



# Quantitative aspects of linear and affine closed lambda terms

Pierre Lescanne

## ► To cite this version:

Pierre Lescanne. Quantitative aspects of linear and affine closed lambda terms. 2017. ensl-01464047v2

**HAL Id: ensl-01464047**

**<https://ens-lyon.hal.science/ensl-01464047v2>**

Preprint submitted on 14 Feb 2017 (v2), last revised 21 May 2017 (v5)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Quantitative aspects of linear and affine closed lambda terms

Pierre Lescanne  
University of Lyon  
École normale supérieure de Lyon  
LIP (UMR 5668 CNRS ENS Lyon UCBL INRIA)  
46 allée d'Italie, 69364 Lyon, France

pierre.lescanne@ens-lyon.fr

February 14, 2017

## Abstract

Affine  $\lambda$ -terms are  $\lambda$ -terms in which each bound variable occurs at most once and linear  $\lambda$ -terms are  $\lambda$ -terms in which each bound variables occurs once. and only once. In this paper we count the number of closed affine  $\lambda$ -terms of size  $n$ , closed linear  $\lambda$ -terms of size  $n$ , affine  $\beta$ -normal forms of size  $n$  and linear  $\beta$ -normal forms of sizes  $n$ , for different ways of measuring the size of  $\lambda$ -terms. From these formulas, we show how we can derive programs for generating all the terms of size  $n$  for each class. The foundation of all of this is specific data structures, which are contexts in which one counts all the holes at each level of abstractions by  $\lambda$ 's.

**Keywords:** Lambda calculus, combinatorics, functional programming

## 1 Introduction

The  $\lambda$ -calculus [1] is a well known formal system designed by Alonzo Church [6] for studying the concept of function. It has three kinds of basic operations: variables, application and abstraction (with an operator  $\lambda$  which is a binder of variables). We assume the reader familiar with the  $\lambda$ -calculus and with de Bruijn indices.<sup>1</sup>

In this paper we are interested in terms in which bound variables occur once. A *closed  $\lambda$ -term* is a  $\lambda$ -term in which there is no free variable. An *affine  $\lambda$ -term* is a  $\lambda$ -term in which bound variables occur at most once. A *linear  $\lambda$ -term* is a  $\lambda$ -term in which bound variables occur once and only once

In this paper we propose a method for counting and generating (including random generation) linear and affine closed  $\lambda$ -terms based on a data structure which we call *SwissCheese* because of its holes. Actually we count those  $\lambda$ -terms up-to  $\alpha$ -conversion. Therefore it is adequate to use de Bruijn indices [9], because a term with de Bruijn indices represents an  $\alpha$ -equivalence class. An interesting aspect of these terms is the fact that they are simply typed [14, 13]. For instance, generated by the program of Section 5, there are 16 linear terms of natural size 8:

$(\lambda 0 (\lambda 0 \lambda 0)) \quad (\lambda 0 \lambda (\lambda 0 0)) \quad (\lambda 0 \lambda (0 \lambda 0)) \quad ((\lambda 0 \lambda 0) \lambda 0) \quad (\lambda (\lambda 0 0) \lambda 0) \quad (\lambda (0 \lambda 0) \lambda 0) \quad \lambda (\lambda 0 (\lambda 0 0)) \quad \lambda (\lambda 0 (0 \lambda 0))$   
 $\lambda ((\lambda 0 \lambda 0) 0) \quad \lambda (\lambda (\lambda 0 0) 0) \quad \lambda (\lambda (0 \lambda 0) 0) \quad \lambda (0 (\lambda 0 \lambda 0)) \quad \lambda (0 \lambda (\lambda 0 0)) \quad \lambda (0 \lambda (0 \lambda 0)) \quad \lambda ((\lambda 0 0) \lambda 0) \quad \lambda ((0 \lambda 0) \lambda 0)$   
written with explicit variables

$\lambda x.x (\lambda x.x \lambda x.x) \quad \lambda x.x \lambda y.(\lambda x.x y) \quad \lambda x.x \lambda y.(y \lambda x.x) \quad (\lambda x.x \lambda x.x) \lambda x.x$   
 $\lambda y.(\lambda x.x y) \lambda x.x \quad \lambda y.(y \lambda x.x) \lambda x.x \quad \lambda y.(\lambda x.x (\lambda x.x y)) \quad \lambda y.(\lambda x.x (y \lambda x.x))$

---

<sup>1</sup>If the reader is not familiar with the  $\lambda$ -calculus, we advise him to read the introduction of [12].

$\lambda y.((\lambda x.x \lambda x.x) y) \quad \lambda y.(\lambda z.(\lambda x.x z) y) \quad \lambda y.(\lambda z.(z \lambda x.x) y) \quad \lambda y.(y (\lambda x.x \lambda x.x))$   
 $\lambda y.(y \lambda z.(\lambda x.x z)) \quad \lambda y.(y \lambda z.(z \lambda x.x)) \quad \lambda y.((\lambda x.x y) \lambda x.x) \quad \lambda y.((y \lambda x.x) \lambda x.x)$

and there are 25 affine terms of natural size 7:

$(\lambda 0 \lambda \lambda 1) \quad (\lambda 0 \lambda \lambda \lambda 0) \quad (\lambda \lambda 0 \lambda \lambda 0) \quad (\lambda \lambda 1 \lambda 0) \quad (\lambda \lambda \lambda 0 \lambda 0) \quad \lambda(\lambda \lambda 1 0) \quad \lambda(\lambda \lambda \lambda 0 0) \quad \lambda(0 \lambda \lambda 1)$   
 $\lambda(0 \lambda \lambda \lambda 0) \quad \lambda(\lambda 0 \lambda \lambda 1) \quad \lambda(\lambda 1 \lambda 0) \quad \lambda \lambda(\lambda 0 1) \quad \lambda \lambda(1 \lambda 0) \quad \lambda(\lambda 0 \lambda \lambda 0) \quad \lambda(\lambda \lambda 0 \lambda 0) \quad \lambda \lambda(\lambda \lambda 0 0)$   
 $\lambda \lambda(0 \lambda \lambda 0) \quad \lambda \lambda \lambda(0 1) \quad \lambda \lambda \lambda(1 0) \quad \lambda \lambda \lambda \lambda 2 \quad \lambda \lambda(\lambda 0 \lambda 0) \quad \lambda \lambda \lambda(\lambda 0 0) \quad \lambda \lambda \lambda(0 \lambda 0) \quad \lambda \lambda \lambda \lambda 1 \quad \lambda \lambda \lambda \lambda \lambda 0$

The Haskell programs of this development are on GitHub: <https://github.com/PierreLescanne/CountingGeneratingAffineLinearClosedLambdaterms>.

## Notations

In this paper we use specific notations.

Given a predicate  $p$ , the Iverson notation written  $[p(x)]$  is the function taking natural values which is 1 if  $p(x)$  is true and which is 0 if  $p(x)$  is false.

Let  $\mathbf{m} \in \mathbb{N}^p$  be the  $p$ -tuple  $(m_0, \dots, m_{p-1})$ . In Section 5, we consider infinite tuples. Thus  $\mathbf{m} \in \mathbb{N}^\omega$  is the sequence  $(m_0, m_1, \dots)$ .

- $p$  is the *length* of  $\mathbf{m}$ , which we write also  $\text{length } \mathbf{m}$
- The  $p$ -tuple  $(0, \dots, 0)$  is written  $0^p$ .  $0^\omega$  is the infinite sequence made of 0's.
- The *increment* of a  $p$ -tuple at  $i$  is:

$$\mathbf{m}^{\uparrow i} = \mathbf{n} \in \mathbb{N}^p \text{ where } n_j = m_j \text{ if } j \neq i \text{ and } n_i = m_i + 1$$

- Putting an element  $x$  as *head* of a tuple is written

$$x : \mathbf{m} = x : (m_0, \dots) = (x, m_0, \dots)$$

tail removes the head of an tuple:

$$\text{tail}(x : \mathbf{m}) = \mathbf{m}.$$

- $\oplus$  is the componentwise addition on tuples.

## 2 SwissCheese

The basic concept is this of **m-SwissCheese** or **SwissCheese** if there is no ambiguity on  $\mathbf{m}$ . That is a  $\lambda$ -term with holes of  $p$  levels, which are all counted, using  $\mathbf{m}$ . The  $p$  levels of holes are  $\square_0, \dots, \square_{p-1}$ . A hole  $\square_i$  is meant to be a location for a variable at level  $i$ , that is under  $i$   $\lambda$ 's. According to the way bound variables are inserted when creating abstractions (see below), we consider linear or affine SwissCheeses. The holes have size 0. An  $\mathbf{m}$ -SwissCheese has  $m_0$  holes at level 0,  $m_1$  holes at level 1, ...  $m_{p-1}$  holes at level  $p$ . Let  $l_{n,\mathbf{m}}$  (resp.  $a_{n,\mathbf{m}}$ ) count the linear (resp. the affine)  $\mathbf{m}$ -SwissCheese of size  $n$ .  $l_{n,\mathbf{m}} = l_{n,\mathbf{m}'}$  and  $a_{n,\mathbf{m}} = a_{n,\mathbf{m}'}$  if  $\mathbf{m}$  is finite,  $\text{length } \mathbf{m} \geq n$ ,  $m_i = m'_i$  for  $i \leq \text{length } \mathbf{m}$ , and  $m'_i = 0$  for  $i > \text{length } \mathbf{m}$ .  $l_{n,0^n}$  (resp.  $a_{n,0^n}$ ) counts the closed linear (resp. the closed affine)  $\lambda$ -terms.

### 2.1 Growing a SwissCheese

Given two SwissCheeses, we can build a SwissCheese by application like in Fig 1. In Fig. 1,  $c_1$  is a  $(0, 1, 0, 0, 0)$ -SwissCheese,  $c_2$  is a  $(1, 1, 0, 0, 0)$ -SwissCheese and  $c_1@c_2$  is a  $(1, 2, 0, 0, 0)$ -SwissCheese.

Given a SwissCheese, there are two ways to grow a SwissCheese to make another SwissCheese by abstraction.

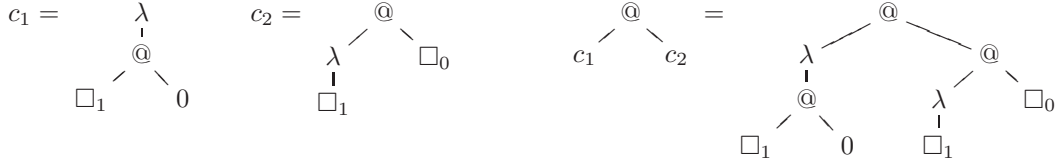


Figure 1: Building a SwissCheese by application

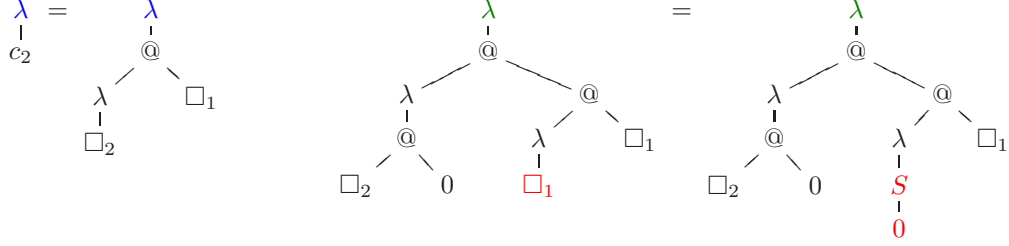


Figure 2: Abstracting SwissCheeses without and with binding

1. We put a  $\lambda$  on the top of a  $\mathbf{m}$ -SwissCheese  $c$ . This increases the levels of the holes: a hole  $\square_i$  becomes a hole  $\square_{i+1}$ .  $\lambda c$  is a  $(0 : \mathbf{m})$ -SwissCheese. See Fig 2 on the left. This way, no index is bound by the top  $\lambda$ , therefore this does not preserve linearity (it preserves affinity however). Therefore this construction is only for building affine SwissCheeses, not for building linear SwissCheeses. In Figure 2 (left), we colour the added  $\lambda$  in blue and we call it *abstraction with no binding*.
2. In the second method for growing a SwissCheese by abstraction, we select first a hole  $\square_i$ , we top the SwissCheese by a  $\lambda$ , we increment the levels of the other holes and we replace the chosen box by  $S^i 0$ . In Figure 2 (right), we colour the added  $\lambda$  in green and we call it *abstraction with binding*.

## 2.2 Measuring SwissCheese

We consider several ways of measuring the size of a SwissCheese derived from what is done on  $\lambda$ -terms. In all these sizes, applications  $@$  and abstractions  $\lambda$  have size 1 and holes have size 0. The differences are in the way variables are measured.

- Variables have size 0, we call this **variable size 0**.
- Variables have size 1, we call this **variable size 1**.
- Variables (or de Bruijn indices)  $S^i 0$  have size  $i + 1$ , we call this **natural size**.

## 3 Counting linear closed terms

We start with counting linear terms since they are slightly simpler. We will give recursive formulas first for the numbers  $l'_{n,\mathbf{m}}$  of linear SwissCheeses of natural size  $n$  with holes set by  $\mathbf{m}$ , then for the numbers  $l^0_{n,\mathbf{m}}$  of linear SwissCheeses of size  $n$ , for variable size 0, with holes set by  $\mathbf{m}$ , eventually for the numbers  $l^1_{n,\mathbf{m}}$  of linear SwissCheeses of size  $n$ , for variable size 1, with holes set by  $\mathbf{m}$ . When we do not want to specify a chosen size, we write only  $l_{n,\mathbf{m}}$  without superscript.

### 3.1 Natural size

First let us count linear SwissCheeses with natural size. This is given by the coefficient  $l^\nu$  which has two arguments: the size  $n$  of the SwissCheese and a tuple  $\mathbf{m}$  which specifies the number of holes of each level. In other words we are interested by the quantity  $l_{n,\mathbf{m}}$ . We assume that the length of  $\mathbf{m}$  is  $p$ , greater than  $n$ .

**$n = 0$**  whatever size is considered, there is only one SwissCheese of size 0 namely  $\square_0$ . This means that the number of SwissCheeses of size 0 is 1 if and only if  $\mathbf{m} = (1, 0, 0, \dots)$ :

$$l_{0,\mathbf{m}} = [m_0 = 1 \wedge \bigwedge_{j=1}^p m_j = 0]$$

**$n \neq 0$  and application** if a  $\lambda$ -term of size  $n$  has holes set by  $\mathbf{m}$  and is an application, then it is obtained from a  $\lambda$  term of size  $k$  with holes set by  $\mathbf{q}$  and a  $\lambda$  term of size  $n - k - 1$  with holes set by  $\mathbf{r}$ , with  $\mathbf{m} = \mathbf{q} \oplus \mathbf{r}$ :

$$\sum_{\mathbf{q} \oplus \mathbf{r} = \mathbf{m}} \sum_{k=0}^n l_{k,\mathbf{q}} l_{n-1-k,\mathbf{r}}$$

**$n \neq 0$  and abstraction with binding** consider a level  $i$ , that is a level of hole  $\square_i$ . In this hole we put a term  $S^{i-1}0$  of size  $i$ . There are  $m_i$  ways to choose a hole  $\square_i$ . Therefore there are  $m_i l_{n-i-1,\mathbf{m}}^\nu$  SwissCheeses which are abstractions with binding in which a  $\square_i$  has been replaced by the de Bruijn index  $S^{i-1}0$  among  $l_{n,0:\mathbf{m}^{\downarrow i}}^\nu$  SwissCheeses, where  $\mathbf{m}^{\downarrow i}$  is  $\mathbf{m}$  in which  $m_i$  is decremented. We notice that this refers only to an  $\mathbf{m}$  starting with 0. Hence by summing over  $i$  and adjusting  $\mathbf{m}$ , this part contributes as:

$$\sum_{i=0}^p (m_i + 1) l_{n-i,\mathbf{m}^{\uparrow i}}^\nu$$

to  $l_{n+1,0:\mathbf{m}}^\nu$ .

We have the following recursive definitions of  $l_{n,\mathbf{m}}^\nu$ :

$$\begin{aligned} l_{n+1,0:\mathbf{m}}^\nu &= \sum_{\mathbf{q} \oplus \mathbf{r} = 0:\mathbf{m}} \sum_{k=0}^n l_{k,\mathbf{q}}^\nu l_{n-k,\mathbf{r}}^\nu + \sum_{i=0}^p (m_i + 1) l_{n-i,\mathbf{m}^{\uparrow i}}^\nu \\ l_{n+1,(h+1):\mathbf{m}}^\nu &= \sum_{\mathbf{q} \oplus \mathbf{r} = (h+1):\mathbf{m}} \sum_{k=0}^n l_{k,\mathbf{q}}^\nu l_{n-k,\mathbf{r}}^\nu \end{aligned}$$

### 3.2 Variable size 0

The only difference is that the inserted de Bruijn index has size 0. Therefore we have  $m_i l_{n-1,\mathbf{m}}^0$  where we had  $m_i l_{n-i-1,\mathbf{m}}^\nu$  for natural size. Hence the formulas:

$$\begin{aligned} l_{n+1,0:\mathbf{m}}^0 &= \sum_{\mathbf{q} \oplus \mathbf{r} = 0:\mathbf{m}} \sum_{k=0}^n l_{k,\mathbf{q}}^0 l_{n-k,\mathbf{r}}^0 + \sum_{i=0}^p (m_i + 1) l_{n,\mathbf{m}^{\uparrow i}}^0 \\ l_{n+1,(h+1):\mathbf{m}}^0 &= \sum_{\mathbf{q} \oplus \mathbf{r} = (h+1):\mathbf{m}} \sum_{k=0}^n l_{k,\mathbf{q}}^0 l_{n-k,\mathbf{r}}^0 \end{aligned}$$

The sequence for closed linear terms is 0, 1, 0, 5, 0, 60, 0, 1105, 0, 27120, 0, 828250, which looks like sequence A062980 in the *On-line Encyclopedia of Integer Sequences*.

### 3.3 Variable size 1

The inserted de Bruijn index has size 1. We have  $m_i l_{n-2, \mathbf{m}}^1$  where we had  $m_i l_{n-i-1, \mathbf{m}}^\nu$  for natural size. Moreover we have to give the value of  $l_{1, \mathbf{m}}^1$  which is

$$l_{1, \mathbf{m}}^1 = [m_0 = 2 \wedge \bigwedge_{j=1}^p m_j = 0]$$

since there is only one SwissCheese of size 1, namely  $\square_0 \square_0$ , which corresponds to the tuple  $[2, 0, \dots]$ .

$$\begin{aligned} l_{1, \mathbf{m}}^1 &= [m_0 = 2 \wedge \bigwedge_{j=1}^p m_j = 0] \\ l_{n+1, 0: \mathbf{m}}^1 &= \sum_{\mathbf{q} \oplus \mathbf{r} = 0: \mathbf{m}} \sum_{k=0}^n l_{k, \mathbf{q}}^1 l_{n-k, \mathbf{r}}^1 + \sum_{i=0}^p (m_i + 1) l_{n-1, \mathbf{m}^{\uparrow i}}^1 \\ l_{n+1, (h+1): \mathbf{m}}^1 &= \sum_{\mathbf{q} \oplus \mathbf{r} = (h+1): \mathbf{m}} \sum_{k=0}^n l_{k, \mathbf{q}}^1 l_{n-k, \mathbf{r}}^1 \end{aligned}$$

There are no linear closed  $\lambda$ -terms of size  $3k$  and  $3k + 1$ . However for the values  $3k + 2$  we get the sequence: 1, 5, 60, 1105, 27120, ... which corresponds to the sequence A062980 in the *On-line Encyclopedia of Integer Sequences*, as noticed by Zeilberger [16].

## 4 Counting affine closed terms

We have just to add the case  $n \neq 0$  and *abstraction without binding*. Since no index is added, the size increases by 1. The numbers are written  $a_{n, \mathbf{m}}^\nu$ ,  $a_{n, \mathbf{m}}^0$ ,  $a_{n, \mathbf{m}}^1$ , and  $a_{n, \mathbf{m}}$  when the size does not matter. There are  $(0 : \mathbf{m})$ -SwissCheeses of size  $n$  that are abstraction without binding. We get the recursive formulas:

**Natural size**

$$\begin{aligned} a_{n+1, 0: \mathbf{m}}^\nu &= \sum_{\mathbf{q} \oplus \mathbf{r} = 0: \mathbf{m}} \sum_{k=0}^n a_{k, \mathbf{q}}^\nu a_{n-k, \mathbf{r}}^\nu + \sum_{i=0}^p (m_i + 1) a_{n-1, \mathbf{m}^{\uparrow i}}^\nu + a_{n, \mathbf{m}}^\nu \\ a_{n+1, (h+1): \mathbf{m}}^\nu &= \sum_{\mathbf{q} \oplus \mathbf{r} = (h+1): \mathbf{m}} \sum_{k=0}^n a_{k, \mathbf{q}}^\nu a_{n-k, \mathbf{r}}^\nu \end{aligned}$$

**Variable size 0**

$$\begin{aligned} a_{n+1, 0: \mathbf{m}}^0 &= \sum_{\mathbf{q} \oplus \mathbf{r} = 0: \mathbf{m}} \sum_{k=0}^n a_{k, \mathbf{q}}^0 a_{n-k, \mathbf{r}}^0 + \sum_{i=0}^p (m_i + 1) a_{n-1, \mathbf{m}^{\uparrow i}}^0 + a_{n, \mathbf{m}}^0 \\ a_{n+1, (h+1): \mathbf{m}}^0 &= \sum_{\mathbf{q} \oplus \mathbf{r} = (h+1): \mathbf{m}} \sum_{k=0}^n a_{k, \mathbf{q}}^0 a_{n-k, \mathbf{r}}^0 \end{aligned}$$

**Variable size 1**

$$\begin{aligned} a_{1, \mathbf{m}}^1 &= [m_0 = 2 \wedge \bigwedge_{j=1}^p m_j = 0] \\ a_{n+1, 0: \mathbf{m}}^1 &= \sum_{\mathbf{q} \oplus \mathbf{r} = 0: \mathbf{m}} \sum_{k=0}^n a_{k, \mathbf{q}}^1 a_{n-k, \mathbf{r}}^1 + \sum_{i=0}^p (m_i + 1) a_{n-1, \mathbf{m}^{\uparrow i}}^1 + a_{n, \mathbf{m}}^1 \\ a_{n+1, (h+1): \mathbf{m}}^1 &= \sum_{\mathbf{q} \oplus \mathbf{r} = (h+1): \mathbf{m}} \sum_{k=0}^n a_{k, \mathbf{q}}^1 a_{n-k, \mathbf{r}}^1 \end{aligned}$$

## 5 Generating functions

Consider families  $F_{\mathbf{m}}(z)$  of generating functions indexed by  $\mathbf{m}$ , where  $\mathbf{m}$  is an infinite tuple of naturals. In fact, we are interested in the infinite tuples  $\mathbf{m}$  that are always 0, except a finite number of indices, in order to compute  $F_{0^\omega}(z)$ , which corresponds to closed  $\lambda$ -terms. Let  $\mathbf{u}$  stands for the infinite sequences of variables  $(u_0, u_1, \dots)$  and  $\mathbf{u}^{\mathbf{m}}$  stands for  $(u_0^{m_0}, u_1^{m_1}, \dots, u_n^{m_n}, \dots)$  and  $\text{tail}(\mathbf{u})$  stand for  $(u_1, \dots)$ . We consider the series of two variables  $z$  and  $\mathbf{u}$  or double series associated with  $F_{\mathbf{m}}(z)$ :

$$\mathcal{F}(z, \mathbf{u}) = \sum_{\mathbf{m} \in \mathbb{N}^\omega} F_{\mathbf{m}}(z) \mathbf{u}^{\mathbf{m}}.$$

### Natural size

$L_{\mathbf{m}}^\nu(z)$  is associated with the numbers of *closed linear SwissCheeses* for natural size:

$$\begin{aligned} L_{0:\mathbf{m}}^\nu(z) &= z \sum_{\mathbf{m}' \oplus \mathbf{m}'' = 0:\mathbf{m}} L_{\mathbf{m}'}^\nu(z) L_{\mathbf{m}''}^\nu(z) + z \sum_{i=0}^{\infty} (m_i + 1) z^i L_{\mathbf{m}^\uparrow i}^\nu(z) \\ L_{(h+1):\mathbf{m}}^\nu(z) &= [h = 0 + \bigwedge_{i=0}^{\infty} m_i = 0] + z \sum_{\mathbf{m}' \oplus \mathbf{m}'' = (h+1):\mathbf{m}} L_{\mathbf{m}'}^\nu(z) L_{\mathbf{m}''}^\nu(z) \end{aligned}$$

$L_{0^\omega}^\nu$  is the generating function for the closed linear  $\lambda$ -terms.  $\mathcal{L}^\nu(z, \mathbf{u})$  is the double series associated with  $L_{\mathbf{m}}^\nu(z)$  and is solution of the equation:

$$\mathcal{L}^\nu(z, \mathbf{u}) = u_0 + z(\mathcal{L}^\nu(z, \mathbf{u}))^2 + z u_0 \sum_{i=1}^{\infty} z^i \frac{\partial \mathcal{L}^\nu(z, \text{tail}(\mathbf{u}))}{\partial u^i}$$

$\mathcal{L}^\nu(z, 0^\omega)$  is the generating function of closed linear  $\lambda$ -terms.

For *closed affine SwissCheeses* we get:

$$\begin{aligned} A_{0:\mathbf{m}}^\nu(z) &= z \sum_{\mathbf{m}' \oplus \mathbf{m}'' = 0:\mathbf{m}} A_{\mathbf{m}'}^\nu(z) A_{\mathbf{m}''}^\nu(z) + z \sum_{i=0}^{\infty} (m_i + 1) z^i A_{\mathbf{m}^\uparrow i}^\nu(z) + z A_{\mathbf{m}}^\nu(z) \\ A_{(h+1):\mathbf{m}}^\nu(z) &= [h = 0 + \bigwedge_{i=0}^{\infty} m_i = 0] + z \sum_{\mathbf{m}' \oplus \mathbf{m}'' = (h+1):\mathbf{m}} A_{\mathbf{m}'}^\nu(z) A_{\mathbf{m}''}^\nu(z) \end{aligned}$$

$A_{0^\omega}^\nu$  is the generating function for the affine linear  $\lambda$ -terms.  $\mathcal{A}^\nu(z, \mathbf{u})$  is the double series associated with  $A_{\mathbf{m}}^\nu(z)$  and is solution of the equation:

$$\mathcal{A}^\nu(z, \mathbf{u}) = u_0 + z(\mathcal{A}^\nu(z, \mathbf{u}))^2 + z u_0 \sum_{i=1}^{\infty} z^i \frac{\partial \mathcal{A}^\nu(z, \text{tail}(\mathbf{u}))}{\partial u^i} + z \mathcal{A}^\nu(z, \text{tail}(\mathbf{u}))$$

$\mathcal{A}^\nu(z, 0^\omega)$  is the generating function of closed linear  $\lambda$ -terms.

### Variable size 0

$L_{\mathbf{m}}^0$  is associated with the numbers of *closed linear SwissCheeses* for variable size 0:

$$\begin{aligned} L_{0:\mathbf{m}}^0(z) &= z \sum_{\mathbf{m}' \oplus \mathbf{m}'' = \mathbf{m}} L_{\mathbf{m}'}^0(z) L_{\mathbf{m}''}^0(z) + z \sum_{i=0}^{\infty} (m_i + 1) L_{\mathbf{m}^\uparrow i}^0(z) \\ L_{(h+1):\mathbf{m}}^0(z) &= [h = 0 + \bigwedge_{i=0}^{\infty} m_i = 0] + \sum_{\mathbf{m}' \oplus \mathbf{m}'' = \mathbf{m}} z L_{\mathbf{m}'}^0(z) L_{\mathbf{m}''}^0(z) \end{aligned}$$

$L_{0^\omega}^0$  is the generating function for the closed linear  $\lambda$ -terms.  $\mathcal{L}^0(z, \mathbf{u})$  is the double series associated with  $L_{\mathbf{m}}^0(z)$  and is solution of the equation:

$$\mathcal{L}^0(z, \mathbf{u}) = u_0 + z(\mathcal{L}^0(z, \mathbf{u}))^2 + z u_0 \sum_{i=1}^{\infty} \frac{\partial \mathcal{L}^0(z, \text{tail}(\mathbf{u}))}{\partial u^i}$$

$\mathcal{L}^0(z, 0^\omega)$  is the generating function of closed linear  $\lambda$ -terms.

For *closed affine SwissCheeses* we get:

$$\begin{aligned} A_{0:\mathbf{m}}^0(z) &= z \sum_{\mathbf{m}' \oplus \mathbf{m}'' = 0:\mathbf{m}} A_{\mathbf{m}'}^0(z) A_{\mathbf{m}''}^0(z) + z \sum_{i=0}^{\infty} (m_i + 1) A_{\mathbf{m} \uparrow i}^0(z) + z A_{\mathbf{m}}^0(z) \\ A_{(h+1):\mathbf{m}}^0(z) &= [h = 0 + \bigwedge_{i=0}^{\infty} m_i = 0] + \sum_{\mathbf{m}' \oplus \mathbf{m}'' = (h+1):\mathbf{m}} z A_{\mathbf{m}'}^0(z) A_{\mathbf{m}''}^0(z) \end{aligned}$$

$A_{0^\omega}^0$  is the generating function for the affine linear  $\lambda$ -terms.  $\mathcal{A}^0(z, \mathbf{u})$  is the double series associated with  $A_{\mathbf{m}}^0(z)$  and is solution of the equation:

$$\mathcal{A}^0(z, \mathbf{u}) = u_0 + z(\mathcal{A}^0(z, \mathbf{u}))^2 + z u_0 \sum_{i=1}^{\infty} \frac{\partial \mathcal{A}^0(z, \text{tail}(\mathbf{u}))}{\partial u^i} + z \mathcal{A}^0(z, \text{tail}(\mathbf{u}))$$

$\mathcal{A}^0(z, 0^\omega)$  is the generating function of closed linear  $\lambda$ -terms. We do not present variable size 1, since it goes exactly the same way.

## Variable size 1

The generating functions for  $l_{n,\mathbf{m}}^1$  are:

$$\begin{aligned} L_{0:\mathbf{m}}^1(z) &= z \sum_{\mathbf{m}' \oplus \mathbf{m}'' = \mathbf{m}} L_{\mathbf{m}'}^1(z) L_{\mathbf{m}''}^1(z) + z^2 \sum_{i=0}^{\infty} (m_i + 1) L_{\mathbf{m} \uparrow i}^1(z) \\ L_{(h+1):\mathbf{m}}^1(z) &= [h = 0 + \bigwedge_{i=0}^{\infty} m_i = 0] + z[h = 1 + \bigwedge_{i=0}^{\infty} m_i = 0] + \sum_{\mathbf{m}' \oplus \mathbf{m}'' = \mathbf{m}} z L_{\mathbf{m}'}^1(z) L_{\mathbf{m}''}^1(z) \end{aligned}$$

Then we get as associated double series :

$$\mathcal{L}^1(z, \mathbf{u}) = u_0 + z u_0^2 + z(\mathcal{L}^1(z, \mathbf{u}))^2 + z^2 u_0 \sum_{i=1}^{\infty} \frac{\partial \mathcal{L}^1(z, \text{tail}(\mathbf{u}))}{\partial u^i}$$



## 6 Effective computations

The definition of the coefficients  $a_m'$  and others is highly recursive and requires a mechanism of memoization. In Haskell, this can be done by using the call by need which is at the core of this language. Assume we want to compute the values of  $a_m'$  until a value `bound` for  $n$ . We use a recursive data structure:

```
data Mem = Mem [Mem] | Load [Integer]
```

in which we store the computed values of a function

```
am :: Int -> [Int] -> Integer
```

In our implementation the depth of the recursion of `Mem` is limited by `bound`, which is also the longest tuple `m` for which we will compute  $a_m'$ . Associated with `Mem` there is a function

```
access :: Mem -> Int -> [Int] -> Integer
access (Load l) n [] = l !! n
access (Mem listM) n (k:m) = access (listM !! k) n m
```

The leaves of the tree memory, corresponding to `Load`, contains the values of the function:

```
memory :: Int -> [Int] -> Mem
memory 0 m = Load [am n (reverse m) |
n<-[0..]]
memory k m = Mem [memory (k-1) (j:m) | j<-[0..]]
```

The memory relative to the problem we are interested in is

```
theMemory = memory (bound) []
```

and the access to `theMemory` is given by a specific function:

```
acc :: Int -> [Int] -> Integer
acc n m = access theMemory n m
```

Notice that `am` and `acc` have the same signature. This is not a coincidence, since `acc` accesses values of `am` already computed. Now we are ready to express `am`.

```
am 0 m = iv (head m == 1 && all ((==) 0) (tail m))
am n m = amAPP n m + amABSwB n m
```

`amAPP` counts affine terms that are applications:

```
amAPP n m = sum (map (\((q,r),(k,nk)) -> (acc k q) * (acc nk
r)) (allCombinations m (n-1)))
```

where `allCombinations` returns a list of all the pairs of pairs  $(\mathbf{m}', \mathbf{m}'')$  such  $\mathbf{m} = \mathbf{m}' \oplus \mathbf{m}''$  and of pairs  $(k, nk)$  such that  $k + nk = n$ . `amABSwB` counts affine terms that are abstractions with binding.

```
amABSwB n m | head m == 0 = sum [amABSAtd n m i | i<-[1..(n-1)]] |
otherwise = 0
```

`amABSAtd` counts affine terms that are abstractions with binding at level  $i$ :

```
amABSAtd n m i = (fromIntegral (1 + m!!i)) * (acc (n - i - 1) (tail (inc i
m) ++ [0]))
```

`amABSnB` counts affine terms that are abstractions with no binding:

```
amABSnB n m | head m == 0 = (acc (n-1) (tail m ++ [0])) | otherwise = 0
```

Anyway the efficiency of this program is limited by the size of the memory, since for computing  $a_{n,0^n}'$ , for instance, we need to compute  $a_r'$  for about  $n!$  values.

## 7 Generating affine and linear terms

By relatively small changes it is possible to build programs which generate linear and affine terms. For instance for generating affine terms we get.

```
amg :: Int -> [Int] -> [SwissCheese]
amg 0 m = if (head m == 1 && all ((==) 0) (tail m)) then [Box 0] else []
amg n m = allAPP n m ++ allABSwB n m ++ allABSnB n m

allAPP :: Int -> [Int] -> [SwissCheese]
allAPP n m = foldr (++) [] (map (\((q,r),(k,nk)) -> appSC (cartesian (accAG k q)
                                                                    (accAG nk r))
                                (allCombinations m (n-1))))

allABSAtd :: Int -> [Int] -> Int -> [SwissCheese]
allABSAtd n m i = foldr (++) [] (map (abstract (i-1)) (accAG (n - i - 1)
                                                             (tail (inc i m) ++ [0])))

allABSwB :: Int -> [Int] -> [SwissCheese]
allABSwB n m
  | head m == 0 = foldr (++) [] [allABSAtd n m i | i<-[1..(n-1)]]
  | otherwise = []

allABSnB :: Int -> [Int] -> [SwissCheese]
allABSnB n m
  | head m == 0 = map (AbsSC . raise) (accAG (n-1) (tail m ++ [0]))
  | otherwise = []

memoryAG :: Int -> [Int] -> MemSC
memoryAG 0 m = LoadSC [amg n (reverse m) | n<-[0..]]
memoryAG k m = MemSC [memoryAG (k-1) (j:m) | j<-[0..]]

theMemoryAG = memoryAG (upBound) []

accAG :: Int -> [Int] -> [SwissCheese]
accAG n m = accessSC theMemoryAG n m
```

There is similar programs for generating all the terms of size  $n$  for variable size 0 and variable size 1. From this, we get programs for generating random affine terms or random linear terms.

## 8 Normal forms

From the method used for counting affine and linear closed terms, it is easy to deduce method for counting affine and linear closed normal forms. Like before, we use SwissCheeses. In this section we consider only natural size.

### 8.1 Natural size

#### Affine closed normal forms

Let us call  $anf_{n,m}$  the number of affine SwissCheeses with no  $\beta$ -redex and  $ane_{n,m}$  the number of neutral affine SwissCheeses, i.e., affine SwissCheeses with no  $\beta$ -redexes that are sequences of applications starting with a de Bruijn index. In addition we count:

- $anf\lambda w_{n,m}$  the number of affine SwissCheeses with no  $\beta$ -redex which are abstraction with a binding of a de Bruijn index,
- $anf\lambda n_{n,m}$  the number of affine SwissCheeses with no  $\beta$ -redex which are abstraction with no binding.

$$\begin{aligned} anf_{0,\mathbf{m}} &= ane_{0,\mathbf{m}} \\ anf_{n+1,\mathbf{m}} &= ane_{n+1,\mathbf{m}} + anf\lambda w_{n+1,m} + anf\lambda n_{n+1,m} \end{aligned}$$

where

$$\begin{aligned} ane_{0,\mathbf{m}} &= m_0 = 1 \wedge \bigwedge_{j=1}^p m_j = 0 \\ ane_{n+1,\mathbf{m}} &= \sum_{\mathbf{q} \oplus \mathbf{r} = 0:\mathbf{m}} \sum_{k=0}^n ane_{k,\mathbf{q}} anf_{n-k,\mathbf{r}} \end{aligned}$$

and

$$anf\lambda w_{n,m} = \sum_{i=0}^p (m_i + 1) anf_{n-i,\mathbf{m} \uparrow i}$$

and

$$anf\lambda n_{n+1,m} = anf_{n,m}$$

There are two generating functions,  $\mathcal{A}^{nf}$  and  $\mathcal{A}^{ne}$ , which are associated to  $anf_{n,\mathbf{m}}$  and  $anf_{n,\mathbf{m}}$ :

$$\begin{aligned} \mathcal{A}^{nf}(z, \mathbf{u}) &= \mathcal{A}^{ne}(z, \mathbf{u}) + z u_0 \sum_{i=1}^{\infty} z^i \frac{\partial \mathcal{A}^{nf}(z, (\mathbf{tail}(\mathbf{u}))}{\partial u^i} + z \mathcal{A}^{nf}(z, (\mathbf{tail}(\mathbf{u})) \\ \mathcal{A}^{ne}(z, \mathbf{u}) &= u_0 + z \mathcal{A}^{ne}(z, \mathbf{u}) \mathcal{A}^{nf}(z, \mathbf{u}) \end{aligned}$$

### Linear closed normal forms

Let us call  $lnf_{n,\mathbf{m}}$  the number of linear SwissCheeses with no  $\beta$ -redex and  $lne_{n,\mathbf{m}}$  the number of neutral linear SwissCheeses, linear SwissCheeses with no  $\beta$ -redexes that are sequences of applications starting with a de Bruijn index. In addition we count  $lnf\lambda w_{n,m}$  the number of linear SwissCheeses with no  $\beta$ -redex which are abstraction with a binding of a de Bruijn index.

$$\begin{aligned} lnf_{0,\mathbf{m}} &= lne_{0,\mathbf{m}} \\ lnf_{n+1,\mathbf{m}} &= lne_{n+1,\mathbf{m}} + lnf\lambda w_{n+1,m} \end{aligned}$$

where

$$\begin{aligned} lne_{0,\mathbf{m}} &= m_0 = 1 \wedge \bigwedge_{j=1}^p m_j = 0 \\ lne_{n+1,\mathbf{m}} &= \sum_{\mathbf{q} \oplus \mathbf{r} = 0:\mathbf{m}} \sum_{k=0}^n lne_{k,\mathbf{q}} lnf_{n-k,\mathbf{r}} \end{aligned}$$

and

$$lnf\lambda w_{n,m} = \sum_{i=0}^p (m_i + 1) lnf_{n-i,\mathbf{m} \uparrow i}$$

with the two generating functions:

$$\begin{aligned}\mathcal{L}^{nf}(z, \mathbf{u}) &= \mathcal{L}^{ne}(z, \mathbf{u}) + z u_0 \sum_{i=1}^{\infty} z^i \frac{\partial \mathcal{L}^{nf}(z, (\text{tail}(\mathbf{u}))}{\partial u^i} \\ \mathcal{L}^{ne}(z, \mathbf{u}) &= u_0 + z \mathcal{L}^{ne}(z, \mathbf{u}) \mathcal{L}^{nf}(z, \mathbf{u})\end{aligned}$$

We also deduce programs for generating all the closed affine or linear normal forms of a given size.

## 8.2 Variable size 0 or 1

To be done.

## 9 Related works

There are several works on counting  $\lambda$ -terms, for instance on natural size [3, 2], on variable size 1 [7, 8, 15], on variable size 0 [11], on affine terms with variable size 1 [5], on linear  $\lambda$ -terms [17], also on a size based binary representation of the  $\lambda$ -calculus [12] (see [10] for a synthetic view of both natural size and binary size).

## 10 Conclusion

This work on counting opens new perspective on the generation, for instance the random generation of closed lambda terms in the line of [12, 4].

## References

- [1] Henk P. Barendregt. *The Lambda-Calculus, its syntax and semantics*. Studies in Logic and the Foundation of Mathematics. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1984. Second edition.
- [2] Maciej Bendkowski, Katarzyna Grygiel, Pierre Lescanne, and Marek Zaionc. Combinatorics of  $\lambda$ -terms: a natural approach. *CoRR*, abs/1609.07593, 2016.
- [3] Maciej Bendkowski, Katarzyna Grygiel, Pierre Lescanne, and Marek Zaionc. A natural counting of lambda terms. In Rusins Martins Freivalds, Gregor Engels, and Barbara Catania, editors, *SOFSEM 2016: Theory and Practice of Computer Science - 42nd International Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 23-28, 2016, Proceedings*, volume 9587 of *Lecture Notes in Computer Science*, pages 183–194. Springer, 2016.
- [4] Maciej Bendkowski, Katarzyna Grygiel, and Paul Tarau. Boltzmann samplers for closed simply-typed lambda terms. In Yuliya Lierler and Walid Taha, editors, *Practical Aspects of Declarative Languages - 19th International Symposium, PADL 2017, Paris, France, January 16-17, 2017, Proceedings*, volume 10137 of *Lecture Notes in Computer Science*, pages 120–135. Springer, 2017.
- [5] Olivier Bodini, Danièle Gardy, Bernhard Gittenberger, and Alice Jacquot. Enumeration of generalized BCI lambda-terms. *ArXiv e-prints*, May 2013.
- [6] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
- [7] René David, Christophe Raffalli, Guillaume Theyssier, Katarzyna Grygiel, Jakub Kozik, and Marek Zaionc. Asymptotically almost all  $\lambda$ -terms are strongly normalizing. *CoRR*, abs/0903.5505v3, 2009.

- [8] René David, Christophe Raffalli, Guillaume Theyssier, Katarzyna Grygiel, Jakub Kozik, and Marek Zaionc. Some properties of random lambda terms. *Logical Methods in Computer Science*, 9(1), 2009.
- [9] N. G. 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.
- [10] Bernhard Gittenberger and Zbigniew Golebiewski. On the number of lambda terms with prescribed size of their de bruijn representation. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPICs*, pages 40:1–40:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
- [11] Katarzyna Grygiel and Pierre Lescanne. Counting and generating lambda terms. *J. Funct. Program.*, 23(5):594–628, 2013.
- [12] Katarzyna Grygiel and Pierre Lescanne. Counting and generating terms in the binary lambda calculus. *J. Funct. Program.*, 25, 2015.
- [13] J. Roger Hindley. BCK-combinators and linear lambda-terms have types. *Theor. Comput. Sci.*, 64(1):97–105, 1989.
- [14] J. Roger Hindley. *Basic Simple Type Theory*. Number 42 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1997.
- [15] Pierre Lescanne. On counting untyped lambda terms. *Theor. Comput. Sci.*, 474:80–97, 2013.
- [16] Noam Zeilberger. Counting isomorphism classes of  $\beta$ -normal linear lambda terms. *CoRR*, abs/1509.07596, 2015.
- [17] Noam Zeilberger. Linear lambda terms as invariants of rooted trivalent maps. *J. Funct. Program.*, 26:e21, 2016.

## Data

In the appendix, we give the first values of  $l_{n,0^n}^\nu$ ,  $a_{n,0^n}^\nu$ ,  $anf_{n,0^n}$ , and  $lnf_{n,0^n}$ .

0	0	51	51496022711337536
1	0	52	124591137939086496
2	1	53	299402908258405410
3	0	54	721839933329222924
4	0	55	1747307145272084192
5	3	56	4211741383777966592
6	2	57	10165998012602469888
7	0	58	24620618729658655936
8	16	59	59482734150603634286
9	24	60	143764591607556354344
10	8	61	348379929166234350008
11	117	62	843169238563254723200
12	252	63	2040572920613086128400
13	180	64	4948102905207104837424
14	1024	65	11992521016286173712196
15	2680	66	29059897435554891991144
16	2952	67	70516464312280927105392
17	10350	68	171105110698292441423968
18	29420	69	415095704639682396539232
19	42776	70	1008016383720573882885792
20	116768	71	2448305474519849567597826
21	335520	72	5945721872300885649415632
22	587424	73	14449388516068567845838736
23	1420053	74	35125352062243788817753856
24	3976424	75	85382289240293493116120064
25	7880376	76	207650379931166057815603296
26	18103936	77	505172267243918348155299780
27	48816576	78	1229005880128485245247395000
28	104890704	79	2991079243470267667831893408
29	237500826	80	7281852742753184123608419712
30	617733708	81	17729171587798767750815341440
31	1396750576	82	43177454620325445122944305984
32	3171222464	83	105185452787117035266315446868
33	8014199360	84	256273862465425158211948020048
34	18688490336	85	624527413292252904584121980208
35	42840683418	86	1522355057007327280427270436480
36	106063081288	87	3711429775030704772089070886624
37	251769197688	88	9050041253711022076275958636128
38	583690110208	89	22073150301758857110072042919800
39	1425834260080	90	53844910909398928990641101351664
40	3417671496432	91	131371135544173914537076774932576
41	8007221710652	92	320588677238085642820920910555968
42	19404994897976	93	782465218885869813183863213231424
43	46747189542384	94	1910077425906069707804966102543936
44	110498345360800	95	4663586586924802791117231052636349
45	266679286291872	96	11388259565942452837717688743953504
46	644021392071840	97	27813754361897984543467478917223008
47	1533054190557133	98	67941781284113201998645699501746176
48	3693823999533360	99	165989485724048964272023600773271424
49	8931109667692464	100	405588809305168453963137377442321728
50	21375091547312128		

Figure 3: *Natural size*: numbers of closed linear terms of size  $n$  from 0 to 100

0	0	51	803928779462727941247
1	0	52	2314623127904669382002
2	1	53	6667810436356967142481
3	1	54	19218411059885449257096
4	2	55	55421020161661024650870
5	5	56	159899218321197381984561
6	12	57	461557020400062903560120
7	25	58	1332920908954281811200519
8	64	59	3851027068336583693412910
9	166	60	11131032444503136571789527
10	405	61	32186581221116996967632029
11	1050	62	93108410048006285466998584
12	2763	63	269446191702411420790402033
13	7239	64	780043726186403167392453886
14	19190	65	2259043189995515315930349650
15	51457	66	6544612955390252336187266873
16	138538	67	18966737218108971681014445025
17	374972	68	54985236298270057405776629352
18	1020943	69	159455737350384637847783055311
19	2792183	70	462562848624435724964181323484
20	7666358	71	1342251884451664733064283251627
21	21126905	72	3896065622127200625653134100538
22	58422650	73	11312117748805772104795220337816
23	162052566	74	32853646116456632492645965741531
24	450742451	75	95442534633482460553801961967438
25	1256974690	76	277342191547330839640289978813667
26	3513731861	77	806125189457291902863848267463755
27	9843728012	78	2343682130911232279285707290604156
28	27633400879	79	6815564023736534208079367816340359
29	77721141911	80	19824812322145727566417303371819466
30	218984204904	81	57679033022808238913186144092831856
31	618021576627	82	167851787082561392384648248846390041
32	1746906189740	83	488574368670832093243802790464796207
33	4945026080426	84	1422426342380883254459783410845365006
34	14017220713131	85	4142104564089044203901190817275864665
35	39784695610433	86	12064305885705003967881526911560653106
36	113057573020242	87	35145647815239737143373764367447378676
37	321649935953313	88	102406303052123097062053564818109468705
38	916096006168770	89	298446029598661205216170897850336550644
39	2611847503880831	90	869935452705023302189031644932803990417
40	7453859187221508	91	2536229492704354513309696228592784181158
41	21292177500898858	92	7395518143425160073537967606298755947391
42	60875851617670699	93	21568776408467701927134211542478146593789
43	174195916730975850	94	62915493935623036562559989770249004382816
44	498863759031591507	95	183553775888862113259168150130266362416356
45	1429753835635525063	96	535600661621556969155453544692826625532079
46	4100730353324163138	97	1563109720672526919899689366626240867515144
47	11769771167532816128	98	4562542818801138452310024131223304186909233
48	33804054749367200891	99	13319630286623965617386598746472280781972745
49	97151933333668422006	100	38890520391341859449843201188612375394153776
50	279385977720772581435		

Figure 4: *Natural size*: numbers of closed affine terms of size  $n$  from 0 to 100

0	0	41	3037843646560
1	0	42	6895841598615
2	1	43	15666498585568
3	1	44	35620848278448
4	2	45	81052838239593
5	3	46	184564847153821
6	7	47	420564871255118
7	10	48	958975854646984
8	20	49	2188068392529104
9	40	50	4995528560788451
10	77	51	11411921511827547
11	160	52	26084524952754538
12	318	53	59654682828889245
13	671	54	136500653558490261
14	1405	55	312496493161999851
15	2981	56	715760763686417314
16	6312	57	1640194881084692664
17	13672	58	3760284787917366081
18	29399	59	8624561382605096780
19	63697	60	19789639944299656346
20	139104	61	45427337308377290201
21	304153	62	104320438668034814453
22	667219	63	239656248361374562433
23	1469241	64	550769764273325683828
24	3247176	65	1266217774600330829940
25	7184288	66	2912050679107531357883
26	15949179	67	6699418399886008666265
27	35480426	68	15417663698156810292010
28	79083472	69	35492710197462925262295
29	176607519	70	81732521943462960197057
30	395119875	71	188270363628099910161436
31	885450388	72	433807135012774797924026
32	1987289740	73	999851681931974600766994
33	4466760570	74	2305129188866501774481545
34	10053371987	75	5315847675735178072941600
35	22656801617	76	12262083079763320881047944
36	51121124910	77	28292248892584567512609357
37	115478296639	78	65294907440089718078048829
38	261139629999	79	150729070403767032817820543
39	591138386440	80	348031015577337732605480908
40	1339447594768		

Figure 5: *Natural size*: numbers of closed affine normal forms of size  $n$  from 0 to 8