147162.\\
e_9 &= 234806,
\end{aligned}
$$

Figure 3: Linear recurrence defining $e_n$ with corresponding initial conditions.

the correctness of respective abstract machine implementations as well as facilitate their optimisation and eventual perfection.

With analytic generating functions $C_\infty(z)$ and $E_\infty(z)$ for plain closures and environments, respectively, it becomes possible to design efficient exact- or approximate-size samplers

(i.e. algorithms constructing random structures) corresponding to both combinatorial classes. In particular, we can use the general frameworks of Boltzmann samplers [20] by Duchon et al. or the so-called recursive method [35, 23] of Nijenhuis and Wilf. Remarkably, in both frameworks the sampler design resembles the recursive structure of the target combinatorial specification. Moreover, for a broad class of discrete structures such as, for instance, algebraic specifications, the sampler construction itself can be effectively automatised. Respective branching probabilities dictating the sampler's decisions are precomputed once and fixed throughout all subsequent executions.

In the recursive method, branching probabilities are established so to obtain an exact-size sampler, i.e. a sampler which generates random structures of a specific, given size $n$. In particular, using holonomic specifications for $C_\infty(z)$ and $E_\infty(z)$ it is possible to compute the related coefficients $[z^n]C_\infty(z)$ and $[z^n]E_\infty(z)$ using just $O(n)$ arithmetic operations, thus reach larger target sizes in a reasonable amount of time. On the other hand, if we drop the exact-size requirement of the outcome structures, it is possible to (again, automatically) construct an approximate-size Boltzmann sampler generating closures (respectively environments) of varying size in linear time, in terms of outcome size. Although the output size of constructed objects is itself random, it is possible to *calibrate* its expectation around a (not necessarily) finite mean. Furthermore, using an optional rejection phase, meant to dismiss inadmissible structures, it is possible to gain additional control over the sampler outcome.

Remarkably, both mentioned sampler frameworks admit effective tuning procedures influencing the expected internal shape of constructed objects, e.g. frequencies of desired sub-patterns [6]. It is therefore possible to control the expected internal structure of the generated closures and environments.

We offer prototype sampler implementations for plain environments and closures, within the above sampler frameworks at Github[3]. Likewise, we provide similar samplers for so-called closed closures and environments (see Section 8) based on the recursive method.

## 8. Counting closed closures

A closure $\langle M, e\rangle$ is said to be *$m$-open*, denoted also as $\langle M, e\rangle \in \mathcal{C}los_m$, if there exists a non-negative $p$ such that $M \in \mathcal{L}_{m+p}$ (i.e. $M$ is an $(m+p)$-open $\lambda$-term) and $e$ is a finite list (i.e. environment) of length $p$ consisting itself of $m$-open closures. In other words, $m$-open closures are structures defined by means of the following implicit combinatorial specification:

$$\mathcal{C}los_m ::= \mathcal{L}_m \times \Box \;\mid\; \mathcal{L}_{m+1} \times \langle \mathcal{C}los_m\rangle \;\mid\; \mathcal{L}_{m+2} \times \langle \mathcal{C}los_m, \mathcal{C}los_m\rangle \;\mid\; \cdots \tag{8.1}$$

In particular, a closure is said to be *closed*[4] if it is 0-open. Like in the case of $m$-open $\lambda$-terms, if a closure $\langle M, e\rangle$ is $m-$open, then it is also $(m+1)$-open. Consider the following example:

**Example 8.1.**

- $\langle \underline{0}\,\underline{1}, \langle \lambda\underline{3}\rangle : \Box\rangle$ is a 3-open closure.
- $\langle \underline{1}\,\underline{0}, \langle \lambda\underline{0}, \Box\rangle : \langle \lambda\lambda\underline{0}, \Box\rangle : \Box\rangle$ is a closed closure (0-open closure).

Let us remark that an $m$-open closure corresponds to a not yet evaluated $m$-open $\lambda$-term. Certainly, due to their ubiquity in the context of abstract machines, the most interesting

[3] `https://github.com/PierreLescanne/CountingAndGeneratingClosuresAndEnvironments`

[4] We acknowledge that speaking of closed closures is a bit odd, however terms "closure" and "closed" form a consecrated terminology that we merely associate together.

$m$-open closures are in fact closed. In the current section, we focus therefore on counting closed closures and corresponding closed environments.

**Example 8.2.** The following table lists the first few closed closures.

| size | closures | total |
|---|---|---|
| $0, 1$ | | 0 |
| 2 | $\langle \lambda \underline{0}, \square \rangle$ | 1 |
| 3 | $\langle \lambda\lambda \underline{0}, \square \rangle \quad \langle \underline{0}, \langle \lambda \underline{0}, \square \rangle \rangle$ | 2 |
| 4 | $\langle \lambda\lambda\lambda \underline{0}, \square \rangle \quad \langle \lambda\lambda \underline{1}, \square \rangle \quad \langle \lambda(\underline{00}), \square \rangle$ $\langle \lambda \underline{0}, \langle \lambda \underline{0}, \square \rangle \rangle \quad \langle \underline{0}, \langle \lambda\lambda \underline{0}, \square \rangle \rangle \quad \langle \underline{0}, \langle \underline{0}, \langle \lambda \underline{0}, \square \rangle \rangle \rangle$ | 6 |

Figure 4 gives the first 50 numbers of closed closures.

| n | $c_{0,n}$ | n | $c_{0,n}$ |
|---|---:|---|---:|
| 0 | 0 | 25 | 2039291268600 |
| 1 | 0 | 26 | 7690787869550 |
| 2 | 1 | 27 | 29071665271653 |
| 3 | 2 | 28 | 110130490287410 |
| 4 | 6 | 29 | 418043342219865 |
| 5 | 18 | 30 | 1589843149170521 |
| 6 | 58 | 31 | 6056959298323505 |
| 7 | 188 | 32 | 23113998858734867 |
| 8 | 630 | 33 | 88343015816573484 |
| 9 | 2140 | 34 | 338147576768474959 |
| 10 | 7384 | 35 | 1296106542004047500 |
| 11 | 25775 | 36 | 4974412840517200748 |
| 12 | 90919 | 37 | 19115189068830345885 |
| 13 | 323529 | 38 | 73539781161982872915 |
| 14 | 1160285 | 39 | 283234718823200209560 |
| 15 | 4189666 | 40 | 1092009621308203935814 |
| 16 | 15221235 | 41 | 4214435736178031843666 |
| 17 | 55602475 | 42 | 16280366813995192858378 |
| 18 | 204119165 | 43 | 62947860010954764058213 |
| 19 | 752691547 | 44 | 243596693995304845906020 |
| 20 | 2786900678 | 45 | 943448667650667612945764 |
| 21 | 10357265495 | 46 | 3656836859592859541767133 |
| 21 | 38623769249 | 47 | 14184639891328996401070032 |
| 23 | 144488013135 | 48 | 55060786067960705278258741 |
| 24 | 542090016461 | 49 | 213877295469617703331719718 |

Figure 4: The number of closed closures for $n = 0, \ldots, 49$

Establishing the asymptotic growth rate of the sequence $(c_{0,n})_n$ corresponding to closed closures of size $n$ poses a considerable challenge, much more involved than its plain counterpart. In the following theorem we show that there exists two constants $\underline{\rho}, \overline{\rho} < \rho_{L_\infty}$ such that $\lim_{n\to\infty} \frac{\underline{\rho}^{-n}}{c_{0,n}} = 0$ and $\lim_{n\to\infty} \frac{c_{0,n}}{\overline{\rho}^{-n}} = 0$. In other words, the asymptotic growth rate of $(c_{0,n})_n$ is bounded by two exponential functions of $n$.

**Theorem 8.3.** *There exist $\overline{\rho} < \underline{\rho}$ satisfying $\overline{\rho} < \underline{\rho} < \rho_{L_\infty}$ and functions $\theta(n), \kappa(n)$ satisfying $\limsup_{n\to\infty} \theta(n)^{1/n} = \limsup_{n\to\infty} \kappa(n)^{1/n} = 1$ such that for sufficiently large $n$ we have $\underline{\rho}^{-n}\theta(n) < c_{0,n} < \overline{\rho}^{-n}\kappa(n)$.*

*Proof.* Let us start with the generating function $C_0(z)$ associated with closed closures $\mathcal{C}los_0$. Note that from the specification (8.1), instantiated to $m = 0$, $C_0(z)$ is implicitly defined as

$$C_0(z) = \sum_{m\geq 0} L_m(z) C_0(z)^m. \tag{8.2}$$

We can therefore identify a closed closure $\mathfrak{c}$ with a tuple $(t, c_1, \ldots, c_m)$ where $m \geq 0$, $t$ is an $m$-open $\lambda$-term and $c_1, \ldots, c_m$ are closed closures themselves. We proceed with defining two auxiliary lower and upper bound classes $\underline{C}_0(z)$ and $\overline{C}_0(z)$ such that $[z^n]\underline{C}_0(z) \leq [z^n]C_0(z) \leq [z^n]\overline{C}_0(z)$ for all $n$. Next, we establish their asymptotic behaviour and, in doing so, provide exponential lower and upper bounds on the growth rate of closed closures.

We start with $\underline{C}_0(z) = \sum_{m\geq 0} L_0(z)\underline{C}_0(z)^m$. Note that $\underline{C}_0(z)$ is associated with closures in which each term is closed, independently of the corresponding environment length. Hence, as closed $\lambda$-terms are $m$-open for all $m \geq 0$, we have $[z^n]\underline{C}_0(z) \leq [z^n]C_0(z)$. Furthermore

$$\underline{C}_0(z) = \sum_{m\geq 0} L_0(z)\underline{C}_0(z)^m = L_0(z) \sum_{m\geq 0} \underline{C}_0(z)^m = \frac{L_0(z)}{1 - \underline{C}_0(z)}. \tag{8.3}$$

Solving the above equation for $\underline{C}_0(z)$ we find that $\underline{C}_0(z) = \frac{1}{2}\left(1 - \sqrt{1 - 4L_0(z)}\right)$. In such a form, it is clear that there are two potential sources of singularities, i.e. the singularity $\rho_{L_\infty}$ of $L_0(z)$, see Theorem 5.6, or the roots of the radicand $1 - 4L_0(z)$. Since $L_0(z)$ is increasing and continuous in the interval $(0, \rho_{L_\infty})$ we know that if $L_0(\rho_{L_\infty}) > \frac{1}{4}$ then there exists a $\underline{\rho} < \rho_{L_\infty}$ such that $L_0(\underline{\rho}) = \frac{1}{4}$. Unfortunately, we cannot simply check that $L_0(\rho_{L_\infty}) > \frac{1}{4}$ as there exists no known method of evaluating $L_0(z)$, defined by means of an infinite system of equations, at a given point. For that reason we propose the following approach.

Recall that a $\lambda$-term $M$ is said to be *$h$-shallow* if all its de Bruijn index values are (strictly) bounded by $h$, see [25]. Let $L_m^{(h)}(z)$ denote the generating function associated with $m$-open $h$-shallow $\lambda$-terms. Note that $L_0^{(h)}(z)$, i.e. the generating function corresponding to closed $h$-shallow $\lambda$-terms, has a finite computable representation. Indeed, we have

$$\begin{aligned}
L_0^{(h)}(z) &= zL_1^{(h)}(z) + zL_0^{(h)}(z)L_0^{(h)}(z)\\
L_1^{(h)}(z) &= zL_2^{(h)}(z) + zL_1^{(h)}(z)L_1^{(h)}(z) + z\\
L_2^{(h)}(z) &= zL_3^{(h)}(z) + zL_2^{(h)}(z)L_2^{(h)}(z) + z + z^2\\
&\cdots\\
L_{h-1}^{(h)}(z) &= zL_h^{(h)}(z) + zL_{h-1}^{(h)}(z)L_{h-1}^{(h)}(z) + z + z^2 + \cdots + z^{h-1}\\
L_h^{(h)}(z) &= zL_h^{(h)}(z) + zL_h^{(h)}(z)L_h^{(h)}(z) + z + z^2 + \cdots + z^h
\end{aligned} \tag{8.4}$$

Consider $m < h$. Each $m$-open $h$-shallow $\lambda$-term is either (a) in form of $\lambda M$ where $M$ is an $(m+1)$-open $h$-shallow $\lambda$-term due to the head abstraction, (b) in form of $MN$ where both $M$ and $N$ are $m$-open $h$-shallow $\lambda$-terms, or (c) a de Bruijn index in the set $\{\underline{0}, \underline{1}, \ldots, \underline{m-1}\}$. When $m = h$, we have the same specification with the exception of the first summand

$zL_h^{(h)}(z)$ where, as we cannot exceed $h$, terms under abstractions are $h$-open, instead of $(h+1)$-open.

Using such a form it is possible to evaluate $L_0^{(h)}(z)$ at each point $z \in (0, \rho_{(h)})$ where $\rho_{(h)} > \rho_{L_\infty}$ is the dominating singularity of $L_0^{(h)}(z)$ satisfying $\rho_{(h)} \xrightarrow[h\to\infty]{} \rho$, see [25]. Certainly, each closed $h$-shallow $\lambda$-term is in particular a closed $\lambda$-term. In consequence, $[z^n]L_0^{(h)}(z) \leq [z^n]L_0(z)$ for each $n$. Moreover, for all sufficiently large $n$ we have $[z^n]L_0^{(h)}(z) < [z^n]L_0(z)$. This coefficient-wise lower bound transfers onto the level of generating function values and we obtain $L_0^{(h)}(z) < L_0(z)$. Following the same argument, we also have $L_0^{(h)}(z) < L_0^{(h+1)}(z)$ for each $h \geq 1$. We can therefore use $L_0^{(h)}(z)$ to approximate $L_0(z)$ from below — the higher $h$ we choose, the better approximation we obtain. Using computer algebra software[5] it is possible to automatise the evaluation process of $L_0^{(h)}(\rho_{L_\infty})$ for increasing values of $h$ and find that for $h = 153$ we obtain

$$L_0^{(153)}(\rho_{L_\infty}) \doteq 0.25000324068941554\,. \tag{8.5}$$

Hence indeed, the asserted existence of $\underline{\rho} < \rho_{L_\infty}$ such that $L_0(\underline{\rho}) = \frac{1}{4}$ follows (interestingly, taking $h = 152$ does not suffice as $L_0^{152}(\rho_{L_\infty}) < \frac{1}{4}$). We fall hence in the super-critical composition schema[6] and note that $\underline{C}_0(z)$ admits a Newton-Puiseux expansion near $\underline{\rho}$ as follows:

$$\underline{C}_0(z) = \underline{a_0} - \underline{b_0}\sqrt{1 - \frac{z}{\underline{\rho}}} + O\left(\left|1 - \frac{z}{\underline{\rho}}\right|\right) \tag{8.6}$$

for some constants $\underline{a_0} > 0$ and $\underline{b_0} < 0$. Hence, $[z^n]C_0(z)$ grows asymptotically faster than $\underline{\rho}^{-n}\theta(n)$ where $\theta(n) = \dfrac{\underline{b_0}}{\Gamma(-\frac{1}{2})}n^{-3/2}$.

For the upper bound we consider $\overline{C}_0(z) = \sum_{m\geq 0} L_\infty(z)\overline{C}_0(z)^m$, i.e. the generating function associated with closures in which all terms are plain (either closed or open), independently of the constraint imposed by the corresponding environment length. Following the same arguments as before, we note that $[z^n]\overline{C}_0(z) > [z^n]C_0(z)$. Now

$$\overline{C}_0(z) = \sum_{m\geq 0} L_\infty(z)\overline{C}_0(z)^m = L_\infty(z)\sum_{m\geq 0}\overline{C}_0(z)^m = \frac{L_\infty(z)}{1 - \overline{C}_0(z)}. \tag{8.7}$$

Solving the equation for $\overline{C}_0(z)$ we find that $\overline{C}_0(z) = \frac{1}{2}\left(1 - \sqrt{1 - 4L_\infty(z)}\right)$. Note that in this case, we can easily handle the radicand expression $1 - 4L_\infty(z)$ and find out that, as in the lower bound case, we are in the super-critical composition schema. Specifically, $\overline{\rho} = \frac{1}{10}\left(25 - \sqrt{545}\right) \doteq 0.165476$, cf. (6.4), is the dominating singularity of $\overline{C}_0(z)$. In consequence, $\overline{C}_0(z)$ admits the following Newton-Puiseux expansion near $\overline{\rho}$:

$$\overline{C}_0(z) = \overline{a_0} - \overline{b_0}\sqrt{1 - \frac{z}{\overline{\rho}}} + O\left(\left|1 - \frac{z}{\overline{\rho}}\right|\right) \tag{8.8}$$

[5] `https://github.com/PierreLescanne/CountingAndGeneratingClosuresAndEnvironments`

[6] *Supercriticality* ensures that meromorphic asymptotics applies and entails strong statistical regularities (see [22] Section V.2 and Section IX.6).

for some constants $\overline{a_0} > 0$ and $\overline{b_0} < 0$. In conclusion, $[z^n]C_0(z)$ grows asymptotically slower than $(\overline{\rho})^{-n}\theta(n)$ where $\theta(n) = \dfrac{\overline{b_0}}{\Gamma(-\frac{1}{2})} n^{-3/2}$, finishing the proof. □

With an implicit expression defining $C_0(z)$, see (8.2), efficient random generation of closed closures poses a difficult task. Though we have no efficient Boltzmann samplers, it is possible to follow the recursive method and obtain exact-size samplers for a moderate range of target sizes. We offer a prototype sampler of this kind, available at Github[7].

## 9. Conclusions

We view our contribution as a small step towards the quantitative, average-case analysis of evaluation complexity in $\lambda$-calculus. Using standard tools from analytic combinatorics, we investigated some combinatorial aspects of environments and closures — fundamental structures present in various formalisms dealing with normalisation in $\lambda$-calculus, especially in its variants with explicit substitutions [31, 10]. Though plain environments and closures are relatively easy to count and generate, their closed counterparts pose a considerable combinatorial challenge. The implicit and infinite specification of closed closures based on closed $\lambda$-terms complicates significantly the quantitative analysis, namely estimating the exponential factor in the asymptotic growth rate, or effectively generating random closed closures. In particular, getting more parameters of the asymptotic growth will require more sophisticated methods, like, for instance, the recent infinite system approximation techniques of Bodini, Gittenberger and Gołębiewski [12].

## References


[1] Beniamino Accattoli and Ugo Dal Lago. (Leftmost-Outermost) beta reduction is invariant, indeed. *Logical Methods in Computer Science*, 12(1), 2016. doi:10.2168/LMCS-12(1:4)2016.

[2] Martin Avanzini, Ugo Dal Lago, and Georg Moser. Analysing the complexity of functional programs: higher-order meets first-order. In Kathleen Fisher and John H. Reppy, editors, *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada.*, pages 152–164. ACM, 2015. doi:10.1145/2784731.2784753.

[3] Martin Avanzini and Georg Moser. Closing the gap between runtime complexity and polytime computability. In Christopher Lynch, editor, *Proceedings of the 21st International Conference on Rewriting Techniques and Applications, RTA 2010, July 11-13, 2010, Edinburgh, Scottland, UK*, volume 6 of *LIPIcs*, pages 33–48. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010. doi:10.4230/LIPIcs.RTA.2010.33.

[4] Zine-El-Abidine Benaissa, Daniel Briaud, Pierre Lescanne, and Jocelyne Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699-722, 1996. doi:10.1017/S0956796800001945.

[5] Maciej Bendkowski. Normal-order reduction grammars. *Journal of Functional Programming*, 27, 2017. doi:10.1017/S0956796816000332.

[6] Maciej Bendkowski, Olivier Bodini, and Sergey Dovgal. Polynomial tuning of multiparametric combinatorial samplers. In Markus E. Nebel and Stephan G. Wagner, editors, *Proceedings of the Fifteenth Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2018, New Orleans, LA, USA, January 8-9, 2018.*, pages 92–106. SIAM, 2018. URL: https://doi.org/10.1137/1.9781611975062.9, doi:10.1137/1.9781611975062.9.


[7] https://github.com/PierreLescanne/CountingAndGeneratingClosuresAndEnvironments

[7] Maciej Bendkowski, Katarzyna Grygiel, Pierre Lescanne, and Marek Zaionc. A natural counting of lambda terms. In *Theory and Practice of Computer Science: 42nd International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM*, pages 183–194. Springer Berlin Heidelberg, 2016.

[8] Maciej Bendkowski, Katarzyna Grygiel, Pierre Lescanne, and Marek Zaionc. Combinatorics of $\lambda$-terms: a natural approach. *Journal of Logic and Computation*, 27(8):2611–2630, 2017. doi:10.1093/logcom/exx018.

[9] Maciej Bendkowski, Katarzyna Grygiel, and Marek Zaionc. On the likelihood of normalization in combinatory logic. *Journal of Logic and Computation*, 2017. doi:10.1093/logcom/exx005.

[10] Maciej Bendkowski and Pierre Lescanne. Combinatorics of explicit substitutions. In David Sabel and Peter Thiemann, editors, *Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming, PPDP 2018, Frankfurt am Main, Germany, September 03-05, 2018*, pages 7:1–7:12. ACM, 2018. URL: http://doi.acm.org/10.1145/3236950.3236951, doi:10.1145/3236950.3236951.

[11] Olivier Bodini, Danièle Gardy, and Bernhard Gittenberger. Lambda-terms of bounded unary height. In Philippe Flajolet and Daniel Panario, editors, *Proceedings of the Eighth Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2011, San Francisco, California, USA, January 22, 2011*, pages 23–32. SIAM, 2011. doi:10.1137/1.9781611973013.3.

[12] Olivier Bodini, Bernhard Gittenberger, and Zbigniew Gołębiewski. Enumerating lambda terms by weighted length of their de bruijn representation. *CoRR*, abs/1707.02101, 2017. URL: https://arxiv.org/abs/1707.02101.

[13] Christine Choppy, Stéphane Kaplan, and Michèle Soria. Algorithmic complexity of term rewriting systems. In Pierre Lescanne, editor, *Rewriting Techniques and Applications, 2nd International Conference, RTA-87, Bordeaux, France, May 25-27, 1987, Proceedings*, volume 256 of *Lecture Notes in Computer Science*, pages 256–273. Springer, 1987. doi:10.1007/3-540-17220-3_22.

[14] Christine Choppy, Stéphane Kaplan, and Michèle Soria. Complexity analysis of term-rewriting systems. *Theor. Comput. Sci.*, 67(2&3):261–282, 1989. doi:10.1016/0304-3975(89)90005-4.

[15] Koen Claessen and John Hughes. Quickcheck: A lightweight tool for random testing of haskell programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, pages 268–279. ACM, 2000.

[16] Pierre-Louis Curien. *Categorical Combinators, Sequential Algorithms, and Functional Programming (2nd Ed.)*. Birkhauser Boston Inc., Cambridge, MA, USA, 1994.

[17] Pierre-Louis Curien, Thérèse Hardin, and Jean-Jacques Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, March 1996. doi:10.1145/226643.226675.

[18] René David, Katarzyna Grygiel, Jakub Kozik, Christophe Raffalli, Guillaume Theyssier, and Marek Zaionc. Asymptotically almost all $\lambda$-terms are strongly normalizing. *Logical Methods in Computer Science*, 9:1–30, 2013.

[19] Nicolaas G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972.

[20] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability and Computing*, 13(4-5):577–625, 2004.

[21] Philippe Flajolet and Andrew M. Odlyzko. Singularity analysis of generating functions. *SIAM Journal on Discrete Mathematics*, 3(2):216–240, 1990.

[22] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 1 edition, 2009.

[23] Philippe Flajolet, Paul Zimmermann, and Bernard Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1):1–35, 1994.

[24] Étienne Ghys. *A singular mathematical promenade*. Ecole Normale Supérieure, 2017. URL: http://perso.ens-lyon.fr/ghys/promenade/.

[25] Bernhard Gittenberger and Zbigniew Gołębiewski. On the number of lambda terms with prescribed size of their de Bruijn representation. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS*, pages 40:1–40:13, 2016.

[26] Katarzyna Grygiel and Pierre Lescanne. Counting and generating terms in the binary lambda calculus. *Journal of Functional Programming*, 25, 2015. doi:10.1017/S0956796815000271.

[27] Donald E. Knuth. Mathematical Analysis of Algorithms, 2000. First chapter of [28].

[28] Donald E. Knuth. *Selected Papers on Analysis of Algorithms*, volume 102 of *CSLI Lecture Notes*. Stanford, California: Center for the Study of Language and Information, 2000.

[29] Ugo Dal Lago and Simone Martini. On constructor rewrite systems and the lambda calculus. *Logical Methods in Computer Science*, 8(3), 2012. doi:10.2168/LMCS-8(3:12)2012.

[30] Peter J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964. doi:10.1093/comjnl/6.4.308.

[31] Pierre Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$: A journey through calculi of explicit substitutions. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 60–69. ACM, 1994.

[32] Pierre Lescanne. The lambda calculus as an abstract data type. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *Recent Trends in Data Type Specification, 11th Workshop on Specification of Abstract Data Types Joint with the 8th COMPASS Workshop, Oslo, Norway, September 19-23, 1995, Selected Papers*, volume 1130 of *Lecture Notes in Computer Science*, pages 74–80. Springer, 1995. URL: https://doi.org/10.1007/3-540-61629-2_37, doi:10.1007/3-540-61629-2_37.

[33] Michel Mauny and Ascánder Suárez. Implementing functional languages in the categorical abstract machine. In *LISP and Functional Programming*, pages 266–278, 1986.

[34] John C. Mitchell. *Concepts in Programming Language (1st Ed.)*. Cambridge University Press, New York, NY, USA, 2002.

[35] Albert Nijenhuis and Herbert S. Wilf. *Combinatorial Algorithms*. Academic Press, 2 edition, 1978.

[36] Michał H. Pałka. *Random Structured Test Data Generation for Black-Box Testing*. PhD thesis, Chalmers University of Technology, 2012.

[37] Gordon David Plotkin. Call-by-name, call-by-value and the $\lambda$-calculus. *Theoretical Computer Science*, 1(2):125 – 159, 1975. doi:https://doi.org/10.1016/0304-3975(75)90017-1.

[38] Bruno Salvy and Paul Zimmermann. Gfun: a Maple package for the manipulation of generating and holonomic functions in one variable. *ACM Transactions on Mathematical Software*, 20(2):163–177, 1994.

[39] Robert Sedgewick and Philippe Flajolet. *An Introduction to the Analysis of Algorithms (2nd Edition)*. Createspace Independent Pub, 2014.

[40] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, 2004. URL: http://doi.acm.org/10.1145/990308.990310, doi:10.1145/990308.990310.

[41] Herbert S. Wilf. *Generatingfunctionology*. A. K. Peters, Ltd., 2006.