Counting and generating lambda terms

Katarzyna Grygiel, Pierre Lescanne

Counting and generating lambda terms

Katarzyna Grygiel *
Theoretical Computer Science Department,
Faculty of Mathematics and Computer Science,
Jagiellonian University,
ul. Prof. Łojasiewicza 6, 30-348 Kraków, Poland
email:– grygiel@tcs.uj.edu.pl

Pierre Lescanne
ENS de Lyon,
LIP (UMR 5668 CNRS ENS Lyon UCBL INRIA)
University of Lyon,
46 allée d'Italie, 69364 Lyon, France
email: pierre.lescanne@ens-lyon.fr

December 14, 2012

Abstract

Lambda calculus is the basis of functional programming and higher order proof assistants. However, few is known about combinatorial properties of lambda terms, in particular, about their asymptotic distribution and random generation. Among others, this paper tries to answer questions like: How many terms of a given size are there? What is a "typical" structure of a simply typed term? Despite their ostensible simplicity, these questions still remain unanswered, whereas solutions to such problems are essential for testing compilers and optimizing programs whose expected efficiency depends on the size of terms. Our approach toward the aforementioned problems may be later extended to any language with bounded variables, i.e., with scopes and declarations.

This paper presents two complementary approaches: one, theoretical, uses complex analysis and generating functions, the other, experimental, is based on a computer algebra software, able to handle huge numbers efficiently. Thanks to de Bruijn indices, we provide formulas for the number of closed lambda terms of a given size and show their relevance to recursively defined integer polynomials. Knowledge on the asymptotic behavior of the polynomial coefficients suggests the approach toward the problem of the asymptotic behavior of numbers of closed lambda terms. Indeed, this problem is unamenable to standard generating function methods due

*Supported by the National Science Center of Poland, grant number 2011/01/B/HS1/00944.

to the unusual form of the recurrences. As a by-product of the counting formulas, we design an algorithm for generating lambda terms. Performed tests provide us with experimental data, like the average depth of bound variables and the average number of head lambdas. We also create random generators for various sorts of terms. Thereafter, we conduct experiments that answer questions like: What is the ratio of simply typed terms among all terms? *(Very small!)* How are simply typed lambda terms distributed among all lambda terms? *(A typed term almost always starts with an abstraction.)*

In this paper, variables have size 0.

Keywords: Lambda calculus, combinatorics, functional programming, test, random generator, Catalan numbers

1 Introduction

Let us start with a few questions relevant to the problems we address.

- How many closed λ-terms are of size 50 (up to α-conversion)?

  $$99665778334452328341705500204014807522670099639155869526994685 2267.$$

- How many terms of size n are there?
  We will give a recursive formula for this number in Section 2*.*

- What is enumerated by the sequence

  $$0, 1, 3, 14, 82, 579, 4741, 43977, 454283, 5159441, 63782411?$$

  This sequence enumerates closed terms of size 0*,* 1*,* 2*,* 3*,* 4*,* 5*,* 6*,* 7*,* 8*,* 9*,* 10*. We will give three ways to compute it (Section* 4*).*

- Is it possible to generate simply typed terms randomly?
  Yes, according to the process, which consists in generating random lambda terms with uniform probability and sieving those that are simply typed. Thus, we can generate randomly simply typed terms of size up to 50 *and less randomly simply typed terms of size* 200*.*

- Is a term starting with an abstraction more likely to be typable than a term starting with an application?
  The answer is positive as shown in Figure 9*, which gives the distribution of simply typed lambda terms among all lambda terms.*

- Do these results have practical consequences?
  Yes, they enable random generation of simply typed terms in an efficient way in the case of terms of size up to 30 *(random) and up to* 100 *(biased) in order to debug compilers or other programs, manipulating terms, e.g., type checkers or pretty printers.*

The above questions seem rather classical, but amazingly very few is known about combinatorial aspects of lambda terms. However, the answers to these questions are extremely important not only for a better understanding of the structure of lambda terms, but also for people who build test samples for debugging compilers or for those who optimize the average run of programs by a better knowledge of the distribution of terms. Perhaps the reason of this ignorance lies in the surprising form of the recurrences. Indeed, due to the presence of bound variables, the recurrence does not work in the way mathematicians expect and are used to. Thus none of the methods used in the reference book of Flajolet and Sedgewick [5] applies. Why is that? In what follows we compute the number of lambda terms (and of normal forms) of size n with at most m bound variables. Denoting the number of such terms by $T_{n,m}$, the formula for $T_{n,m}$ contains $T_{n-1,m+1}$ and this growth of m makes the formula averse to treatments by generating functions and classical analytic combinatorics. We notice that for a given n the expression for $T_{n,m}$ is a polynomial in m. These polynomials can be described inductively and their coefficients are given by recurrence formulas. These formulas are still complex, but can be used to compute the constant coefficients, which correspond to the numbers of closed lambda terms. For instance, the leading coefficients of the polynomials are the well known Catalan numbers which count binary trees.

In order to find the recurrence formula for the number of λ-terms of a given size, we make use of the representation of variables in λ-terms by de Bruijn indices. Recall that a de Bruijn index is a natural number which replaces a term variable and enumerates the number of λ's encountered on the way between the variable and the λ which binds the latter. In this paper, we assume the combinatorial model in which the size of each occurrence of abstraction or application is counted as 1, while the size of variables (de Bruijn indices) as 0. This method is a realistic model of the complexity of λ-terms and allows us to derive the recurrences very naturally. Since we manipulate big numbers, we need for the computation an efficient computer algebra system. We have chosen PARI/GP [16], a package of the software SAGE [15]

From the formula for counting λ-terms we can derive one-to-one assignments of numbers in the interval $[1..P_n(m)]$ to terms of size n with at most m distinct free indices. From this correspondence, we can develop a program for generating λ-terms, more precisely for building the λ-term associated with a number in the interval $[1..P_n(m)]$. If we pick a random number in the interval, then we get a random term of size n with at most m distinct free variables. Beside the interest in such a random generation for applications like testing, this allows us to compute practical values of parameters by Monte-Carlo methods. Overall, we are able to build a random generator for simply typed terms. Unlike the method used traditionally [13], which consists in unfolding the typing tree, we generate random λ-terms and test their typability, until we find a simply typed term. This method allows us to generate on a laptop simply typed λ-terms up to size 50. We also use this method to describe the distribution of well-typed terms among plain terms and well-typed normal forms among plain normal forms. This shows that terms starting with abstractions are more likely to be typable than terms starting with applications, the phenomenon being more manifest on normal forms. From this, we derive a way to generate large typed terms up to 200, with a biased randomness.

Related works

There are very few works on counting lambda terms, whereas counting first order terms is a classical domain of combinatorics. Apparently the first traces of counting expressions with (unbound) variables can be attributed to Hipparchus of Rhodes (c. 190–120 BC) (see [5] p. 68). Flajolet and Sedgewick book [5] is the reference on this subject. Concerning counting λ-terms, we can cite only four works. [3] and [1] study asymptotic behavior of formulas on counting lambda-terms. Strictly speaking they do not exhibit a recurrence formula for counting. In particular, David et al. [3] only bound superiorly and inferiorly the numbers of λ-terms in order to get information about the distribution of families of terms. For instance, they prove that "asymptotically almost all λ-terms are strongly normalizing". In [10] the second author of the present paper proposes formulas for counting λ-terms in the case of variables of weight 1, with more complex formulas and less results. On another hand, Christophe Raffalli proposed a formula for counting closed λ-terms, which he derives from the formula for counting λ-terms with exactly m distinct free variables, whereas in this paper we count terms with at most m such variables. His formula appears only in the *On-line Encyclopedia of Integer Sequences* under number **A135501** and is much more complicated with three embedded levels of Σ's. He considers weight 1 for the variables. His formula can be easily adapted to variables of weight 0, but remains complex (see Section 4 and Appendix A).

As concerns random generation, [17] proposed algorithms for randomly generating untyped λ-terms in the spirit of the counting formula of Raffalli. [12, 13] use generation of λ-term to test Haskell compilers. Pałka acknowledges that, due to his method, he cannot guarantee the randomness of his generator (see discussion in [12] p. 21 and p. 45). Nonetheless, he found eight failures and four bugs in the *Glasgow Haskell Compiler* showing the interest in the method. [14] study the feasibility of generic programming for the enumeration of typed terms. The given examples are of size 4 or 5, no realistic examples are provided, randomness is not addressed and the authors confess that their algorithm is not efficient. Knowing that there are 11807 simply typed closed terms of size 7, one wonders the actual use of such an enumeration and it seems unrealistic to address enumeration for larger numbers. The "related work" section of [14] covers similar approaches, which all consist in cutting branches. They all fail to generate random terms. A presentation of tree-like structure generation and a history of combinatorial generation is given in [8].

Structure of the paper

According to its title, the paper is divided into two parts, one focuses on counting terms and its mathematical treatment, the other on term generation and its applications. The first part (Sections 2 and 5) is devoted to the formulas counting λ-terms. In Section 2 we study polynomials giving the numbers of terms of size n with at most m distinct free variables, especially formulas giving the coefficients of the polynomials. In Section 3, we shows that the numbers of i-contexts give a combinatoric interpretation of the coefficients of the polynomials and yield a new formula for counting the closed terms of size n. If we add formulas for counting lambda terms of size n with exactly m variables, we have three formulas of three different origins for counting closed terms which we describe and compare in Section 4. In Section 5 we derive generating functions and asymptotical values

for these coefficients. In Section 6 we give a formula for counting normal forms. In the second part, i.e., in Section 7 and Section 8, we propose programs to generate untyped and typed terms and normal forms. Section 9 is devoted to experimental results. SAGE script related to this paper can be found at

`https://dl.dropbox.com/u/2518969/LambdaTermsEnumerationAndGeneration.sws`

and raw statistics can be found at

`https://dl.dropbox.com/u/2518969/Statistics.txt`.

2 Counting terms with at most m variables

We represent terms by de Bruijn indices [4], which means that variables are represented by numbers $\underline{1}, \underline{2}, \ldots, \underline{\mathsf{m}}, \ldots$, where an index, for instance $\underline{\mathsf{k}}$, is the number of λ's, above the location of the index and below the λ that binds the variable, in a representation of λ-terms by trees. For instance, the term with variables $\lambda x.\lambda y.x\,y$ is represented by the term with de Bruijn indices $\lambda\lambda\underline{21}$. The variable x is bound by the top λ. Above the occurrence of x, there are two λ's, therefore x is represented by $\underline{2}$ and from the occurrence of y, we count just the λ that binds y; so y is represented by $\underline{1}$. In what follows we will call *terms*, the untyped terms with de Bruijn indices and often we will speak indifferently of variables and (de Bruijn) indices. Assume that not all the indices are bound. In other words, there may be indices that do not correspond to surrounding λ's, we call them "free". Here is the convention on the interval of "free" indices which appear in a term t. An interval is a set $\mathcal{I}(m) = \{\underline{1}, \underline{2}, \ldots, \underline{\mathsf{m}}\}$ of indices, If t is an index $\underline{\mathsf{i}}$, the interval of indices of t is any interval $[\underline{1}, \ldots, \underline{\mathsf{m}}]$ with $1 \leq i \leq m$. Now assume that the interval of free indices of t is $[\underline{1}, \ldots, \underline{\mathsf{m}+1}]$, then the interval of free indices of λt is $[\underline{1}, \ldots, \underline{\mathsf{m}}]$, because the indice $\underline{1}$ have been bound and the others are assumed to decrease by one. If the interval of indices of t and s is $[\underline{1}, \ldots, \underline{\mathsf{m}}]$, then the interval of indices of $s\,t$ is $[\underline{1}, \ldots, \underline{\mathsf{m}}]$. For instance, the interval of the term $\lambda\underline{3}\,\underline{1}$ is the interval $[\underline{1}, \underline{2}, \ldots, \underline{\mathsf{m}}]$ for any $m \geq 2$.

Let us denote the set of terms of size n, with at most m "free" de Bruijn indices (with $[\underline{1}, \ldots, \underline{\mathsf{m}}]$ as interval of indices) by $\mathcal{T}_{n,m}$. A term from $\mathcal{T}_{n,m}$ is either a de Bruijn index or an abstraction on a term with at most $m+1$ indices, i.e., a term in $\mathcal{T}_{n,m+1}$, or an application of a term in $\mathcal{T}_{n,m}$ on a term in $\mathcal{T}_{n,m}$. We can write, using @ as the application symbol,

$$\mathcal{T}_{n+1,m} = \lambda\mathcal{T}_{n,m+1} \uplus \biguplus_{k=0}^{n} \mathcal{T}_{n-k,m} @ \mathcal{T}_{k,m}.$$

We assume that the operators λ and @ have size 1 and that de Bruijn indices have size 0. From this, we get the following two equations specifying $T_{n,m}$:

$$\begin{aligned} T_{0,m} &= m \\ T_{n+1,m} &= T_{n,m+1} + \sum_{i=0}^{n} T_{i,m} T_{n-i,m}. \end{aligned}$$

This means that there are m terms of size 0 with at most m free de Bruijn indices, which are terms that are just these indices. Terms of size n with at most m de Bruijn indices are either abstractions with at most $m+1$ indices on a term of size $n-1$ or applications

n\m	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	3	7	13	21	31	43
2	3	13	41	99	199	353	573
3	14	76	312	962	2386	5064	9596
4	82	542	2784	10732	32510	82122	181132
5	579	4493	27917	131715	482015	1440929	3687513
6	4741	42131	307943	1741813	7612097	26763551	79193491
7	43977	439031	3690055	24537945	126536933	519788827	1771730211
8	454283	5020105	47635777	365779679	2198772055	10477986133	40973739725
9	5159441	62382279	658405747	5744911157	39769404045	218213327131	974668783199
10	63782411	835980065	9695617821	94786034723	746744227319	4681133293821	23769847893305
11	851368766	12004984120	151488900012	1639198623818	14531624611594	103244315616876	593009444765240
12	12188927818	183754242626	2502346785164	29658034018852	292747054367966	2338363467319958	15112319033576416
13	186132043831	2984264710781	43560247035581	560484305049943	6100545513799835	54347237563601321	393031286917940401
14	3017325884473	51220227153987	796828655891895	11046637024014049	131425939696979805	1295642289776992983	10425601907159190187

Figure 1: Values of $T_{n,m}$ for n and m up to 14 and 6, respectively

n	P_n
0	m
1	$m^2 + m + 1$
2	$2m^3 + 3m^2 + 5m + 3$
3	$5m^4 + 10m^3 + 22m^2 + 25m + 14$
4	$14m^5 + 35m^4 + 94m^3 + 154m^2 + 163m + 82$
5	$42m^6 + 126m^5 + 396m^4 + 838m^3 + 1277m^2 + 1235m + 579$
6	$132m^7 + 462m^6 + 1654m^5 + 4260m^4 + 8384m^3 + 11791m^2 + 10707m + 4741$
7	$429m^8 + 1716m^7 + 6868m^6 + 20742m^5 + 49720m^4 + 90896m^3 + 120628m^2 + 104055m + 43977$
8	$1430m^9 + 6435m^8 + 28396m^7 + 98028m^6 + 275886m^5 + 617096m^4 + 1068328m^3 + 1352268m^2 + 1117955m + 454283$

Figure 2: The first eight polynomials P_n

of terms with at most m indices to make a term of size n. As we said in the introduction, the 11 first values of $T_{n,0}$ are:

$$0, 1, 3, 14, 82, 579, 4741, 43977, 454283, 5159441, 63782411.$$

Figure 1 gives all the values of $T_{n,m}$ for n up to 14 and m up to 6. For instance, there is 1 closed term of size 1, namely $\lambda\underline{1}$, there are 3 closed terms of size 2, namely $\lambda\lambda\underline{1}, \lambda\lambda\underline{2}, \lambda\underline{1}\,\underline{1}$, and there are 14 closed terms of size 3, namely

$$\lambda\lambda\lambda\underline{1}, \lambda\lambda\lambda\underline{2}, \lambda\lambda\lambda\underline{3}, \lambda\lambda\underline{1}\,\underline{1}, \lambda\lambda\underline{1}\,\underline{2}, \lambda\lambda\underline{2}\,\underline{1}, \lambda\lambda\underline{2}\,\underline{2}, \lambda(\underline{1}\,\lambda\underline{1}),$$
$$\lambda(\underline{1}\,\lambda\underline{2}), \lambda\underline{1}(\underline{11}), \lambda((\lambda\underline{1})\,\underline{1}), \lambda((\lambda\underline{2})\,\underline{1}), \lambda((\underline{1}\,\underline{1})\,\underline{1}), (\lambda\underline{1})\,\lambda\underline{1}.$$

Notice that in Section 7, we describe how to assign a number to a term and therefore how to list terms with increasing numbers. The above terms have been listed in that order.

For every $n \geq 0$, we can associate with $T_{n,m}$ a polynomial $P_n(m)$ in m. First, let us define polynomials P_n in the following recursive way:

$$\begin{aligned} P_0(m) &= m \\ P_{n+1}(m) &= P_n(m+1) + \sum_{i=0}^{n} P_i(m) P_{n-i}(m) \end{aligned}$$

The sequence $(P_n(0))_{n\geq 0}$ corresponds to the sequence $(T_{n,0})_{n\geq 0}$ enumerating closed lambda terms. The first eight polynomials are given in Figure 2.

This means that the constant coefficient of a polynomial $P_n(m)$ is exactly the number of closed lambda terms of size n. We propose a way of computing the coefficients of these polynomials.

Lemma 1 *For every n, the degree of the polynomial P_n is equal to $n+1$.*

Proof: The result follows immediately by induction on n from the definition of P_n. $\square$

For $i > 0$ and $n \geq 0$, let us denote by p_n^i the i-th leading coefficient of the polynomial P_n, i.e., we have

$$P_n(m) = p_n^1 m^{n+1} + p_n^2 m^n + \ldots + p_n^i m^{n+2-i} + \ldots + p_n^{n+1} m + p_n^{n+2}.$$

Lemma 2 *For every $n \geq 0$ and $i > 0$,*

$$\begin{aligned} p_0^1 &= 1, \quad p_0^i = 0 \ \ \textit{for} \ \ i > 1, \\ p_{n+1}^i &= \sum_{j=0}^{i-2} \binom{n+1-j}{i-2-j} p_n^{j+1} + \sum_{k=1}^{i} \sum_{j=0}^{n} p_j^k p_{n-j}^{i+1-k}. \end{aligned}$$

Proof: Since $P_0(m) = m$, equations from the first line in the above lemma are trivial.

The i-th leading coefficient in the polynomial $P_{n+1}(m)$ is equal to the sum of coefficients standing at m^{n+3-i} in polynomials $P_n(m+1)$ and $\sum_{j=0}^{n} P_j(m) P_{n-j}(m)$.

The first of these polynomials, $P_n(m+1)$, is as follows:

$$p_n^1 (m+1)^{n+1} + \ldots + p_n^{i-1} (m+1)^{n+3-i} + \ldots + p_n^{n+2},$$

therefore the coefficient of m^{n+3-i} in $P_n(m+1)$ is equal to

$$\binom{n+1}{i-2} p_n^1 + \binom{n}{i-3} p_n^2 + \ldots + \binom{n+3-i}{0} p_n^{i-1} = \sum_{j=0}^{i-2} \binom{n+1-j}{i-2-j} p_n^{j+1}.$$

In the case of the second polynomial, $\sum_{j=0}^{n} P_j(m) P_{n-j}(m)$, we have

$$\begin{aligned} &\left(p_j^1 m^{j+1} + \ldots + p_j^k m^{j+2-k} + \ldots + p_j^{j+2}\right) \\ \cdot &\left(p_{n-j}^1 m^{n-j+1} + \ldots + p_{n-j}^{i+1-k} m^{n-j+1+k-i} + \ldots + p_{n-j}^{n-j+2}\right), \end{aligned}$$

therefore the coefficient of m^{n+3-i} in $\sum_{j=0}^{n} P_j(m) P_{n-j}(m)$ is equal to

$$\sum_{k=1}^{i} \sum_{j=0}^{n} p_j^k p_{n-j}^{i+1-k}.$$

$\square$

3 Counting contexts

In λ-calculus, a i-context is a closed term with i holes. We consider that a hole has size 0 and we assume the holes are numbered $1, ..., i$ as we meet them when traversing the term from left to right. For instance, if we represent the holes by [ ], then $(\lambda\underline{1}[\ ])\lambda\lambda[\ ]\underline{2}$ is a 2-context of size 4, its holes are numberd as follows $(\lambda\underline{1}[\ ]_1)\lambda\lambda[\ ]_2\underline{2}$. The 0-contexts are the closed terms. Therefore there is only one 1-context of size 0 and no context of size 0 for $i \neq 0$. Let us write $c_{n,i}$ the number of i-contexts of size n. One notices that

$$\begin{aligned} c_{0,1} &= 1 \\ c_{0,i} &= 0 \text{ for } i \neq 1. \end{aligned}$$

Let us now see how we build a context from smaller contexts.

By abstraction, a i-context of size $n+1$ can be built from a j-context of size n and a set of $j-i$ holes among the j holes of the j-context where one puts variables (or indices) to be abstracted. There are $\binom{j}{i} c_{n,j}$ such i-contexts. One has to sum those quantities from i to $n+1$ to get the numbers of i-contexts built this way.

By application, a i-context of size $n+1$ can be built by applying a context on another context, i.e., a j-context of size k applied on a $i-j$-context of size $n-k$ (recall that the composition operator has size 1). To get all the contexts built this way, one has to sum from $j=0$ to $j=i$ and form $k=0$ to $k=n$.

Hence we get the formula:

$$c_{n+1,i} = \sum_{j=i}^{n+1} \binom{j}{i} c_{n,j} + \sum_{j=0}^{i} \sum_{k=0}^{n} c_{k,j} c_{n-k,i-j}. \qquad (*)$$

From contexts we can see how we can build terms. More precisely from a i-context of size n and a map f from $[1..i]$ to $[1..m]$, we can insert the index $f(j)$ in the j^{th} hole to build a term of size n with i occurrences of free variables taken among m ones. There are $c_{n,i}m^i$ such terms. Therefore

$$T_{n,m} = c_{n,n+1}m^{n+1} + ... + c_{n,i}m^i + ... + c_{n,0}$$

is the number of λ-terms of size n with at most m variables, which is the polynomial P_n. In particular $c_{n,n+2-i} = p_n^i$. The coefficients $c_{n,i}$ of the polynomials P_n's count the i-contexts of size n. We see that $c_{n,i} = 0$ when $i > n+1$.

The case $i = n+2$. In the case when $i = n+2$, using the fact that $c_{n,i} = 0$, when $i > n+1$, the equation $(*)$ boils down to:

$$c_{n+1,n+2} = \sum_{k=0}^{n} c_{k,k+1} c_{n-k,n-k+1}$$

which is characteristic of the Catalan numbers. Indeed $n+1$-contexts of size n have no abstraction, only applications and are therefore binary trees.

4 Three formulas for counting closed terms

We have found three formulas to compute the number of closed terms of size n. Let us summarize them:

Case $m = 0$ for terms with at most m distinct free variables
$T_{n,0}$ where

$$\begin{aligned} T_{0,m} &= m \\ T_{n+1,m} &= T_{n,m+1} + \sum_{i=0}^{n} T_{i,m} T_{n-i,m} \end{aligned}$$

This formula is clearly the simplest. Its simplicity, one sum and no binomial, allows unfolding it and on this basis building a program for term generation.

Case $m = 0$ for terms with exactly m distinct free variables
$f_{n,0}$ where

$$\begin{aligned} f_{0,0} &= 0 \\ f_{0,1} &= 1 \\ f_{n,m} &= 0 \text{ if } m > n+1 \\ f_{n+1,m} &= f_{n,m} + f_{n,m+1} + \\ &\quad \sum_{p=0}^{n} \sum_{c=0}^{m} \sum_{k=0}^{m-c} \binom{m}{c} \binom{m-c}{k} f_{p,k+c} f_{n-p,m-k}. \end{aligned}$$

This formula is the most complex. The way it can be derived is given in Appendix A.1. In Appendix A.2 we give a simple connection between the $T_{n,m}$'s and the $f_{n,m}$'s.

0-contexts
$c_{n,0}$ where

$$\begin{aligned} c_{0,1} &= 1 \\ c_{0,i} &= 0 \text{ for } i \neq 1 \\ c_{n+1,i} &= \sum_{j=i}^{n+1} \binom{j}{i} c_{n,j} + \sum_{j=0}^{i} \sum_{k=0}^{n} c_{k,j} c_{n-k,i-j}. \end{aligned}$$

5 Generating functions for the coefficients of the P_n's

For every positive integer i, let us denote by a_i the generating function for the sequence $(p_n^i)_{n \geq 0}$, i.e.,

$$a_i(z) = \sum_{n=0}^{\infty} p_n^i z^n = \sum_{n=0}^{\infty} c_{n,n+2-i} z^n.$$

The p_n^i's count the number of contexts of size n having $n+2-i$ holes i.e., having almost as many holes as their size, where "almost as many as" means "except a fixed number $i-2$". For the sake of clarity, instead of writing $a_i(z)$ sometimes we simply write a_i.

In order to compute functions a_i, we apply the following basic fact about generating functions.

Fact 3 *Let f and g be generating functions for sequences $(f_n)_{n\geq 0}$ and $(g_n)_{n\geq 0}$, respectively. Then*

(i) the generating function for the sequence $\left(\binom{n}{k} f_n\right)_{n\geq 0}$, where k is a fixed positive integer, is given by $\frac{z^k f^{(k)}}{k!}$,

(ii) the generating function for the sequence $\left(\sum_{i=0}^{n} f_i g_{n-i}\right)_{n\geq 0}$ is given by $f \cdot g$.

(iii) the generating function for the sequence $\left(\binom{n-j}{i} f_n\right)_{n\geq 0}$, where $i \geq 0$ and $j > 0$, is given by $\sum_{k=0}^{i} (-1)^k \binom{k+j-1}{j-1} z^{i-k} \frac{f^{(i-k)}}{(i-k)!}$.

Proof: Items (i) and (ii) can be found, e.g., in Chapter 7 of [6].

The third part follows from (i) and the following equality:

$$\binom{n-j}{i} = \sum_{k=0}^{i} (-1)^k \binom{n}{i-k} \binom{k+j-1}{j-1},$$

which holds for every $n, i \geq 0$ and $j > 0$. This equality can be easily derived from two equalities known as "upper negation" and "Vandermond convolution", which can be found in Table 174 of [6]. □

Now we are ready to provide a recurrence for functions a_i.

Theorem 4 *The following equations are valid:*

$$\begin{aligned}
a_1 &= z a_1^2 + 1, \quad a_1(0) = 1 \\
a_2 &= z a_1 + 2 z a_1 a_2 \\
a_i &= z^{i-1} \frac{a_1^{(i-2)}}{(i-2)!} + z^{i-2} \frac{a_1^{(i-3)}}{(i-3)!} + z^{i-2} \frac{a_2^{(i-3)}}{(i-3)!} \\
&\quad + z \cdot \sum_{j=1}^{i-3} \sum_{k=0}^{i-3-j} (-1)^k \binom{k+j-1}{j-1} z^{i-3-j-k} \frac{a_{j+2}^{(i-3-j-k)}}{(i-3-j-k)!} \\
&\quad + z \cdot \sum_{j=1}^{i} a_j a_{i-j+1}, \qquad \text{for } i > 2.
\end{aligned}$$

Proof: All these equations follow from Lemma 2 and Fact 3. □

$$
\begin{aligned}
a_1(z) &= \left(\frac{1}{2} - \frac{(1-4z)^{1/2}}{2}\right) z^{-1} \\
a_2(z) &= -\frac{1}{2} + \frac{1}{2\,(1-4z)^{1/2}} \\
a_3(z) &= \left(\frac{1}{1-4z} + \frac{z}{(1-4z)^{3/2}}\right) z \\
a_4(z) &= \left(\frac{3}{(1-4z)^2} + \frac{z}{(1-4z)^{5/2}}\right) z^2 \\
a_5(z) &= \left(\frac{4z+9}{(1-4z)^3} + \frac{z^2-19z+5}{(1-4z)^{7/2}}\right) z^3 \\
a_6(z) &= \left(\frac{24z+31}{(1-4z)^4} + \frac{3z^2-203z+51}{(1-4z)^{9/2}}\right) z^4 \\
a_7(z) &= \left(\frac{16z^2-128z+181}{(1-4z)^5} + \frac{2z^3-194z^2-1541z+398}{(1-4z)^{11/2}}\right) z^5
\end{aligned}
$$

Figure 3: The generating functions for the coefficients of the polynomials $P_n(m)$

Notice that the a_i's can be computed by induction. Indeed a_i occurs twice in the lefthand side of the last equation and we have:

$$
\begin{aligned}
a_i(1-2a_1) &= z^{i-1}\frac{a_1^{(i-2)}}{(i-2)!} + z^{i-2}\frac{a_1^{(i-3)}}{(i-3)!} + z^{i-2}\frac{a_2^{(i-3)}}{(i-3)!} \\
&\quad + z\cdot\sum_{j=1}^{i-3}\sum_{k=0}^{i-3-j}(-1)^k\binom{k+j-1}{j-1}z^{i-3-j-k}\frac{a_{j+2}^{(i-3-j-k)}}{(i-3-j-k)!} \\
&\quad + z\cdot\sum_{j=2}^{i-1} a_j a_{i-j+1}.
\end{aligned}
$$

Since $1-2a_1 = \sqrt{1-4z}$ we get:

$$
a_i = \left(\begin{array}{l}
z^{i-1}\frac{a_1^{(i-2)}}{(i-2)!} + z^{i-2}\frac{a_1^{(i-3)}}{(i-3)!} + z^{i-2}\frac{a_2^{(i-3)}}{(i-3)!} \\
+z\cdot\sum_{j=1}^{i-3}\sum_{k=0}^{i-3-j}(-1)^k\binom{k+j-1}{j-1}z^{i-3-j-k}\frac{a_{j+2}^{(i-3-j-k)}}{(i-3-j-k)!} \\
+z\cdot\sum_{j=2}^{i-1} a_j a_{i-j+1}
\end{array}\right) / \sqrt{1-4z}. \qquad (\dagger)
$$

Corollary 5 *Exact formulas for the functions a_1–a_7 are given in Figure 3.*

Proof: Let us first compute the function a_1 which, according to Theorem 4, is given by

$$a_1 = z a_1^2 + 1, \quad a_1(0) = 1.$$

By solving this equation, we obtain $a_1(z) = \frac{1-\sqrt{1-4z}}{2z}$, which is exactly the generating function for Catalan numbers—see, e.g., Chapter I.1 of [5].

Now, let us notice that on the basis of Theorem 4 all the other functions can be immediately obtained by tedious, however elementary, computations. In order to get exact values we applied SAGE software [15]. □

Let $[z^n]f(z)$ denote the n-th coefficient of z^n in the formal power series $f(z) = \sum_{n=0}^{\infty} f_n z^n$. The theorem below (Theorem VI.1 of [5]) serves as a powerful tool that allows us to estimate coefficients of certain functions that frequently appear in combinatorial considerations.

Fact 6 *Let α be an arbitrary complex number in $\mathbb{C} \setminus \mathbb{Z}_{\leq 0}$. The coefficient of z^n in*

$$f(z) = (1-z)^{\alpha}$$

admits the following asymptotic expansion:

$$[z^n]f(z) \sim \frac{n^{\alpha-1}}{\Gamma(\alpha)}\left(1 + \frac{\alpha(\alpha-1)}{2n} + \frac{\alpha(\alpha-1)(\alpha-2)(3\alpha-1)}{24n^2} + \frac{\alpha^2(\alpha-1)^2(\alpha-2)(\alpha-3)}{48n^3} + O\left(\frac{1}{n^4}\right)\right),$$

where Γ is the Euler Gamma function defined for $\Re(\alpha) > 0$ as

$$\Gamma(\alpha) := \int_0^{\infty} e^{-t}t^{\alpha-1}dt.$$

We can prove the following approximation.

Proposition 7

$$a_i(z) \sim \frac{C_{i-2}}{2^{3i-5}(1-4z)^{(2i-3)/2}} \qquad \text{when } z \to \frac{1}{4}$$

where C_i is the i^{th} Catalan number.

Proof: In this proof, when we write $\sim$ or "is of order" we mean when $z \to \frac{1}{4}$. We prove the result by induction using Theorem 4. The result is true for $i = 1$. For $i > 1$ and $j \leq i$, assume that $a_j(z)$ is of order $\frac{1}{(1-4z)^{(2j-3)/2}}$ and look at equation (†) to prove that $a_{i+1}(z)$ is of order $\frac{1}{(1-4z)^{(2i-1)/2}}$

Notice that the i^{th} derivative of a_1 is of order $\frac{1}{(1-4z)^{2i-1/2}}$, hence its $(i-2)^{th}$ is of order $\frac{1}{(1-4z)^{(2i-5)/2}}$ and its $(i-3)^{th}$ derivative is of order $\frac{1}{(1-4z)^{(2i-7)/2}}$. Similarly the i^{th} derivative of a_2 is of order $\frac{1}{(1-4z)^{2i+1/2}}$, hence its $(i-3)^{th}$ derivative is of order $\frac{1}{(1-4z)^{(2i-5)/2}}$.

By induction for $j+2 \leq i-3$, a_{j+2} is of order $\frac{1}{(1-4z)^{(2j+1)/2}}$. Among its successive derivative, one derives at most $i-3-j$ times, hence the items in the sum are of order at most $\frac{1}{(1-4z)^{(2i-5)/2}}$.

Hence the four first terms in (†) do not contribute to the asymptotic value of $a_{i+1}(z)$. Therefore the contribution to the asymptotic value is given by the

product $a_j a_{i-j+1}$'s, which are of order $\frac{1}{(1-4z)^{i-2}}$ and when multiplied by $\frac{1}{\sqrt{1-4z}}$, the last sum is of order $\frac{1}{(1-4z)^{(2i-1)/2}}$.

Let us call K_i the multiplicative coefficient $C_{i-2}/2^{3i-5}$ of $\frac{1}{(1-4z)^{(2j-3)/2}}$. One notices that $K_2 = \frac{1}{2} = \frac{C_0}{2^{3.2-5}}$. The sum $z \ \sum_{j=2}^{i-1} a_j a_{i-j+1}$ shows the inductive part. Indeed when $z = \frac{1}{4}$:

$$\begin{aligned} z \ \sum_{j=2}^{i-1} K_j K_{i-j+1} &= \frac{1}{4} \ \sum_{j=2}^{i-1} \frac{C_{j-2}}{2^{3j-5}} \ \frac{C_{i-j+1-2}}{2^{3(i-j+1)-5}} \\ &= \frac{1}{2^{3i-5}} \ \sum_{j=0}^{i-3} C_j C_{i-j-3} \\ &= \frac{C_{i-2}}{2^{3i-5}} \ = \ K_i. \end{aligned}$$

□

Theorem 8

$$[z^n]a_k(z) \ = \ \frac{1}{2^{k-1}(k-1)!\sqrt{\pi}} \ 4^n n^{(2k-5)/2} \ . \ \Psi(n,k)$$

where

$$\Psi(n,k) \ = \ 1 + \frac{(2k-3)(2k-5)}{8n} + \frac{(2k-3)(2k-5)(2k-7)(3k-11)}{384n^2} + \frac{(2k-3)^2(2k-5)^2(2k-7)(2k-9)}{3672n^3} + O(\frac{1}{n^4}).$$

Proof: First recall that:

$$\Gamma((2k-3)/2) = \Gamma((k-2)+\frac{1}{2}) = \frac{(2(k-2))!\sqrt{\pi}}{2^{2(k-2)}(k-2)!}.$$

Now using Fact 6, we can compute the principal part:

$$\begin{aligned} [z^n]a_k(z) &= \frac{C_{k-2}}{2^{3k-5}} \ 4^n \ [z^n](1-z)^{(2k-3)/2} \\ &\sim \frac{C_{k-2}}{2^{3k-5}} \ 4^n \ \frac{n^{(2k-5)/2}}{\Gamma((2k-3)/2)} \\ &= \frac{C_{k-2}}{2^{3k-5}} \frac{(k-2)!2^{2(k-2)}}{(2(k-2))!\sqrt{\pi}} \ 4^n n^{(2k-5)/2} \\ &= \frac{C_{k-2}(k-2)!}{2^{k-1}(2(k-2))!\sqrt{\pi}} \ 4^n n^{(2k-5)/2} \\ &= \frac{1}{2^{k-1}(k-1)!\sqrt{\pi}} \ 4^n n^{(2k-5)/2}. \end{aligned}$$

For $\Psi(n,k)$ we use Fact 6, with $\alpha = \frac{2k-3}{2}$. □

By looking at Figure 3, we can easily notice a recurring pattern concerning the structure of functions a_i. Therefore, we state the following proposition.

Proposition 9 *For every $i > 2$ we have*

$$a_i(z) = z^{i-2}\left(\frac{Q_i(z)}{(1-4z)^{i-2}} + \frac{R_i(z)}{(1-4z)^{i-\frac{3}{2}}}\right),$$

where Q_i and R_i are polynomials over $\mathbb{Z}$ in z and $\deg Q_i = \left\lfloor \frac{i-3}{2} \right\rfloor$ and $\deg R_i = \left\lfloor \frac{i-1}{2} \right\rfloor$.

> Proof: By induction using formula (†), on the same vein as the proof of Proposition 7. In particular, the two first members of (†) are derivatives of the generating function of Catalan numbers studied in [9]. □

As we have already mentioned, the number of closed terms of size n is given by $P_n(0)$, which corresponds to the n-th term of the Taylor expansion of the function a_{n+2}. Hence, the sequence of the numbers of closed lambda terms is equal to the sequence $([z^n]a_{n+2}(z))_{n\geq 0}$. From Proposition 9, the number of closed terms of size n is equal to $Q_{n+2}(0) + R_{n+2}(0)$. Currently, we have no recursive formula for the Q_n's and the R_n's. However from Proposition 7, we know that

$$R_{n+2}\left(\frac{1}{4}\right) = \frac{C_n}{2^{n+1}}.$$

6 Counting normal forms

Beside counting terms, one can also count normal forms. To this end, we describe the set of normal forms as follows

$$\begin{aligned} \mathcal{G}_m &= \mathcal{I}(m) \uplus \mathcal{G}_m @ \mathcal{F}_m \\ \mathcal{F}_m &= \lambda\, \mathcal{F}_{m+1} \uplus \mathcal{G}_m \end{aligned}$$

Recall that a normal form is made by a sequence of abstractions on terms which is a variable (a de Bruijn index) applied to a sequence of normal forms. $\mathcal{F}_m$ represents the normal forms and $\mathcal{G}_m$ represents the terms starting with an index. From this we derive the formulas for counting:

$$\begin{aligned} G_{0,m} &= m \\ G_{n+1,m} &= \sum_{k=0}^{n} G_{n-k,m} F_{k,m} \end{aligned}$$

$$\begin{aligned} F_{0,m} &= m \\ F_{n+1,m} &= F_{n,m+1} + G_{n+1,m} \end{aligned}$$

Like for terms we derive polynomials:

$$\begin{aligned} {}^{\mathsf{NF}}P_0(m) &= m \\ {}^{\mathsf{NF}}P_{n+1}(m) &= {}^{\mathsf{NF}}P_n(m+1) + {}^{\mathsf{NF}}Q_{n+1}(m) \end{aligned}$$

$$\begin{aligned} {}^{\mathsf{NF}}Q_0(m) &= m \\ {}^{\mathsf{NF}}Q_{n+1}(m) &= \sum_{k=0}^{n} {}^{\mathsf{NF}}P_k(m)\,{}^{\mathsf{NF}}Q_{n-k}(m). \end{aligned}$$

Lemma 10 *For every n, the degree of the polynomials ${}^{\mathsf{NF}}P_n$ and ${}^{\mathsf{NF}}Q$ is equal to $n+1$.*

Proof: Like the proof of Lemma 1, by induction on n from the definition of ${}^{\mathsf{NF}}P_n$ and ${}^{\mathsf{NF}}Q$. □

We have not derived the formulas for the coefficients yet. But these formulas are useful to derive generators of normal forms used in the rest of the paper.

7 Lambda term generation

From the simple equations defining the number $T_{n,m}$ of terms we can define a function generating them. More precisely, we define a function $T(k,n,m)$ which returns the k^{th} term of size n with at most m variables (see the program in Figure 4). The integer k belongs to the interval $[1..P_n(m)]$ which requires to handle big numbers. This program can be used to enumerate all the λ-terms of size n with at most m distinct free variables. This is appropriate only for small values, since the number of λ-terms is very large. But overall, in order to generate a random term of size n with at most m distinct free variables, it suffices to feed T with a random value k in the interval $[1..P_n(m)]$. Similarly, one can define from the recursive formula for the number of normal forms a program for their generation.

8 Simply typed terms

Once we have a random generator for untyped terms, it is easy to build a random generator for simply typed terms. It suffices to sieve the plain terms by a predicate, which we call *is_typable*. This predicate, which was implemented in SAGE [15] (i.e., in Python) like the rest of the programs, is a classical principal type algorithm [11, 2, 7]. For instance, applying the random generator with parameter 10 (for the size of the term), we got:

$$\lambda(\lambda(((1\ \lambda(1))\ \lambda((3\ \lambda(((1\ 2)\ 3)))))))$$

This is a "typical" simply typed random closed lambda term of size 10 written with de Bruijn indices. Its type is

$$\begin{gathered}((\alpha \to (((\beta \to \beta) \to (\alpha \to \gamma) \to \delta) \to \zeta)) \to \zeta) \to \gamma \to \\ ((\beta \to \beta) \to (\alpha \to \gamma) \to \delta) \to \delta\end{gathered}$$

```
Term(k,n,m) :
   if  n = 0: return k
   elif k ≤ P(n-1)(m+1): return λTerm(k,n-1,m+1)
   else:
      j := 0
      h := k - P(n-1)(m+1)
      while True:
         if h ≤ P(j)(m) * P(n-1-j)(m):
            if h mod P(n-1-j)(m) = 0 :
                return Term(h ÷ P(n-1-j)(m),j,m) @ Term(P(n-1-j)(m),
n-1-j,m)
             else:
                return Term(⌊h ÷ P(n-1-j)(m)⌋+1,j,m) @ Term((h mod
P(n-1-j)(m)), n-1-j,m)
          else:
             h := h - P(j)(m) * P(n-1-j)(m)
             j := j + 1
```

Figure 4: The program for term generation

We were able to generate terms of size 50 (or of size about 80 provided we work in the model in which the size of each variable is 1). For such terms, the generating process is slow, since it requires 50 000 generations of terms, with (unsuccessful) tests of their typability before getting a typed one. But for size 40, for size about 65 if one would count also the variables, the number of attempts falls at 1000, which is reasonable. However, according to the distribution given in Figure 9, if one accepts a bias toward terms starting with abstractions, the search is easier (see Section 9.5)

This kind of a random generator is useful for testing functional programs. Michał Pałka [12, 13] proposed a tool to debug Haskell compilers based on a lambda term generator. His generator is designed on the development of a typing tree, with choices made when a new rule is created. Such a method needs to cut branches in developing the tree to avoid loops. This way his generator is not random, which may be a drawback in some cases.

9 Experimental data

Given a random term generator, we are able to write programs to make statistics on some features of terms. Experiments recorded here have been performed on a laptop with a 2.4 GHz Intel Core i5 processor.

9.1 Average depth of variables in terms

Let us define the *depth of a variable* as the number of symbols (abstractions and applications) between this variable and the top of the term. For instance, given the term $\lambda x.(\lambda yz.x)(\lambda u.u)$, the variable x has depth 4, while the depth of u equals 3. In Figure 5,

we draw both the average depth of the variables for 300 throws of random terms of size 15 up to 175 (scatted plot) and the curve $\frac{2n}{\ln(n)}$ (plain line). We also provide the comparison between the average depth of variables in normal forms for 300 throws of normal forms of size 15 up to 175 (scatted plot) and the same curve $\frac{2n}{\ln(n)}$ (Figure 6). On this basis, we conjecture that the average depth of variables in terms has an asymptotic upper bound $\frac{2n}{\ln(n)}$.

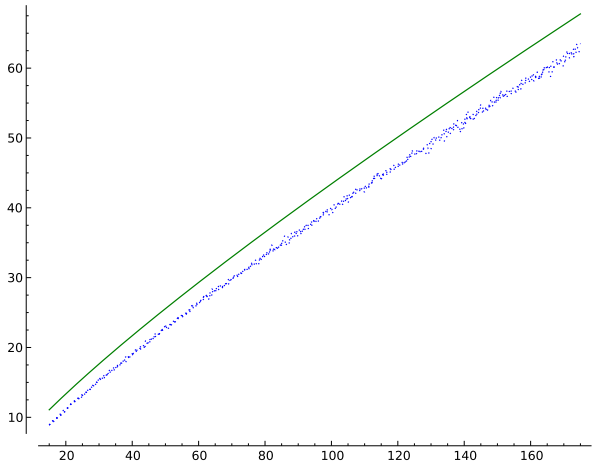

Figure 5: Average depth of variables and curve $\frac{2n}{\ln(n)}$

.

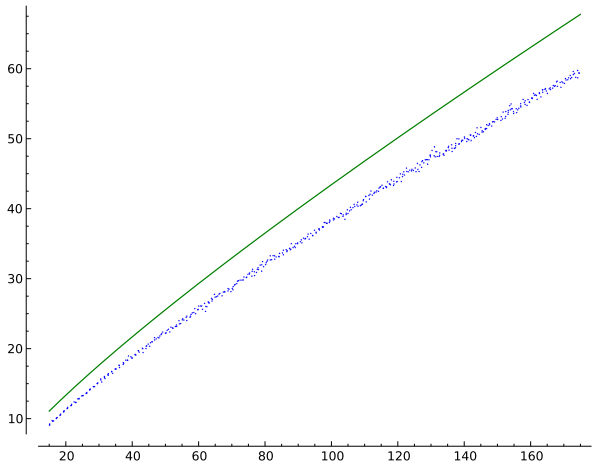

Figure 6: Average depth of variables in normal forms and curve $\frac{2n}{\ln(n)}$

.

9.2 Average number of head λ's in terms

We say that λx is a *head lambda* in a term t if the latter is of the form $\lambda x_1 \dots \lambda x_n \lambda x.s$ for some positive integer n and a certain term s. In order to know the structure of an average term, we are interested in the average number of head λ's occurring in terms. In Figure 7, we compare the average number of head λ's in 400 random terms of size

15 to 250 with $\frac{n}{\ln(n)}$. In Figure 8, we repeat this study in the case of random normal forms. Our experiments show that, as concern head λ's, terms and normal forms have approximatively the same shape, but the average number of head λ's is slightly larger in the case of all terms than in the case of normal forms.

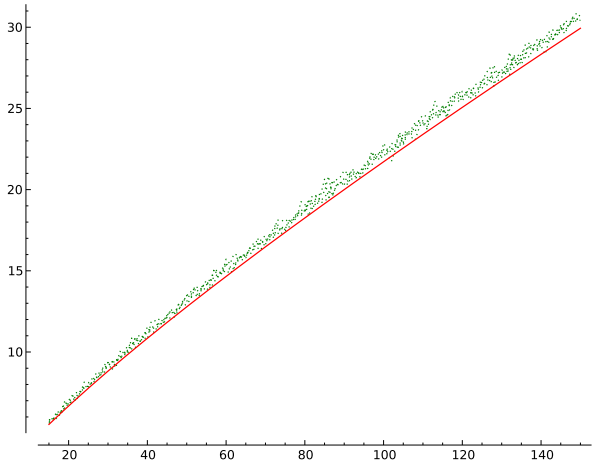

Figure 7: Average number of head λ's in terms and curve $\frac{n}{\ln(n)}$

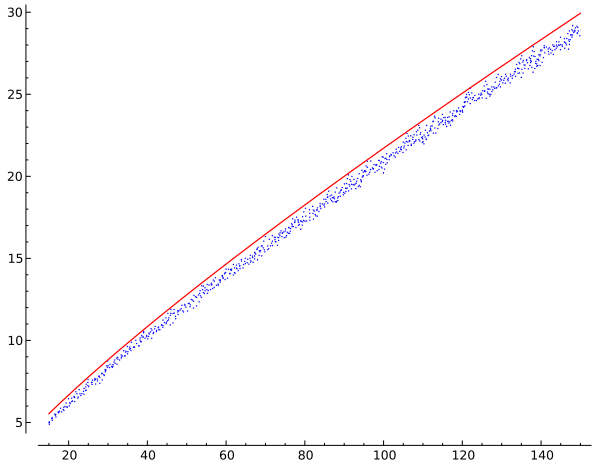

Figure 8: Average number of head λ's in normal forms and curve $\frac{n}{\ln(n)}$

9.3 Ratio of simply typed terms among terms

It is interesting to investigate the ratio of simply typed terms among untyped ones. Actually, there are 454 283 lambda terms of size 8, whereas there are 43 977 lambda terms of size 7. Therefore, in the case of our implementation, 7 is the upper limit for an exhaustive computation of this ratio. The array below gives the ratio of simply typed

size	8	9	10	11	12	13	14	15	16	20	30	40	45	50
ratio	.216	.178	.143	.111	.089	.073	.056	.047	.039	.0014	.0012	.0003	.00005	$<10^{-5}$

Table 1: Ratio of simply typed terms

size	8	9	10	11	12	13	14	15	16	20	30	40	45	50
ratio	.140	.108	.094	.068	.057	.048	.038	.029	.024	.0010	.0009	.00006	$<10^{-5}$	$<10^{-5}$

Table 2: Ratio of simply typed normal forms

terms over plain terms by an exhaustive examination of the terms up to 7.

size	4	5	6	7
nb of terms	82	579	4 741	43 977
nb of typed terms	40	238	1 564	11 807
ratio	0.4878	0.4110	0.3299	0.2684

After 8, we computed the ratio by the Monte Carlo method. The results are given in Table 1.

We conclude that simply typed terms become very rare as the size of the terms grows, falling at less than one over 10000 when the size gets larger than 50. Like before, we have done the same task for normal forms. We got the ratio by an exhaustive examination of normal forms up to 7:

size	4	5	6	7
nb of terms	53	323	2 359	19 877
nb of typed terms	23	106	587	3 789
ratio	0.434	0.328	0.249	0.190

and by the Monte Carlo method thereafter (see Table 2).

9.4 Distribution of simply typed lambda terms among terms

We said that simply typed terms are rare, but we may wonder what rare means exactly. More precisely we may wonder how terms are distributed. To provide an answer to this question, we realized experiments for approaching the distribution of the frequency of typed lambda terms in segments of the interval $[1..P_n(0)]$. For that we divided the interval $[1..P_n(0)]$ in segments and we computed on samples of randomly thrown terms, the ratio of simply typed terms over general terms we may expect in each segment. Figure 9 is typical of the results we got. This corresponds to an experiment on terms of size 25 on 250 segments with tests for simple typability on 200 random terms in each segment. It shows that the simply typed terms are not evenly distributed. They are more concentrated on the left of the interval corresponding to terms with low numbers. Those terms correspond to terms starting rather with abstractions than with applications and this is recursively so for subterms giving this impressions of rolling waves. For instance, there are 2% to 3% of typable terms (of size 25) starting with many abstractions, whereas for terms starting with many applications, there are large subintervals with almost no typable terms. Figure 10 which gives the same statistics for terms of size 30 shows that typed terms gets more rare as the size of the terms grows.

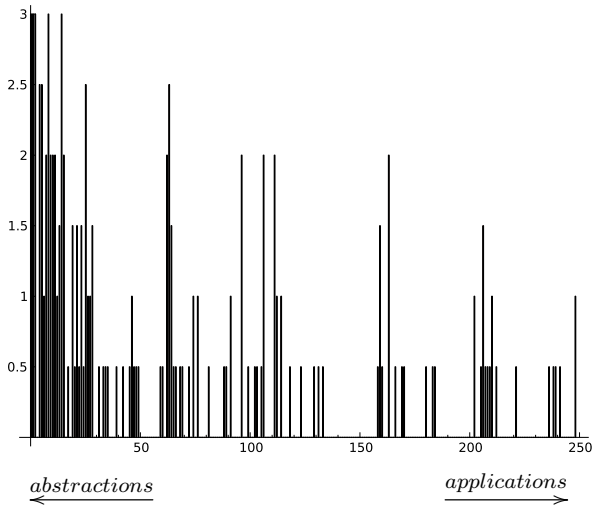

Figure 9: Distribution of simply typed lambda terms of size 25. 250 segments on the horizontal axis, percentage (0% – 3%) of typable terms in segments on the vertical axis.

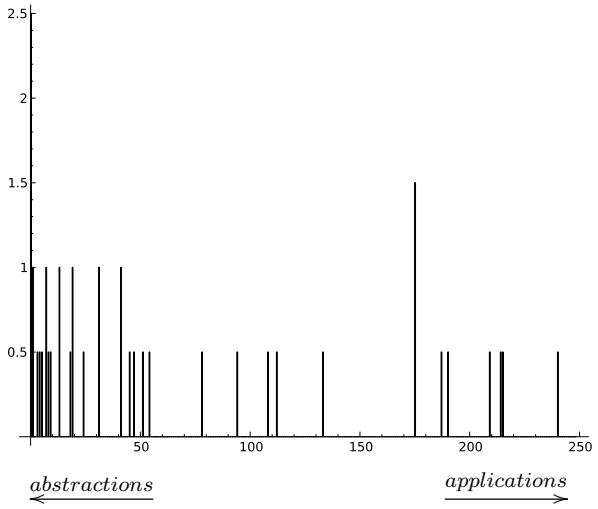

Figure 10: Distribution of simply typed lambda terms of size 30. 250 segments on the horizontal axis, percentage (0% – 2.5%) of typable terms in segments on the vertical axis.

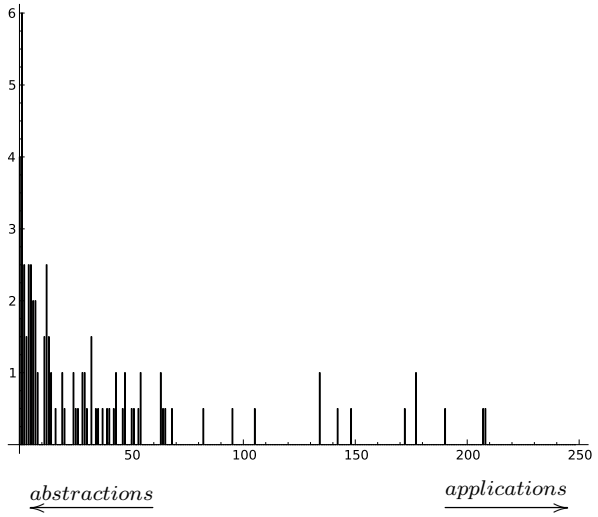

Figure 11: Distribution of simply typed normal forms of size 25. 250 segments on the horizontal axis, percentage (0% – 6%) of typable normal forms in segments on the vertical axis.

The normal forms are even more scarcely distributed. As a comparison, we drew the same graphs for normal forms (size of the normal forms: 25 and 30, number of segments 250, tests on 200 terms) in Figure 11. The typable normal forms aggregate more on the left of the interval where terms start mostly with abstractions, with peaks of 4% to 6% by segments. Figure 12 shows that scarcity of typed normal forms increases as the size of terms grow. .

9.5 Biased generation

If we renounce full randomness, the distribution of simply typed terms provides us with a clue for getting large typed terms. We propose to call this a *biased generation*. This may be convenient if we look for a closed term of large size, not necessarily random. For that we search for term numbers in the low part of the interval $[1..P_n(0)]$. In experiments, we have chosen for instance the first subinterval of size $P_n(0)/2^n$. This way we were able to generate large closed simply typed lambda terms of size up to 200. With no surprise we got terms which are clearly not random, they have a third of the symbols as head λ's and the rest as applications, that is that terms thrown that way have a very specific shape as they are sequences of abstractions followed by a sequence of applications.

10 Acknowledgments

We would like to thank Marek Zaionc for interesting discussions and for setting the problem of counting lambda terms and Bruno Salvy for his help in the proof of Proposition 7.

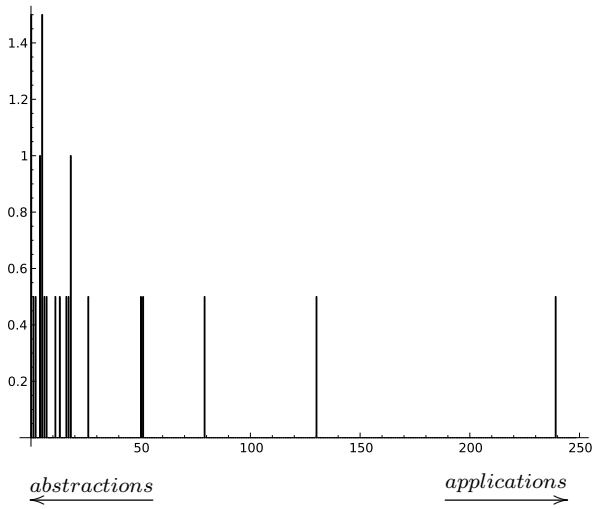

Figure 12: Distribution of simply typed normal forms of size 30. 250 segments on the horizontal axis, percentage (0% – 1.45%) of typable normal forms in segments on the vertical axis.

11 Conclusion

The results we obtained open many tracks of research. We have to know more about the polynomials Q_i and R_i in Conjecture 9. Here we have considered variables of weight 0, because it is simpler but still challenging and informative. A model with variable of weight 1 is worth studying and being compared with one presented here. This has been initiated in [10] but the generation of random terms has not been considered. Moreover we have considered simple types (almost no term is simply typed). In further research we plan to focus on other type systems, e.g., system F. But in this case, counting methods will not apply, since typing is undecidable [18]. Perhaps like in [3], a method based on upper and lower approximations of the numbers of terms may apply. From those results we may expect to say something about beta reduction and its average efficiency in the untyped case as in the typed case.

Lemma 2 gives recurrence formulas for the coefficients. We may exploit those results to derive a bivariate generating function for the coefficients p_n^i's and then a formula or at least an asymptotic value of the p_n^{n+2}'s which are the numbers of closed terms of size n. Finally, we study plain lambda terms, but it seems straightforward to study terms with specific types like nat and specific constants like $suc : nat \to nat$. This can be extended to languages with variable scopes, not necessarily functional programming languages.

References

[1] Olivier Bodini, Danièle Gardy, and Bernhard Gittenberger. Lambda terms of bounded unary height. In *Proceedings of the Eighth Workshop on Analytic Algorithmics and Combinatorics*, pages 23–32, 2011.

[2] Luís Damas and Robin Milner. Principal type-schemes for functional programs. In *POPL*, pages 207–212. ACM Press, 1982. Richard A. DeMillo ed.

[3] René David, Katarzyna Grygiel, Jakub Kozik, Christophe Raffalli, Guillaume Theyssier, and Marek Zaionc. Asymptotically almost all λ-terms are strongly normalizing. *CoRR*, abs/0903.5505v3, 2009.

[4] Nicolaas Govert de Bruijn. Lambda calculus with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Proc. Koninkl. Nederl. Akademie van Wetenschappen*, 75(5):381–392, 1972.

[5] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2008.

[6] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete mathematics*. Addison-Wesley, Reading, MA, 1989.

[7] J. Roger Hindley. M. H. Newman's typability algorithm for lambda-calculus. *J. Log. Comput.*, 18(2):229–238, 2008.

[8] Donald E. Knuth. *The art of Computer Programming, Generating All Trees-History of Combinatorial Generation*, volume 4 (fascicle 4). Addison-Wesley Publishing Company, 2006.

[9] Wolfdieter Lang. On polynomials related to derivatives of the generative functions of the Catalan numbers. *The Fibonacci Quarterly*, 40(4):299–313, 2002.

[10] Pierre Lescanne. On counting untyped lambda terms. *CoRR*, abs/1107.1327, 2011. to be published in *Theoretical Computer Science*.

[11] M. H. A. Newman. Stratified systems of logic. *Proceedings of the Cambridge Philosophical Society*, 39:69–83, 1943.

[12] Michał Pałka. Testing an optimising compiler by generating random lambda terms. Licentiatavhandling, Department of Computer Science and Engineering, Chalmers University of Technology and Göteborg University, may 2012.

[13] Michał H. Pałka, Koen Claessen, Alejandro Russo, and John Hughes. Testing an optimising compiler by generating random lambda terms. In *Proceedings of the 6th International Workshop on Automation of Software Test*, AST'11, pages 91–97, New York, NY, USA, 2011. ACM.

[14] Alexey Rodriguez Yakushev and Johan Jeuring. Enumerating well-typed terms generically. In Ute Schmid, Emanuel Kitzelmann, and Rinus Plasmeijer, editors, *AAIP*, volume 5812 of *Lecture Notes in Computer Science*, pages 93–116. Springer, 2009.

[15] W. A. Stein et al. *Sage Mathematics Software (Version 4.8)*. The Sage Development Team, 2012. `http://www.sagemath.org`.

[16] The PARI Group. *PARI/GP, version* `2.5.0`. Bordeaux, 2011. available from `http://pari.math.u-bordeaux.fr/`.

[17] Jue Wang. The efficient generation of random programs and their applications. Master's thesis, Honors Thesis, Wellesley College, Wellesley, MA, May 2004.

[18] J. B. Wells. Typability and type-checking in the second-order lambda-calculus are equivalent and undecidable. In *LICS*, pages 176–185. IEEE Computer Society, 1994.

A Number of terms with exactly m distinct free variables

In the rest of paper we were mostly interested by the numbers of terms with at most m free variables. Here we study the numbers of terms with exactly m distinct free variables, the formulas for counting those numbers and their relations with quantities we considered.

A.1 A formula

Let us show how to derive the formula for counting λ-terms with exactly m distinct free variables. This formula is adapted from a similar one when variables have weight 1 due to Raffalli (*On-line Encyclopedia of Integer Sequences* under number **A135501**). We assume that terms are built of usual variables (not de Bruijn indices) and that they are equivalent up to a renaming free variables and up to α-conversion. Let us denote the number of λ-terms of size n with exactly m distinct free variables by $f_{n,m}$.

Notice first that there is no term of size 0 with no free variable, hence $f_{0,0} = 0$. There is one term of size 0 with one free variable, namely x, up to a renaming of the variables, hence $f_{0,1} = 1$. The maximum number of variables for a λ-term of size n is when the only operators are applications and all the variables are different. One has then a binary tree with n interior nodes and $n+1$ leaves holding $n+1$ variables. This means that for m beyond $n+1$ variables there is no term of size n with exactly m distinct free variables. Hence

$$f_{n,m} = 0 \qquad \text{when } m > n+1.$$

In the general case, a term of size $n+1$ with m free variables starts either with an abstraction or with an application. Terms starting with an abstraction, say λx, on a term M contribute in two ways, either M does not contain x as a free variables or M contains x as a free variable. There are $f_{n,m}$ such M's in the first case and $f_{n,m+1}$ in the second. This gives the two first summands $f_{n,m} + f_{n,m+1}$ of the formula. Let us look how terms starting with an application look like. Assume they are of the form $N\,P$ and of size $n+1$. For some $p \leq n$, the term N is of size p and P is of size $n-p$. Both these terms share c common variables ($0 \leq c \leq m$), while $N\,P$ has m distinct free variables. N has k distinct free variables, which do not occur in P, hence N has $k+c$ distinct free variables altogether. The term P has $m-k$ distinct free variables. Therefore, given a set of own variables for N, a set of common variables, and a set of own variables for P, there are $f_{p,k+c}f_{n-p,m-k}$ possible pairs (N,P). There are $\binom{m}{c}$ ways to choose the c common variables among m and there are $\binom{m-c}{k}$ ways to split the remaining variables into N and

P, namely k for N and $m-c-k$ for P, hence the third summand of the formula:

$$\sum_{p=0}^{n}\sum_{c=0}^{m}\sum_{k=0}^{m-c}\binom{m}{c}\binom{m-c}{k}f_{p,k+c}f_{n-p,m-k}.$$

Now, we obtain the whole formula:

$$f_{n+1,m} \;=\; f_{p,k+c}f_{n-p,m-k} + \sum_{p=0}^{n}\sum_{c=0}^{m}\sum_{k=0}^{m-c}\binom{m}{c}\binom{m-c}{k}f_{p,k+c}f_{n-p,m-k}.$$

A.2 Relations between $T_{n,m}$ and $f_{n,m}$

The number of terms of size n with exactly i indices in $[1..m]$ is $\binom{m}{i}f_{n,i}$. Therefore the number of terms with at most indices in $[1..m]$ is:

$$T_{n,m} \quad = \quad \sum_{i=0}^{m}\binom{m}{i}f_{n,i}.$$

By the inversion formula ([6] p. 192), we get:

$$f_{n,m} \quad = \quad \sum_{i=0}^{m}(-1)^{m+i}\binom{m}{i}T_{n,i}.$$

This shows the non surprising fact that $f_{n,m}$ and $T_{n,m}$ are simply connected. Knowing that the $T_{n,m}$'s can be easily computed, this provides a formula simpler that Raffalli's to compute the $f_{n,m}$'s.