Quantitative aspects of linear and affine closed lambda terms
Pierre Lescanne

To cite this version:

HAL Id: ensl-01464047
https://hal-ens-lyon.archives-ouvertes.fr/ensl-01464047v5
Submitted on 21 May 2017

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, élaborés 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
May 21, 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 variable 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 size $n$, 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 [8] 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.\(^1\)

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, i.e., only free variables. An affine $\lambda$-term (or BCK term) is a $\lambda$-term in which bound variables occur at most once. A linear $\lambda$-term (or BCI 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 [11], 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 [17, 16]. For instance, generated by the program of Section 5.6, there are 16 linear terms of natural size 8:

\[
\begin{align*}
(\lambda 0 (\lambda 0 0)) & \quad (\lambda 0 (\lambda 0 0)) & \quad (\lambda 0 (\lambda 0 0)) & \quad (\lambda 0 (\lambda 0 0)) & \quad (\lambda 0 (\lambda 0 0)) & \quad (\lambda 0 (\lambda 0 0)) & \quad (\lambda 0 (\lambda 0 0)) & \quad (\lambda 0 (\lambda 0 0)) \\
(\lambda (\lambda 0 0) 0) & \quad (\lambda (\lambda 0 0) 0) & \quad (\lambda (\lambda 0 0) 0) & \quad (\lambda (\lambda 0 0) 0) & \quad (\lambda (\lambda 0 0) 0) & \quad (\lambda (\lambda 0 0) 0) & \quad (\lambda (\lambda 0 0) 0) & \quad (\lambda (\lambda 0 0) 0)
\end{align*}
\]

written with explicit variables

\[
\begin{align*}
\lambda x.x & \quad (\lambda x.x \lambda x.x) & \quad \lambda x.x \lambda y.(\lambda x.x y) & \quad \lambda x.x \lambda y.(\lambda x.x) & \quad (\lambda x.x \lambda x.x) \lambda x.x
\end{align*}
\]

\(^1\)If the reader is not familiar with the $\lambda$-calculus, we advise him to read the introduction of [15], for instance.
and there are 25 affine terms of natural size 7:

\[
\begin{align*}
(\lambda 0 \lambda 1) & \quad (\lambda 0 \lambda \lambda 0) & \quad (\lambda \lambda 0 \lambda 0) & \quad (\lambda \lambda 0 \lambda 0) & \quad (\lambda \lambda 0 0) & \quad (\lambda 0 \lambda 1) \\
\lambda(0 \lambda \lambda 0) & \quad \lambda(\lambda 0 \lambda 1) & \quad \lambda(\lambda 1 \lambda 0) & \quad \lambda(\lambda 1 0 1) & \quad \lambda(1 \lambda 0) & \quad \lambda(\lambda 0 \lambda 0) & \quad \lambda(\lambda \lambda 0) & \quad \lambda(\lambda \lambda 0 0) \\
\lambda(0 \lambda \lambda 0) & \quad \lambda(\lambda(0 1) & \quad \lambda(1 0) & \quad \lambda(0 \lambda 0) & \quad \lambda(\lambda(0 0) & \quad \lambda(\lambda(0 0) & \quad \lambda(\lambda(0 0) & \quad \lambda(\lambda(0 0) & \quad \lambda(\lambda(0 0) \\
\end{align*}
\]

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

**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 \( m \in \mathbb{N}^p \) be the \( p \)-tuple \((m_0, ..., m_{p-1})\). In Section 5, we consider also infinite tuples. Thus \( m \in \mathbb{N}^\omega \) is the sequence \((m_0, m_1, ...)\). Notice in the case of infinite tuples, we are only interested in infinite tuples equal to 0 after some index.

- \( p \) is the length of \( m \), which we write also length \( m \)
- The \( p \)-tuple \((0, ..., 0)\) is written \( 0^p \). \( 0^\omega \) is the infinite tuple made of 0’s.
- The increment of a \( p \)-tuple at \( i \) is:
  \[
m^i = n \in \mathbb{N}^p \text{ where } n_j = m_j \text{ if } j \neq i \text{ and } n_i = m_i + 1
  \]
- Putting an element \( x \) as head of a tuple is written
  \[
x : m = x : (m_0, ..., ) = (x, m_0, ...)
  \]
  tail removes the head of a tuple:
  \[
tail(x : m) = m.
  \]
- \( + \) is the componentwise addition on tuples.

**2 SwissCheese**

The basic concept is this of \( m \)-SwissCheese or Swisscheese of characteristic \( m \) or simply SwissCheese if there is no ambiguity on \( m \). A \( m \)-SwissCheese or a SwissCheese of characteristic \( m \), where \( m \) is of length \( p \) is, a \( \lambda \)-term with holes at \( p \) levels, which are all counted, using \( m \). The \( p \) levels of holes are \( \square_0, ..., \square_{p-1} \). 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 \( m \)-SwissCheese or a SwissCheese of characteristic \( m \) has \( m_0 \) holes at level 0, \( m_1 \) holes at level 1, ..., \( m_{p-1} \) holes at level \( p-1 \). Let \( l_{n,m} \) (resp. \( a_{n,m} \)) count the linear (resp. the affine) \( m \)-SwissCheese of size \( n \). \( l_{n,m} = l_{n,m'} \) and \( a_{n,m} = a_{n,m'} \) if \( m \) is finite, \( \text{length } m \geq n \), \( m_i = m'_i \) for \( i \leq \text{length } m \), and \( m'_i = 0 \) for \( i > \text{length } m \). \( l_{n,0^\omega} \) (resp. \( a_{n,0^\omega} \)) 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 \( c_1 \circ c_2 \) is a \((1, 2, 0, 0, 0)\)-SwissCheese. Said otherwise, \( c_1 \) has characteristic \((0, 1, 0, 0, 0)\), \( c_2 \) has characteristic \((1, 1, 0, 0, 0)\) and \( c_1 \circ c_2 \) has characteristic \((1, 2, 0, 0, 0)\). According to what we said, \( c_1 \circ c_2 \) has characteristic \((1, 2)\) as well as characteristic \((1, 2, 0, 0, 0, \ldots)\) (a tuple starting with 1, followed by 2, followed by infinitely many 0’s). We could also say that \( c_1 \) has characteristic \((0, 1)\) and \( c_2 \) has characteristic \((1, 1)\) making \( \circ \) a binary operation on SwissCheeses of length 2 whereas previously we have made \( \circ \) a binary operation on SwissCheeses of length 5. In other words, when counting SwissCheeses of characteristic \( m \), the trailing 0’s are irrelevant. In actual computations, we make the lengths of characteristics consistent by adding trailing 0’s to too short ones.

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

1. We put a \( \lambda \) on the top of a \( m \)-SwissCheese \( c \). This increases the levels of the holes: a hole \( \Box_i \) becomes a hole \( \Box_{i+1} \). \( \lambda c \) is a \((0 : m)\)-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 \( \Box_i \), we top the SwissCheese by a \( \lambda \), we increment the levels of the other holes and we replace the chosen hole by \( S^0 \). In Figure 2 (right), we colour the added \( \lambda \) in green and we call it abstraction with binding.

2.2 Measuring SwissCheese

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

\[
c_1 = \lambda \Box_1 \quad \Box_1 \quad 0 \quad c_2 = \Box_0 \quad 0 \quad c_1 \circ c_2 = \Box_0 \quad 0
c_1 \circ c_2 = \Box_0 \quad 0
\]
Variables have size 0, we call this \textbf{variable size} 0.

Variables have size 1, we call this \textbf{variable size} 1.

Variables (or de Bruijn indices) \( S^i \) have size \( i + 1 \), we call this \textbf{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 \( l_{\nu,n,m} \) of linear SwissCheeses of natural size \( n \) with holes set by \( m \), then for the numbers \( l_{0,n,m} \) of linear SwissCheeses of size \( n \), for variable size 0, with holes set by \( m \), eventually for the numbers \( l_{1,n,m} \) of linear SwissCheeses of size \( n \), for variable size 1, with holes set by \( m \). When we do not want to specify a chosen size, we write only \( l_{\nu,n,m} \) without superscript.

#### 3.1 Natural size

First let us count linear SwissCheeses with natural size. This is given by the coefficient \( l_{\nu,n,m} \) which has two arguments: the size \( n \) of the SwissCheese and a tuple \( m \) which specifies the number of holes of each level. In other words we are interested in the quantity \( l_{\nu,n,m} \).

\( n = 0 \) whatever size is considered, there is only one SwissCheese of size 0 namely \( □_0 \). This means that the number of SwissCheeses of size 0 is 1 if and only if \( m = (1,0,0,...) \):

\[
l_{\nu,0,m} = l_{0,0,m} = l_{1,0,m} = [m_0 = 1 \land \bigwedge_{j=1}^{p-1} m_j = 0]
\]

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

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

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

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

to \( l_{\nu,n+1,0,m}^m \).

We have the following recursive definitions of \( l_{\nu,0,m}^m \):

\[
l_{\nu,n+1,0,m} = \sum_{q \oplus r = 0,m} \sum_{k=0}^{n} l_{k,q} l_{n-k,r} + \sum_{i=0}^{p-1} (m_i + 1) l_{\nu,n-i,m_i}^m
\]

\[
l_{\nu,n+1,(h+1):m} = \sum_{q \oplus r = (h+1):m} \sum_{k=0}^{n} l_{k,q} l_{n-k,r}
\]

Numbers of closed linear terms with natural size are given in Figure 3.
3.2 Variable size 0

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

$$l^0_{n+1,0,m} = \sum_{q@r=0} \sum_{k=0}^{n} l^0_{k,q} l^0_{n-k,r} + \sum_{i=0}^{p-1} (m_i + 1) l^0_{n,m,i}$$

$$l^0_{n+1,(h+1):m} = \sum_{q@r=(h+1):m} \sum_{k=0}^{n} l^0_{k,q} l^0_{n-k,r}$$

The sequence $l^0_{n,0,m}$ of the numbers of closed linear terms is 0, 1, 0, 5, 0, 60, 0, 1105, 0, 27120, 0, 828250, which is sequence A062980 in the On-line Encyclopedia of Integer Sequences with 0’s at even indices.

3.3 Variable size 1

The inserted de Bruijn index has size 1. We have $m_i l^1_{n-2,m}$ where we had $m_i l^\nu_{n-i-1,m}$ for natural size.

$$l^1_{n+1,0,m} = \sum_{q@r=0} \sum_{k=0}^{n} l^1_{k,q} l^1_{n-k,r} + \sum_{i=0}^{n-1} (m_i + 1) l^1_{n-1,m,i}$$

$$l^1_{n+1,(h+1):m} = \sum_{q@r=(h+1):m} \sum_{k=0}^{n} l^1_{k,q} l^1_{n-k,r}$$

As noticed by Grygiel et al. [13] (§ 6.1) 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 is again sequence A062980 of the On-line Encyclopedia of Integer Sequences.

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 $a^\nu_{n,m}$, $a^0_{n,m}$, $a^1_{n,m}$, and $a_{n,m}$ when the size does not matter. There are (0 : m)-SwissCheeses of size $n$ that are abstraction without binding. We get the recursive formulas:

4.1 Natural size

$$a^\nu_{n+1,0,m} = \sum_{q@r=0} \sum_{k=0}^{n} a^\nu_{k,q} a^\nu_{n-k,r} + \sum_{i=0}^{p-1} (m_i + 1) a^\nu_{n-i,m,i} + a^\nu_{n,m}$$

$$a^\nu_{n+1,(h+1):m} = \sum_{q@r=(h+1):m} \sum_{k=0}^{n} a^\nu_{k,q} a^\nu_{n-k,r}$$

The numbers of closed affine size with natural size are given in Figure 4.

4.2 Variable size 0

$$a^0_{n+1,0,m} = \sum_{q@r=0} \sum_{k=0}^{n} a^0_{k,q} a^0_{n-k,r} + \sum_{i=0}^{p-1} (m_i + 1) a^0_{n,m,i} + a^0_{n,m}$$

$$a^0_{n+1,(h+1):m} = \sum_{q@r=(h+1):m} \sum_{k=0}^{n} a^0_{k,q} a^0_{n-k,r}$$
The sequence $a_{0,0}^{0,0}$ of the numbers of affine closed terms for variable size 0 is

$$0, 1, 2, 8, 29, 140, 661, 3622, 19993, 120909, 744890, 4887401, 32795272, ...$$

It does not appear in the *On-line Encyclopedia of Integer Sequences*. It corresponds to the coefficients of the generating function $A(z, u)$ where

$$A(z, u) = u + z(A(z, u))^2 + z \frac{\partial A(z, u)}{\partial u} + zA(z, u).$$

### 4.3 Variable size 1

$$a_{n+1,0}^{1,0} = \sum_{q,r=0}^{n-1} a_k^0 a_{n-k, r}^1 + \sum_{i=0}^{n-1} (m_i + 1) a_{n-1,m}^i + a_{n,m}^1$$

The sequence $a_{1,0}^{1,0}$ of the numbers of affine closed terms for variable size 1 is

$$0, 0, 1, 2, 3, 9, 30, 81, 242, 838, 2799, 9365, 33616, 122937, 449698, 1696724, 6558855, ...$$

This is sequence A281270 in the *On-line Encyclopedia of Integer Sequences*. However it corresponds to the coefficient of the generating function $A(z, u)$ where $A(z, u)$ is the solution of the functional equation:

$$A(z, u) = zu + z(A(z, u))^2 + z \frac{\partial A(z, u)}{\partial u} + zA(z, u).$$

Notice that this corrects the wrong assumptions of [13] (Section 6.2).

### 5 Generating functions

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

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

**Natural size**

$L_m^\nu(z)$ is associated with the numbers of *closed linear SwissCheeses* for natural size:

$$L_{0,m}^\nu(z) = z \sum_{m' \oplus m'' = 0} L_{m'}^\nu(z)L_{m''}^\nu(z) + z \sum_{i=0}^{\infty} (m_i + 1) z^i L_{m_1}^\nu(z)$$

$$L_{(h+1),m}^\nu(z) = [h = 0 + \bigwedge_{i=0}^{\infty} m_i = 0] + z \sum_{m' \oplus m'' = (h+1)} L_{m'}^\nu(z)L_{m''}^\nu(z)$$
$L^\nu_m$ is the generating function for the closed linear $\lambda$-terms. $L^\nu(z,u)$ is the double series associated with $L^\nu_m(z)$ and is solution of the equation:

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

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

For closed affine SwissCheeses we get:

$$A^\nu_0(z) = z \sum_{m'@m''=0} A^\nu_m(z)A^\nu_{m''}(z) + z \sum_{i=0}^{\infty} (m_i + 1)z^i A^\nu_{m+i}(z) + z A^\nu_m(z)$$

$$A^\nu_{m+1}(z) = [h = 0 + \sum_{i=0}^{\infty} m_i = 0] + \sum_{m'@m''=m+1} A^\nu_m(z)A^\nu_{m''}(z)$$

$A^\nu_m$ is the generating function for the closed linear $\lambda$-terms. $A^\nu(z,u)$ is the double series associated with $A^\nu_m(z)$ and is solution of the equation:

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

$A^\nu(z,0^\nu)$ is the generating function of closed affine $\lambda$-terms.

**Variable size 0**

$L^0_m$ is associated with the numbers of closed linear SwissCheeses for variable size 0:

$$L^0_0(z) = z \sum_{m'@m''=0} L^0_m(z)A^0_{m''}(z) + z \sum_{i=0}^{\infty} (m_i + 1) L^0_{m+i}(z)$$

$$L^0_{m+1}(z) = [h = 0 + \sum_{i=0}^{\infty} m_i = 0] + \sum_{m'@m''=m+1} zL^0_m(z)A^0_{m''}(z)$$

$L^0_m$ is the generating function for the closed linear $\lambda$-terms. $L^0(z,u)$ is the double series associated with $L^0_m(z)$ and is solution of the equation:

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

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

For closed affine SwissCheeses we get:

$$A^0_0(z) = z \sum_{m'@m''=0} A^0_m(z)A^0_{m''}(z) + z \sum_{i=0}^{\infty} (m_i + 1) A^0_{m+i}(z) + z A^0_m(z)$$

$$A^0_{m+1}(z) = [h = 0 + \sum_{i=0}^{\infty} m_i = 0] + \sum_{m'@m''=m+1} zA^0_m(z)A^0_{m''}(z)$$

$A^0_m$ is the generating function for the affine linear $\lambda$-terms. $A^0(z,u)$ is the double series associated with $A^0_m(z)$ and is solution of the equation:

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

$A^0(z,0^\nu)$ is the generating function of closed linear $\lambda$-terms.
Variable size 1

The generating functions for $l^1_{n,m}$ are:

$$L^1_{0,m}(z) = z \sum_{m' @ m'' = m} L^1_{m'}(z) L^1_{m''}(z) + z^2 \sum_{i=0}^{\infty} (m_i + 1) L^1_{m_i}(z)$$

$$L^1_{(h+1),m}(z) = [h = 0 + \bigwedge_{i=0}^{\infty} m_i = 0] + \sum_{m' @ m'' = m} z L^1_{m'}(z) L^1_{m''}(z)$$

Then we get as associated double series:

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

The definition of the coefficients $a^\nu_m$ 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 $a^\nu_m$ until a value upBound for $n$. We use a recursive data structure:

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

in which we store the computed values of a function

```haskell
a :: Int -> [Int] -> Integer
```

In our implementation the depth of the recursion of $\text{Mem}$ is limited by $\text{upBound}$, which is also the longest tuple $\text{m}$ for which we will compute $a^\nu_m$. Associated with $\text{Mem}$ there is a function

```haskell
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 $\text{Load}$, contains the values of the function:

```haskell
memory :: Int -> [Int] -> Mem
memory 0 m = Load [a 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

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

and the access to $\text{theMemory}$ is given by a specific function:

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

Notice that $\text{a}$ and $\text{acc}$ have the same signature. This is not a coincidence, since $\text{acc}$ accesses values of $\text{a}$ already computed. Now we are ready to express $\text{a}$:

```haskell
a 0 m = iv (head m == 1 && all ((==) 0) (tail m))
a n m = aAPP n m + aABSwB n m + aABSnB n m
```

- $\text{aAPP}$ counts affine terms that are applications:

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

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

```haskell
aABSwB n m
| head m == 0 = sum [aABSTd n m i |i<-[1..(n-1)]]
| otherwise = 0
```

- $\text{aABStD}$ counts affine terms that are abstractions with binding at level $i$:

```haskell
aABSTd n m i = (fromIntegral (1 + m!)!!i))\((acc (n-1) (tail (inc i m) ++ [0]))
```

- $\text{aABSnB}$ counts affine terms that are abstractions with no binding:

```haskell
aABSnB 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 $a^\nu_{n,0^n}$, for instance, we need to compute $a^\nu_r$ 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.

\[
\text{amg :: Int -> [Int] -> [SwissCheese]}
\]

\[
\text{amg 0 m = if (head m == 1 && all (==) 0) (tail m)) then [Box 0] else []}
\]

\[
\text{amg n m = allAPP n m ++ allABwB n m ++ allABnB n m}
\]

\[
\text{allAPP :: Int -> [Int] -> [SwissCheese]}
\]

\[
\text{allAPP n m = foldr (+) [] (map (\((q,r),(k,nk))\rightarrow appSC (cartesian (accAG k q)
\text{(accAG nk r)}
\text{(allCombinations m (n-1))})
\]

\[
\text{allABSAtD :: Int -> [Int] -> Int -> [SwissCheese]}
\]

\[
\text{allABSAtD n m i = foldr (+) [] (map (abstract (i-1))) (accAG (n - i - 1)
\text{(tail (inc i m) ++ [0]))})
\]

\[
\text{allABwB :: Int -> [Int] -> [SwissCheese]}
\]

\[
\text{allABwB n m = if head m == 0 = foldr (+) [] [allABSAtD n m i |i<-[1..(n-1)]]
\text{| otherwise = []}
\]

\[
\text{allABnB :: Int -> [Int] -> [SwissCheese]}
\]

\[
\text{allABnB n m = if head m == 0 = map (AbsSC . raise) (accAG (n-1) (tail m ++ [0]))
\text{| otherwise = []}
\]

\[
\text{memoryAG :: Int -> [Int] -> MemSC}
\]

\[
\text{memoryAG 0 m = LoadSC [amg n (reverse m) | n<-[0..]]}
\]

\[
\text{memoryAG k m = MemSC [memoryAG (k-1) (j:m) | j<-[0..]]}
\]

\[
\text{theMemoryAG = memoryAG (upBound) []}
\]

\[
\text{accAG :: Int -> [Int] -> [SwissCheese]}
\]

\[
\text{accAG n m = accessSC theMemoryAG n m}
\]

From this, we get programs for generating random affine terms or random linear terms as follows: if we want a random closed linear term of a given size \( n \), we throw a random number, say \( p \), between 1 and \( l_{\text{lin}} \) and we look for the \( p^{th} \) in the list of all the closed linear terms of size \( n \). Haskell laziness mimics the unranking. Due to high requests in space, we cannot go further than the random generation of closed linear terms of size 23 and closed affine terms of size 19. There are similar programs for generating all the terms of size \( n \) for variable size 0 and variable size 1.

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 $anf_{n,m}^{\nu}$ the numbers of affine SwissCheeses with no $\beta$-redex and $ane_{n,m}^{\nu}$ the numbers 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:

- $anf_{n,m}^{\nu}$ the number of affine SwissCheeses with no $\beta$-redex which are abstraction with a binding of a de Bruijn index,
- $anf_{n,m}^{\nu}$ the number of affine SwissCheeses with no $\beta$-redex which are abstraction with no binding.

\[
anf_{0,m}^{\nu} = ane_{0,m}^{\nu}
anf_{n+1,m}^{\nu} = ane_{n+1,m}^{\nu} + anf_{n+1,m}^{\nu} + anf_{n+1,m}^{\nu}
\]

where

\[
ane_{0,m}^{\nu} = m_0 = 1 \land \bigwedge_{j=1}^{p-1} m_j = 0
\]

\[
ane_{n+1,m}^{\nu} = \sum_{q \in \mathbb{R} = 0:m} \sum_{k=0}^{n} ane_{q,m}^{n-k,r}
\]

and

\[
anf_{n+1,m}^{\nu} = \sum_{i=0}^{n} (m_i + 1) anf_{n-i,m}^{\nu}
\]

and

\[
anf_{n+1,m}^{\nu} = anf_{n,m}^{\nu}
\]

There are two generating functions, $A^{anf}$ and $A^{ane}$, which are associated to $anf_{n,m}^{\nu}$ and $anf_{n,m}^{\nu}$:

\[
A^{anf} (z, u) = A^{ane} (z, u) + \sum_{i=1}^{\infty} z^i \frac{\partial A^{anf} (z, \text{tail} (u))}{\partial u} + z A^{anf} (z, \text{tail} (u))
\]

\[
A^{ane} (z, u) = u_0 + z A^{ane} (z, u) A^{anf} (z, u)
\]

Linear closed normal forms

Let us call $lnf_{n,m}^{\nu}$ the numbers of linear SwissCheeses with no $\beta$-redex and $lne_{n,m}^{\nu}$ the numbers 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 $lnf_{n,m}^{\nu}$ the number of linear SwissCheeses with no $\beta$-redex which are abstraction with a binding of a de Bruijn index.

\[
lnf_{0,m}^{\nu} = lne_{0,m}^{\nu}
lnf_{n+1,m}^{\nu} = lne_{n+1,m}^{\nu} + lnf_{n+1,m}^{\nu}
\]
where

\[ lne^0_{0,m} = m_0 = 1 \land \bigwedge_{j=1}^{p-1} m_j = 0 \]

\[ lne^0_{n+1,m} = \sum_{q \oplus r = 0}^{m} \sum_{k=0}^{n} lne^0_{k,q} lnf^0_{n-k,r} \]

and

\[ lnf^\nu_{n+1,m} = \sum_{i=0}^{n-1} (m_i + 1) lnf^\nu_{n-i,m} \]

with the two generating functions:

\[ L^{nf,\nu}(z, u) = L^{ne,\nu}(z, u) + \sum_{i=1}^{\infty} z^i \frac{\partial L^{nf,\nu}(z, \text{tail}(u))}{\partial u^i} \]

\[ L^{ne,\nu}(z, u) = u_0 + z L^{ne,\nu}(z, u) L^{nf,\nu}(z, u) \]

We also deduce programs for generating all the closed affine or linear normal forms of a given size from which we deduce programs for generating random closed affine or linear normal forms of a given size. For instance, here are three random linear closed normal forms (using de Bruijn indices) of natural size 28:

\[ \lambda \lambda \lambda \lambda (2 \lambda ((1 2) \lambda (0 (5 1)))) \lambda (0 \lambda \lambda (1 \lambda ((0 (2 \lambda \lambda ((1 \lambda 0) 0))) 1)) \lambda ((0 \lambda 0) \lambda \lambda ((0 (1 \lambda 0) (1 \lambda (0 \lambda 0)))) \lambda 0)) \]

8.2 Variable size 0

Linear closed normal forms

A little like previously, let us call \( lnf^0_{n,m} \) the numbers of linear SwissCheeses with no \( \beta \)-redex and \( lne^0_{n,m} \) the numbers 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 \( lnf^0_{\lambda w_{n+1,m}} \) the number of linear SwissCheeses with no \( \beta \)-redex which are abstraction with a binding of a de Bruijn index. We assume that the reader knows now how to proceed.

\[ lnf^0_{0,m} = lne^0_{0,m} \]

\[ lnf^0_{n+1,m} = lne^0_{n+1,m} + lnf^0_{\lambda w_{n+1,m}} \]

where

\[ lne^0_{0,m} = m_0 = 1 \land \bigwedge_{j=1}^{p-1} m_j = 0 \]

\[ lne^0_{n+1,m} = \sum_{q \oplus r = 0}^{m} \sum_{k=0}^{n} lne^0_{k,q} lnf^0_{n-k,r} \]

\[ lnf^0_{\lambda w_{n+1,m}} = \sum_{i=0}^{n} (m_i + 1) lnf^0_{n,m} \]

12
and the two generating functions:

\[
\mathcal{L}^{nf,0}(z, u) = \mathcal{L}^{ne,0}(z, u) + \sum_{i=1}^{\infty} \frac{\partial \mathcal{L}^{nf,0}(z, \text{tail}(u))}{\partial u^i}
\]

\[
\mathcal{L}^{ne,0}(z, u) = u_0 + z \mathcal{L}^{ne,0}(z, u) \mathcal{L}^{nf,0}(z, u)
\]

With no surprise we get for \( \ln f^{0,n}_0 \) the sequence:

\[
0, 1, 0, 3, 0, 26, 0, 367, 0, 7142, 0, 176766, 0, 5304356, \ldots
\]

mentioned by Zeilberger in [19] and listing the coefficients of the generating function \( \mathcal{L}^{nf,0}(z, 0^w) \).

We let the reader deduce how to count closed affine normal forms for variable size 0 and closed linear and affine normal forms for variable size 1 alike. Notice that the Haskell programs are on the GitHub site.

\section{Related works and Acknowledgement}

There are several works on counting \( \lambda \)-terms, for instance on natural size [3, 2], on variable size 1 [5, 10, 18], on variable size 0 [14], on affine terms with variable size 1 [7, 6], on linear \( \lambda \)-terms [21, 19, 20], also on a size based binary representation of the \( \lambda \)-calculus [15] (see [12] for a synthetic view of both natural size and binary size).

The basic idea of this work comes from a discussion with Maciej Bendkowski, Olivier Bodini, Sergey Dovgal and Katarzyna Grygiel, I thank them as I thank Noam Zeilberger for interactions.

\section{Conclusion}

This presentation shares similarity with this of [14, 15, 4]. Instead of considering the size \( n \) and the bound \( m \) of free indices like in expressions of the form:

\[
T_{n+1,m} = T_{n,m+1} + \sum_{i=0}^{n} T_{i,m} T_{n-i,m}
\]

here we replace \( m \) by the characteristic \( m \). We can imagine a common framework. On another hand, as noticed by Paul Tarau, this approach has features of dynamic programming [9], which makes it somewhat efficient.

\section{References}


Figure 3: Natural size: numbers of closed linear terms of size \( n \) from 0 to 100

Data

In the appendix, we give the first values of \( b_{n,0}^\nu \), \( a_{n,0}^\nu \), and \( anf_{n,0}^\nu \).
### Figure 4: Natural size: numbers of closed affine terms of size $n$ from 0 to 100

<table>
<thead>
<tr>
<th>$n$</th>
<th>Numbers of Closed Affine Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>25</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
</tr>
<tr>
<td>9</td>
<td>166</td>
</tr>
<tr>
<td>10</td>
<td>405</td>
</tr>
<tr>
<td>11</td>
<td>1050</td>
</tr>
<tr>
<td>12</td>
<td>2763</td>
</tr>
<tr>
<td>13</td>
<td>7239</td>
</tr>
<tr>
<td>14</td>
<td>19190</td>
</tr>
<tr>
<td>15</td>
<td>51457</td>
</tr>
<tr>
<td>16</td>
<td>13858</td>
</tr>
<tr>
<td>17</td>
<td>374972</td>
</tr>
<tr>
<td>18</td>
<td>1020943</td>
</tr>
<tr>
<td>19</td>
<td>2792183</td>
</tr>
<tr>
<td>20</td>
<td>7666358</td>
</tr>
<tr>
<td>21</td>
<td>21126905</td>
</tr>
<tr>
<td>22</td>
<td>58422650</td>
</tr>
<tr>
<td>23</td>
<td>162052566</td>
</tr>
<tr>
<td>24</td>
<td>450742451</td>
</tr>
<tr>
<td>25</td>
<td>1256074900</td>
</tr>
<tr>
<td>26</td>
<td>3513731861</td>
</tr>
<tr>
<td>27</td>
<td>943728012</td>
</tr>
<tr>
<td>28</td>
<td>2763408879</td>
</tr>
<tr>
<td>29</td>
<td>77721141911</td>
</tr>
<tr>
<td>30</td>
<td>218984204904</td>
</tr>
<tr>
<td>31</td>
<td>618821576627</td>
</tr>
<tr>
<td>32</td>
<td>1746906189740</td>
</tr>
<tr>
<td>33</td>
<td>495402608426</td>
</tr>
<tr>
<td>34</td>
<td>14017220713131</td>
</tr>
<tr>
<td>35</td>
<td>39784696510433</td>
</tr>
<tr>
<td>36</td>
<td>11305753020242</td>
</tr>
<tr>
<td>37</td>
<td>321649935953313</td>
</tr>
<tr>
<td>38</td>
<td>916098061668770</td>
</tr>
<tr>
<td>39</td>
<td>261184703888831</td>
</tr>
<tr>
<td>40</td>
<td>7435859187221508</td>
</tr>
<tr>
<td>41</td>
<td>21292775008988858</td>
</tr>
<tr>
<td>42</td>
<td>60875581617670699</td>
</tr>
<tr>
<td>43</td>
<td>1741959615293355850</td>
</tr>
<tr>
<td>44</td>
<td>498866759031591507</td>
</tr>
<tr>
<td>45</td>
<td>142975338365525063</td>
</tr>
<tr>
<td>46</td>
<td>41007933353241163138</td>
</tr>
<tr>
<td>47</td>
<td>11769771167532816128</td>
</tr>
<tr>
<td>48</td>
<td>33804054749367200891</td>
</tr>
<tr>
<td>49</td>
<td>97151923336684220063</td>
</tr>
<tr>
<td>50</td>
<td>27938597720772581435</td>
</tr>
</tbody>
</table>

$0.80328779462727941247$ $231462127904669382002$ $66678104363560967142481$ $1921841105885449257906$ $55421020166161024658070$ $1598992183211973811984561$ $461557020400062903560120$ $13329209088595128118290519$ $38510297068336583693412910$ $1131032444503136571789527$ $321865581221116909667632929$ $931084100480806285466998584$ $2694461970241142079402003$ $780043726184601367932458866$ $2259043189995513515903949650$ $654461295539052336187266873$ $189667372181089716881014445025$ $5498523629827005745776929352$ $159455735759384637878305531$ $406256284802443357249611323484$ $1342251884451664733064823521627$ $3896065662221720062565314400538$ $1131211774880577210479520337816$ $32853646116456432492645965741531$ $9544251463348924605558901961967438$ $277342191547303839604289978813667$ $80612581945729102863848267463755$ $2343682130911232279285707920060154$ $681556402376534208079367816340359$ $1982841232214577566417303371819466$ $57769013022808238913186414092851856$ $1678517878025139846484284864390041$ $488574368667083292343802790464796207$ $1422426342380888254459783410485365006$ $414210456408904420390199081775864665$ $120943058576505936787852691159653106$ $3514564781152987371433737643774738767$ $102460303505212909760205354818010948705$ $2984460259586120521621708978503365560464$ $86993545270523032108316419492803990417$ $25362294927043545133939906228529728411158$ $739518143425160075353976766029755947391$ $2156877648677019271342151422781445937890$ $62915493955623036556255998997702469004382816$ $183553775888862113259618510513062632416356$ $5356000661251556969155435449628266552079$ $1563109720672569189989636662620867515144$ $4562542818801138452310024132123304180969233$ $133196302866298656173865997447273781972745$ $100895020391341859449843201188612375391453776$
0 0 41 303784364560
1 0 42 6895841598915
2 1 43 15666498585568
3 1 44 35620848278448
4 2 45 81052838239593
5 3 46 184564847153821
6 7 47 420564871255118
7 10 48 95897584645984
8 20 49 21880639292104
9 40 50 4995528560788451
10 77 51 11411921511827547
11 160 52 26084524952754538
12 318 53 5965468288899245
13 671 54 136500653554490261
14 1405 55 312496494161999851
15 2981 56 715760763686417314
16 6312 57 1640194881084692664
17 13672 58 376028478791736081
18 29399 59 8624561382665096780
19 63697 60 1978963994429966346
20 139104 61 4542733708377920041
21 304153 62 10432043866803481453
22 667219 63 239656248361374562433
23 1469241 64 550769764273325683328
24 3247176 65 1266217777460030829940
25 7184288 66 2912050679107531357883
26 15949179 67 669941890986008663265
27 35480426 68 15417663698158012992010
28 79083472 69 35492710197462925262295
29 176607519 70 81732521943462960197057
30 395119875 71 18827063630689910161436
31 885450388 72 43380713501277479924026
32 1987289740 73 999851681931974680766994
33 4466760570 74 2305129188866501774481545
34 10053371987 75 5315847657531578072941600
35 22656801617 76 12262083079763230881047944
36 51121124910 77 28292248929584657512609357
37 115478296639 78 65294907440897718078048829
38 261139629999 79 1507290704376702817820543
39 591138386440 80 34803101557737732605480908

Figure 5: Natural size: numbers of closed affine normal forms of size $n$ from 0 to 80