Linear numeral systems

Ian Mackie

Abstract We investigate numeral systems in the lambda calculus; specifically in the linear lambda calculus where terms cannot be copied or erased. Our interest is threefold: representing numbers in the linear calculus, finding constant time arithmetic operations when possible for successor, addition and predecessor, and finally, efficiently encoding subtraction—an operation that is problematic in many numeral systems. This paper defines systems that address these points, and in addition provides a characterisation of linear numeral systems.

1 Introduction

This paper is about numeral systems in the linear λ-calculus. By a numeral system we mean a mathematical notation (in our case linear λ-terms) for representing both numbers and arithmetic operations. By linear we mean that the λ-terms do not copy or erase arguments. For the purpose of this paper, the arithmetic operations that we are interested in are restricted to: successor, addition, predecessor, subtraction and a test for zero.

The λ-calculus is a model of computation that can represent both programs and data. Representing numbers is an essential step towards proving that the calculus is adequate for capturing all computable functions. When the λ-calculus was introduced, Church [2] gave what are now known as Church numerals, which is the most well-known numeral system. This representation supports all the usual arithmetic operations (successor, addition, predecessor, etc.), but some operations are more efficient (in terms of β-reductions) than others. Specifically for Church numerals, the predecessor operation is costly, and we give a summary later to show that other numeral systems can encode this operation in a more efficient way, but at the cost of introducing a less efficient way of computing one of the other operations. Here we are interested in the linear λ-calculus. This is a proper subset of the usual calculus, where explicit copying and erasing (by using variables in the body of a term more

E-mail: iancmackie@gmail.com

than once or not at all, respectively) are not permitted. All functions must use their arguments exactly once, and as a consequence the calculus has extremely weak expressive power—all functions normalise in linear time. In the linear case, β-reduction is a constant time operation, so counting reductions gives a cost metric. It is important to note that if a function is definable in the linear λ-calculus it will be constant or linear time. This makes the linear λ-calculus an interesting framework for encoding arithmetic. However, it comes as no surprise that there is no notion of adequacy for linear numeral systems (the ability to represent all computable functions). The reason for this is that iteration and recursion are not linearly definable. This makes the definition of addition and subtraction a difficult problem that we address in this paper.

We explore the limits of this system. Once we have defined linear numerals and the arithmetic functions that are linearly definable, it is then possible to add non-linear functions to compute the remaining functions as usual. Our interest here is therefore:

- to define suitable representations (data-types) of numerals in the linear λ-calculus;
- and to define linear functions over these representations to compute:
  - successor, addition and predecessor that are *constant time*;
  - subtraction and a test for zero that are as *efficient as possible*.

It turns out that for all the systems we define, successor, addition and predecessor all must be constant time in fact (we cannot define anything less efficient). The test for zero and subtraction both require erasing, and this is where some cost is introduced. However, they are both worst case linear time. Indeed, erasing is one of the main issues that we must deal with in a linear setting. A test for zero will return a Boolean result (true or false), and consequently the number must be erased. A similar situation arises with subtraction, where part of a number must be erased. Consequently, neither of these operations can be constant time in a linear framework.

The main contributions in this paper are:

- a series of linear numeral systems, leading to a characterisation of linear numerals;
- constant time successor, predecessor and addition operators;
- subtraction that has cost $min(m,n)$;
- test for zero that is linear in the size of the term.

Related work. There has been a great wealth of work since the λ-calculus was introduced on representing data; specifically numeral systems. The λI-calculus [1], where abstractions must bind at least one variable was used, and several systems have been adapted to work in this way.

There are many numeral systems in the literature (and many more used as exercises in λ-calculus courses). Apart from the most well-known system of Church already mentioned, we find in the literature a system by Scott (first reported in [3]), and some so-called unusual ones by Wadsworth [10] (also developed by Böhm). With an emphasis on constant time operations, there is the work of Parigot and Rozière [8], which however doesn't focus on linearity, but is the work closest to ours in spirit.

From a very different perspective, and a focus on compact representation, there are binary representations [7] that give logarithmic size numbers. However, as we

show later, these are not linearly definable. There is also the work of Böhm and Dezani (see for example [9]) where numbers are represented by non-terminating λ-terms, but again these are not possible in the linear λ-calculus.

Rezus [9] and Barendregt [1], contain several other systems that we will not enumerate here, where one of the main concerns is adequacy. Linear systems are never going to be adequate, but for us the focus is on time efficient representations. We provide a selection of systems. Some are closely based on existing systems (for example those defined by Rezus [9]), adapted to a linear setting, and some other ones are new.

Overview. The rest of this paper is structured as follows. We first recall some background concepts for the λ-calculus and numeral systems. In Section 3 we discuss ways of representing numbers generally, so that they can be applied to our setting. In Section 4 we define some linear numeral systems, specifically one of the main contributions of the paper: a linear system that allows constant time operations as well as subtraction. In Section 5 we give a characterisation of linear numeral systems. We discuss these systems and conclude in Section 6.

2 Background

We assume familiarity with standard λ-calculus concepts, including notions of free variables (FV), substitution ($t[u/x]$), β- and η-reduction. We refer the reader to [1,5] for additional background to these. To fix the syntax we start with a definition.

Definition 1 (Linear terms) Assume an infinite set $\mathscr{X}$ of variables denoted by x, y, z, ..., then the set Λ_L of linear λ-terms is the least set satisfying:

1. $x \in \mathscr{X} \Rightarrow x \in \Lambda_L$ (variable)
2. $t \in \Lambda_L, x \in \mathrm{FV}(t) \Rightarrow (\lambda x.t) \in \Lambda_L$ (abstraction)
3. $t,u \in \Lambda_L, \mathrm{FV}(t) \cap \mathrm{FV}(u) = \varnothing \Rightarrow (tu) \in \Lambda_L$ (application)

where $\mathrm{FV}(x) = \{x\}$, $\mathrm{FV}(\lambda x.t) = \mathrm{FV}(t) - \{x\}$, $\mathrm{FV}(tu) = \mathrm{FV}(t) \cup \mathrm{FV}(u)$. Whenever $\mathrm{FV}(t) = \varnothing$, then the term t is *closed*.

In this definition, the free variable constraints ensure linearity of terms by construction. Application associates to the left and abstraction binds as far to the right as possible. Consequently we will economise on parentheses whenever possible. Examples of linear terms include: $I = \lambda x.x$, $B = \lambda xyz.x(yz)$, $C = \lambda xyz.xzy$, and function composition: $t \circ u = Btu$. Notable non-linear terms are $S = \lambda xyz.xz(yz)$ and $K = \lambda xy.x$, which incorporate explicit copying and erasing of arguments respectively.

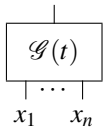
The λI-calculus [1] is an intermediate calculus that does not allow erasing, but does allow copying (this is analogous to relevance logic), and we could define an affine calculus that allows erasing but not copying. Thus the linear λ-calculus is the most constrained calculus with respect to resource usage, and related to linear logic.

Every linear term is typeable, and an alternative way to write the variable constraints of Definition 1 is the following typing system, where $\multimap$ is linear implication (functions that use their arguments exactly once):

$$\frac{}{x : A \vdash x : A} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B} \qquad \frac{\Gamma \vdash t : A \multimap B \qquad \Delta \vdash u : A}{\Gamma, \Delta \vdash tu : B}$$

In this system the contexts Γ and Δ are used to guarantee the linearity constraints. Types will not play much role in the following, but it is worth remembering that all linear terms are typeable.

Occasionally, it will be illuminating to see linear terms drawn as graphs. We define $\mathscr{G}(\cdot)$ inductively over the structure of a linear term t. The general structure is given by the following diagram, where $\mathrm{FV}(t) = \{x_1, \ldots, x_n\}$:



The three diagrams given in Figure 1 show the translation, respectively, of a variable $\mathscr{G}(x)$, which is an edge, an abstraction $\mathscr{G}(\lambda x.u)$, which introduces a new node λ to bind the unique occurrence of the variable x (that we assume is left-most), and finally an application $\mathscr{G}(uv)$ introduces a new node @.

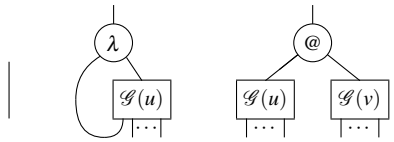


Fig. 1 Translation of linear terms into graphs

To give the general idea of the graphs, we give two examples of closed terms: $\mathscr{G}((\lambda x.x)(\lambda x.x))$ and $\mathscr{G}(\lambda xy.yx)$, which are shown in Figure 2.

Definition 2 (Reduction) There are two reduction rules:

$$\beta : (\lambda x.t)u \to t[u/x]$$
$$\eta : \lambda x.tx \to t$$

Linear β-reduction is a constant time operation (there are implementations where substitution has no cost). The number of β-reductions is therefore a reasonable measure of the cost of a computation. η-reduction does not need a side-condition, as there cannot be another occurrence of x in t. Linear terms are stable under β- and

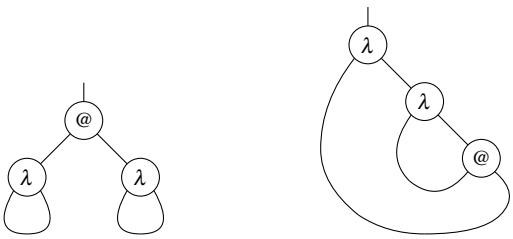


Fig. 2 Example graphs: $\mathscr{G}((\lambda x.x)(\lambda x.x))$ and $\mathscr{G}(\lambda xy.yx)$

η-reduction: if $t \in \Lambda_L, t \to u$ then $u \in \Lambda_L$. We occasionally mention η-reduction, but all reductions will be β-reductions unless explicitly labelled otherwise. The reflexive, transitive closure is denoted $\to^*$, and we write $t \to^n u$ meaning that t reduces to u in exactly n steps.

If no reduction can take place in a term, then the term is a normal form. Reduction in the linear λ-calculus is strongly normalising and confluent. Consequently, all terms have a normal form (and therefore also a head normal form). Moreover, all terms are (simply) typeable, and reduction preserves types.

We can now show a number of properties of terms and reduction that will be of use later. First, it is useful to characterise terms in the following way.

Lemma 1 *1. Each term is a variable, an abstraction term or an application term.*
2. *Each application term is of the form $t_1 \dots t_n$, where $n \geq 2$ and t_1 is not an application term.*
3. *Each abstraction term is of the form $\lambda x_1 \dots x_n.t$, where $n \geq 1$ and t is not an abstraction term.*

As an almost direct consequence of Lemma 1, we have the following result about the shape of normal forms. This is standard result, but it is important for later results so we provide some additional details here:

Lemma 2 *Every closed linear term t in normal form has the shape:*

$$\lambda x_1 \dots x_n.x_i t_1 \dots t_m$$

where $n \geq 1$, $m \geq 0$, $1 \leq i \leq n$, and $t_1, \dots, t_m$, are all normal forms.

Proof Since t is closed it cannot be a variable. Neither can it be an application term because if it were, then t_1 in $t_1 \dots t_n$ would have to be an abstraction, and thus there is a redex contradicting that this is a normal form. Thus t must be an abstraction term, say $\lambda x_1 \dots x_n.u$, $n \geq 1$. Since u cannot be an abstraction it must either be a variable in which case it must be one of $x_i, 1 \leq i \leq n$, or it is an application term $v_1 \dots v_m$, $m \geq 2$. Now v_1 must be a variable, which again must be one of the $x_i, 1 \leq i \leq n$, because if it were an abstraction it would not be a normal form. Thus the only possibilities are instances of the above. □

A convenient metric on linear terms is the size of the term.

Definition 3 (Size of linear term) We define $|\cdot|$ over the structure of terms:

$$\begin{aligned} |x| &= 1 \\ |\lambda x.t| &= |t| + 1 \\ |tu| &= |t| + |u| + 1 \end{aligned}$$

We can now glean some insight into the structure of linear terms, and monitor the structure (and consequently the size) of a term under reduction. The following properties are key to our ideas and have a lot of say in the following about the representation of numeral systems and operations that can be encoded.

Lemma 3 *1. Let t be a closed linear term, then $|t| = 3k+2$ for some $k \geq 0$.*

2. *For any linear terms t and u, where $x \in FV(t)$, $|t[u/x]| = |t| + |u| - 1$.*
3. *If $t \to u$ then $|t| = |u| + 3$. Consequently, if $t \to^n u$, then $|t| = |u| + 3n$.*

Proof 1. We prove a stronger result: if t is a linear term with $\mathrm{FV}(t) = \{x_1, \ldots, x_i\}$, then $|t| = 3k + 2 - i$. We proceed by induction on the structure of t. If t is a variable, say x then we have $i = 1$ and the result follows by setting $k = 0$. If t is an application, say (uv), where $\mathrm{FV}(u) = \{x_1, \ldots, x_m\}$ and $\mathrm{FV}(v) = \{y_1, \ldots, y_n\}$, then by the induction hypothesis twice, we have $|(uv)| = 3(k_1 + 2 - m) + 3(k_2 + 2 - n) + 1$, which we can reorganise as $3(k_1 + k_2 + 1) + 2 - (m + n)$ as required. Finally, if t is an abstraction, say $\lambda x.u$, where $\mathrm{FV}(u) = \{x, x_1, \ldots, x_n\}$, then by the induction hypothesis we have $|(\lambda x.u)| = (3k + 2 - (i + 1)) + 1$, which simplifies to $3k + 2 - i$ as required. The result follows because when t is closed, $i = 0$.

2. The unique occurrence of x in t has size 1 which will be replaced by u.
3. Each β-reduction $(\lambda x.t)u \to t[u/x]$ removes an application, abstraction and a variable. Thus, $|(\lambda x.t)u| = |t| + |u| + 2$, and $|t[u/x]| = |t| + |u| - 1$. Note that the same is true for η-reduction. □

As a corollary, reduction is easily seen to be strongly normalising. This result tells us something very important about representing numbers in the linear λ-calculus, and gives us some hints and a start in the process of characterising linear numeral systems that we will return to later in the paper. In particular, it tells us that any encoded number must be proportional in size to the number it represents, and that operations such as addition must be constant in time. Thus compact representations, such as binary numerals, cannot be defined as each successor cannot be represented by a constant operation. We will return to this later in the paper.

We recall now two standard notations for repeated applications.

Definition 4 Let $t, u \in \Lambda_L$, then

$$\begin{array}{ll} tu^{\sim 0} = t & tu^{\sim n+1} = tu^{\sim n}u \\ t^0 u = u & t^{n+1}u = t(t^n u) \end{array}$$

An important structure that will be used throughout the paper is the following representation of a list of terms, suitably adapted to the linear λ-calculus.

Definition 5 (Linear sequences) Let $t_1, \ldots, t_n$ be linear terms, then $\langle t_1, \ldots, t_n \rangle = \lambda z.zt_1 \ldots t_n$ is a sequence of linear terms.

In particular, we note that the empty sequence is $\langle\,\rangle = \lambda z.z$, and the singleton is $\langle t \rangle = \lambda z.zt$. Function composition ($t \circ u = Btu$) gives concatenation of sequences in four β-reductions:

Lemma 4 $\langle t_1, \ldots, t_n \rangle \circ \langle u_1, \ldots, u_m \rangle \to^4 \langle u_1, \ldots, u_m, t_1, \ldots, t_n \rangle$

If we were interested in preserving the order of the elements in the sequence, then we can define a variant of this using $B' = \lambda xyz.y(xz)$ rather than B. However, we will see that the use we make of this result always joins sequences of the same elements, so the results are equivalent.

The following is a standard definition of a numeral system that we have adapted for the linear case. We restrict to the case where each numeral is a term in normal form, as the linear λ-calculus is terminating.

Definition 6 (Linear numeral system) A numeral system is a sequence of λ-terms: $d = d_0, d_1, \ldots$, such that

1. each d_n is a closed term in normal form;
2. there exist terms S (representing successor), and Z (representing the test for zero) and two different terms T and F (representing the Booleans true and false), that satisfy $Sd_n \to^* d_{n+1}$, $Zd_0 \to^* T$ and $Zd_{n+1} \to^* F$.

Furthermore, we call a numeral system *linear* when:

3. each d_n is a closed linear term;
4. the terms S, Z, T and F are linear;
5. addition, predecessor and subtraction are all linearly definable.

The first two points in the above definition are enough to give an adequate numeral system in the full λ-calculus, but we will need to use iteration or recursion to build the arithmetic functions. Because neither iteration nor recursion are linear definable, it is not possible to get an adequate system. Therefore we have chosen to include in the definition of linear numeral system the requirement to support linear addition, predecessor and subtraction. As a consequence several systems that we put forward in this paper are not linear numeral systems according to this definition, and we will call those linear systems 'candidates'. Therefore, in some candidate systems we present, the numbers may be represented by linear terms, but the arithmetic functions would be non-linear terms.

To summarise, in this paper we are interested in the following arithmetic operations: successor, addition, predecessor, subtraction and test for zero, because these are linearly definable.

3 Representing numbers

We can represent numbers as algebraic terms (i.e., first order syntax). But the introduction of binders (i.e., higher order) gives some new ideas. We briefly recall some of these ideas in this section, and then review some well-known numeral systems in the (non-linear) λ-calculus.

3.1 Numbers

The standard inductive data type for numbers requires a zero (0) and a successor (S). We can then write numbers as 0, $S(0)$, $S(S(0))$, etc., and we can write recursive definitions over this data type. For example addition and subtraction can be defined as:

$$\begin{array}{llll} \mathsf{add}\ 0\ n & = n & \mathsf{sub}\ m\ 0 & = m \\ \mathsf{add}\ (Sm)\ n & = S(\mathsf{add}\ m\ n) & \mathsf{sub}\ (Sm)\ (Sn) & = \mathsf{sub}\ m\ n \end{array}$$

where sub is a partial function. Typically, there is more than one way to write such functions, for example the second case of addition can also be defined as $\mathsf{add}\ (Sm)\ n = \mathsf{add}\ m\ (Sn)$. Other alternatives can do recursion on the second argument, etc. All these

variants give the same answer, but some require (a lot) more computation steps than others. With this inductive data type, add has linear complexity: it depends on the first argument. So for n large, $\mathsf{add}\ 0\ n$ is a very different computation to $\mathsf{add}\ n\ 0$. Consequently, it is difficult to know the effect of commutativity of operations. Subtraction is also a linear function: it depends on the number being subtracted. However, the definition is more complicated, pattern-matching on both arguments, and it only works when the second number is smaller than the first (because we don't have negative numbers). In this notation, we can write a predecessor function and a test for zero both as constant time functions.

The reason that add is so inefficient is because we are essentially concatenating two lists of successors. It is well-known that we can do concatenation much better than this—constant time append operations can be achieved if we just keep a pointer to the start and end of each list. This idea is easy to implement with languages with binders, in our case λ-terms. Specifically, we can represent numbers as $\lambda x.S^n x$, so $0 = \lambda x.x$, $1 = \lambda x.Sx$, $2 = \lambda x.S(Sx)$, etc. In this representation, there is a constant time addition operation (four β-reductions), which is just composition:

$$m \circ n = (\lambda xyz.x(yz))(\lambda x.S^m x)((\lambda x.S^n x)z) \to^2 \lambda z.(\lambda x.S^m x)((\lambda x.S^n x)z) \to^2 \lambda z.S^{m+n} z$$

Equally important, $n \circ m \to^4 n+m$ also, so commutativity does not change the cost of computation. So if we can find a representation of S as a λ-term, then we have an efficient numeral system. In the λ-calculus, a λ can be thought of as a constructor and an application as a destructor (dually, we can also think of application as the constructor, and the λ as the destructor). Thus we have essentially a function/constructor system. We see later that Church numerals work in this way, so they have a constant time addition, but Scott numerals do not because they correspond closer to the inductive data type given at the start of this section. Thus, the complexity of the addition operation depends crucially on the chosen representation.

What about the predecessor and subtraction functions? These bring additional issues. Consider first the predecessor: $\mathsf{pred}\ 0 = 0$, $\mathsf{pred}\ (Sn) = n$. With this definition, we can show $\mathsf{pred}(Sx) = x$, but we cannot simplify $S(\mathsf{pred}\ x)$. This is a problem for constructor systems, and also an issue for us. From these examples we conclude that constant time functions should be possible in some cases.

If addition is like concatenation of lists, then subtraction is like splitting a list. Consequently part of the list will need to be erased, which is not straightforward in a linear system.

There are other ways to represent numbers as λ-terms, for example binary representations (see for example [7]). However, the results of this paper show that these representations are not possible in the linear λ-calculus, and we discuss this topic in more detail in Section 5. We next review four well-known numeral systems in the λ-calculus. None of these are linear but we will refer to them later for inspiration.

We assume for the rest of this section the usual (non-linear) λ-calculus, and standard Booleans: $T = \lambda xy.x, F = \lambda xy.y$. We write succ, add, pred, sub, and zero to represent the successor, addition, predecessor, subtraction and test for zero.

3.2 Church numerals

Church numerals [2] encode numbers with repeated application: $\lambda xf.f^n x$. Thus, $d = \lambda xf.x, \lambda xf.fx, \lambda xf.f(fx), \ldots$. Each number is an iterator, where a binder represents the iterated term, and a second binder is a place-holder at the end of the list of applications. Consequently, Church numerals can be seen as lists with a pointer to the end of the list, and thus we can predict that an efficient addition should be possible in this representation as discussed in Section 3.1

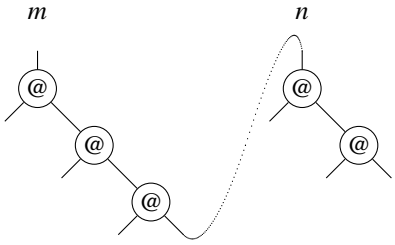
The arithmetic operators can be defined by:

$$\begin{aligned}
\mathsf{succ} &= \lambda nxf.f(nxf) \\
\mathsf{add} &= \lambda mnxf.m(nxf) \\
\mathsf{pred} &= \lambda nxf.n(\lambda y.x)(\lambda pq.q(pf))I \\
\mathsf{sub} &= \lambda mn.mn\ \mathsf{pred} \\
\mathsf{zero} &= \lambda n.nT(\lambda y.F)
\end{aligned}$$

There are a number of alternative definitions of these operations. For example, the successor function can also be defined as $\mathsf{succ}' = \lambda nxf.n(fx)f$. The difference between the two is where we add an extra iteration: at the front or at the end of a list.

- $\mathsf{succ}\ (\lambda xf.f^n(x)) \to^* \lambda xf.f(f^n(x))$
- $\mathsf{succ}'\ (\lambda xf.f^n(x)) \to^* \lambda xf.f^n(fx)$

An alternative addition is: $\mathsf{add}' = \lambda mn.mn\ \mathsf{succ}$. This is a lot more expensive, as it iterates the successor function m times. The one given above is constant time however, because we have access to the end of the list. To see this, consider $m = \lambda xf.f(f(fx))$, $n = \lambda xf.f(fx)$. The resulting term is the addition, so $\lambda xf.f(f(f(f(fx))))$. We depict this, showing just the applications for the numeral, and the concatenation by a dotted line:



This operation is commutative, so $\mathsf{add}\ a\ b = \mathsf{add}\ b\ a$ and moreover the result is obtained with the *same number* of reductions. There are other alternatives for predecessor, for instance using pairs, but we will need iteration or recursion for this one. Church numerals are interesting because a wealth of functions can be defined without using recursion since they have iteration built in. They offer a compact representation, and operations like addition can be efficient (constant time). But on the down side, predecessor, and therefore subtraction, are very expensive.

3.3 Wadsworth systems/Böhm systems

Wadsworth [10] presented a selection of (so-called unusual) numeral systems, based on a chain of λ-abstractions, and then a permutation of the variables. Some of these systems were also developed by Böhm. In one of these systems, n is represented as a permutation of $n+2$ variables: $n = \lambda x_1 x_2 \ldots x_{n+2}.x_1 x_2 \ldots x_n x_{n+2} x_{n+1}$. Interesting for us is that this representation is linear:

$$0 = \lambda x_1 x_2.x_2 x_1, 1 = \lambda x_1 x_2 x_3.x_1 x_3 x_2, 2 = \lambda x_1 x_2 x_3 x_4.x_1 x_2 x_4 x_3$$

However, we lose linearity with some of the operations:

$$\begin{aligned}\mathsf{succ} &= \lambda nxy.n(xy)\\ \mathsf{pred} &= \lambda n.nI\\ \mathsf{zero} &= \lambda n.n(n(KF)(KT))\end{aligned}$$

This representation has a very efficient predecessor, but unfortunately, the addition operation requires a recursive function (so the fixed point combinator). We also remark that we need two copies of the number for the zero test to trigger an η-collapse: this is how all the abstractions can be erased. Subtraction also needs to be defined as a recursive function.

3.4 Standard numerals

Barendregt [1] introduced the standard numerals. These are numbers represented using pairs: $[t,u] = \lambda z.ztu$. Thus we have the sequence: $d = I, [F,I], [F,[F,I]], \ldots$ The arithmetic functions can be defined as:

$$\begin{aligned}\mathsf{succ} &= \lambda x.[F,x]\\ \mathsf{pred} &= \lambda x.xF\\ \mathsf{sub} &= \lambda mn.nm\\ \mathsf{zero} &= \lambda x.xT\end{aligned}$$

These are all very simple operations, especially test for zero and subtraction. But addition needs general recursion, thus becomes an expensive operation.

3.5 Scott numerals

Scott numerals [3], like standard numerals, have an efficient predecessor operation. In addition, although the terms are not linear, they do not duplicate variables.

Scott numerals are given by the following idea: $d_0 = \lambda xy.x$, $d_1 = \lambda xy.y(\lambda xy.x)$, $d_2 = \lambda xy.y(\lambda xy.y(\lambda xy.x))$ ..., so each number is represented in a similar way to an inductive data type. In this representation one can define operations:

$$\begin{aligned}\mathsf{succ} &= \lambda nxy.yn\\ \mathsf{pred} &= \lambda n.n0I\\ \mathsf{zero} &= \lambda x.xT(\lambda x.F)\end{aligned}$$

Successor and predecessor are simple constant time operations, but there is no simple addition or subtraction. To encode the latter, we need iteration or recursion which are not part of a linear system. Alternatively, for addition, we would need a pointer to the end of list, which again is not part of this definition.

3.6 Summary

All of the above systems are adequate for representing all computable functions. Each system has operations that are efficient, for example, addition is a constant time operation using Church numerals, and predecessor is a constant time operation for Scott numerals. However, no system is efficient for all operations. Specifically:

- Each numeral system presented has a good feature, for example constant time addition (Church), constant time predecessor (Scott, Standard), efficient subtraction (Standard).
- Each system presented also has a bad feature: Scott numerals need recursive function to encode addition, and the predecessor in Church numerals (and therefore subtraction) is a very complicated and expensive operation.
- Many of the systems cannot encode an operation without the need for recursion machinery. Church numerals have the advantage here, as they have a built-in iterator.
- No system presented is linear, and no system has subtraction.

Our aim is therefore to understand how these numeral systems manage certain operations, and thus we pick the best features of each where we can.

4 Linear numeral systems

We now limit ourselves to the linear λ-calculus, and give several possible representations of numbers. Linear numeral systems are not new—some of the earliest systems were linear, and this is because of the popularity of the λI-calculus. However, the novelty here is that we want to characterise linear systems, and in particular look at efficient ways of computing test for zero, and also subtraction. Before we can define any, we need some ideas to simulate erasing and to represent Booleans.

4.1 Linear erasing and Booleans.

Copying can never be simulated in the linear λ-calculus because of Lemma 3, Part 3: reduction reduces the size of the term, so there is no way to build a term of arbitrary size. However, it is possible to simulate erasing. There are two ideas that we will make use of to simulate erasing in the linear λ-calculus. The first is inspired by the solvability result for the λ-calculus [1], and is based on consuming data. The second is based on pairing an answer together with the unwanted (garbage) part of the data. We first explain these in more detail and then prove the associated properties.

1. Erasing by consuming. A linear β-reduction erases a λ, an application and a variable, so the size of the term reduces by 3, as shown previously. If we organise our data accordingly, we can use this idea to erase terms. For a trivial example consider $(\lambda x.x)(\lambda y.y)$. This term reduces to $\lambda y.y$, where $\lambda x.x$ has been consumed (erased). The challenge therefore is to see if we can do this for any term t: can we find a context $C[\,]$ such that $C[t] \to^* I$, thus erasing t?
2. Erasing by pairing with garbage. We can simulate the non-linear term $K = \lambda xy.x$ by first defining the linear term $K' = \lambda xy.\langle x,y\rangle$. When we apply this to two arguments, it reduces to a pair: $K'tu \to^2 \langle t,u\rangle$. We can think of this as the result, t, together with the garbage that has yet to be erased, u. We can extend this idea so that we can accumulate all the garbage until the end of the computation, and the answer we require is the first component. This approach has previously been studied by Klop [6].

The second approach has some drawbacks. It requires that our functions return pairs. For example, if $n > 0$, then the test for zero may be like this: $\mathsf{zero}\ n = \langle F,n\rangle$, where F is the result we want, and n is the number that we need to erase. When we want to use the result of this computation, we need to extract the first component of the pair, and carry around the unwanted part, potentially adding to it with further garbage. This approach is therefore possible, but not ideal. We will use it temporarily in defining systems, but try to find ways to eliminate the extra collected garbage. We will not declare that a system has a linear definable test for zero if this is the only way it can be encoded.

The first approach requires a lot of reduction steps, but can be justified if we consider that this is doing nothing other than explicit garbage collection. Erasing by consuming is based on the following result, which is a variant of solvability. It states that a simple applicative context of identity functions will always cause a closed linear term to reduce to the identity function I. This gives an important observation that linear terms are solvable by linear contexts:

Lemma 5 (Erasing) *Let t be a closed linear term. There exists a number n such that $tI^{\sim n} \to^* I$. Moreover, there is a least such number n.*

Proof Since t is terminating, we assume without loss of generality that t is a normal form. We proceed by induction on the size of t. By Lemma 2, the closed linear term t has a normal form of the shape: $\lambda x_1 \ldots x_n.x_i t_1 \ldots t_m$. If $n = 1$ and $m = 0$ (so $t = \lambda x.x$) then there is nothing to do, otherwise we consider two cases:

1. If $n > 1$ then we can construct the term $tI^{\sim n}$. Note that $|tI^{\sim n}| = |t| + 3n$. Now there are n redexes, so this term can reduce to $It'_1 \ldots t'_m$, where t'_i is either t_i or t_i with a free variable substituted for I. By Lemma 3, Part 3, $|It'_1 \ldots t'_m| = |t|$. If we now reduce the head redex we get a term that is strictly smaller than t. We can normalise it and conclude by induction.
2. Otherwise, $n = 1$ and $m > 0$ and we can construct tI, which reduces in one step to $It_1 \ldots t_m$. If we now reduce the head redex we have a smaller term again. We can normalise it and then conclude by induction.

We give an example. Let $t = \lambda x.xIIII$, then this term can be erased by building $tI^{\sim 1}$, i.e., by using the context $C[\,] = [\,]I$. It is then easy to see that $C[t] = (\lambda x.xIIII)I \to^* I$. We will say there is a β-*collapse* for a term t if there is a context $C[\,]$ that causes the term t to reduce to the identity. Once we have the context, since the the size of the term is known, we know how many steps are needed to normalise it.

Using this idea to cause a β-collapse, we can define some data types. For example, inspired by the encoding of Booleans in the λI-calculus [1], we can define linear Booleans.

Definition 7 (Linear Booleans) For $n \geq 0$, we define $B_n = (T_n, F_n)$, where $T_n = \lambda xy.yI^{\sim n}x$ and $F_n = \lambda x.xI^{\sim n}$.

We remark that $F_n = \lambda x.xI^{\sim n} =_\eta \lambda xy.xI^{\sim n}y$. Using Lemma 5, we have:

Lemma 6 *For linear λ-terms t and u, and for some $n \geq 0$, if $tI^{\sim n} \to^* I$ and $uI^{\sim n} \to^* I$, then if $m \geq n$, then: $T_m tu \to t$ and $F_m tu \to u$.*

Thus, if we have enough Is, then we can erase the part of the conditional that is not needed in the result. We now have all we need to start defining linear numeral systems.

4.2 Numbers

We begin by presenting two numeral systems. The first one is based on Church sequences. Some of the ideas for this system are due to Böhm, and studied in detail in Rezus [9]; the main difference is that we need all the operations to be defined as linear terms. The second system is a variant of Scott numerals where we have access to the end of the list. We examine the shortcomings of each system, then conclude the section by introducing a third system which is a combination of the first two, where we are able to define all the operations.

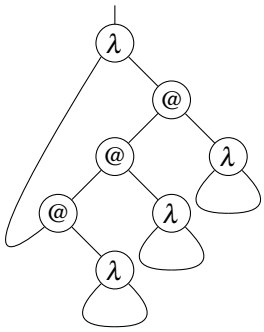
In the following we write S for successor, A for addition, P for predecessor, M for minus (Subtraction), and Z for test-for-zero. We also annotate the names of the functions with underlining and over-lining to distinguish between the first two systems, for example $\overline{S}$, $\underline{S}$, etc.

4.3 First candidate

Our first linear numeral system candidate uses linear sequences, where each element of the sequence is I:

$$\begin{array}{lll} \overline{0} & = \langle\,\rangle & = I \\ \overline{1} & = \langle I \rangle & = \lambda x.xI \\ \overline{2} & = \langle I, I \rangle & = \lambda x.xII \\ & \vdots & \\ \overline{n+1} & = \lambda x.\overline{n}xI & \end{array}$$

In general we have $\overline{n} = \langle I, \ldots, I \rangle = \lambda x.xI^{\sim n}$. We can see how this works with the following diagram, where an application and an identity function are used to represent each successor (this example represents the number $\overline{3} = \lambda x.xIII$):



It is easy to see that each numeral is represented by a linear term. We can define the following linear arithmetic operations:

Definition 8 Successor, addition, predecessor and test for zero are defined as follows:

$$\begin{aligned}
\overline{S} &= \lambda xy.xyI \\
\overline{A} &= \lambda xyz.x(yz) \\
\overline{P} &= \lambda xy.x(\lambda x.xy) \\
\overline{Z} &= \lambda x.xF_2(\lambda x.xI)T_2
\end{aligned}$$

There are some minor variants possible for some of these operations, for instance: $\overline{S'} = \lambda xy.x(yI)$ and $\overline{A'} = \lambda xyz.y(xz)$. However, they have no impact whatsoever on the efficiency of the system (the same number of reductions are needed in each case). The key observation is that a number is represented by a chain of applications with access to the first and last element, which is why there are two variants for successor: adding to the left or the right hand-side of this chain of applications, in a similar way as shown for Church numerals in Section 3.2.

Proposition 1 *The arithmetic functions compute the expected results:*

$$\begin{aligned}
\overline{S}\,\overline{n} &\to^* \overline{n+1} \\
\overline{A}\,\overline{m}\,\overline{n} &\to^* \overline{m+n} \\
\overline{P}\,\overline{n+1} &\to^* \overline{n} \\
\overline{Z}\,\overline{0} &\to^* T_2 \\
\overline{Z}\,\overline{n+1} &\to^* F_3
\end{aligned}$$

Proof We check three of these.

$$\begin{aligned}
S\,\overline{n} &= (\lambda xy.xyI)(\lambda x.xI^{\sim n}) \to \lambda y.(\lambda x.xI^{\sim n})yI \to \lambda y.yI^{\sim n}I = \lambda y.yI^{\sim n+1} = \overline{n+1} \\
A\,\overline{m}\,\overline{n} &= (\lambda xyz.x(yz))\,\overline{m}\,\overline{n} \to^* \overline{m+n} \qquad \text{by Lemma 4} \\
P\,\overline{n+1} &= (\lambda xy.x(\lambda x.xy))(\lambda x.xI^{\sim n+1}) \to \lambda y.(\lambda x.xI^{\sim n+1})(\lambda x.xy) \\
&\to \lambda y.(\lambda x.xy)I^{\sim n+1} \to \lambda y.(\lambda x.xy)II^{\sim n} \to \lambda y.IyI^{\sim n} \to \lambda y.yI^{\sim n} = \overline{n}
\end{aligned}$$

□

One nice aspect of this system is the test for zero: any number will collapse to the identity function when applied to just one I: $\overline{n}I \to^* I$, so the erasing of the number is easily simulated.

Corollary 1 *1. $\overline{S}$, $\overline{A}$ and $\overline{P}$ are all constant time operations.*
2. $\overline{Z}$ is not a constant time operation (there exists no constant time $\overline{Z}$ since the number must be erased).

Although this system meets many of our requirements, it fails on one aspect: subtraction. The situation is not recoverable because of Lemma 5:

Proposition 2 *There is no linear term $\overline{M}$ such that $\overline{M}\,\overline{m}\,\overline{n} \to^* \overline{m-n}$.*

Proof Assume that there is a term $\overline{M}$, then by Lemma 2 this term must have the normal form:

$$\lambda x_1 \dots x_i.x_j t_1 \dots t_k$$

where $i \geq 1$, $k \geq 0$, $1 \leq j \leq i$, and t_1, …, t_k, are all normal forms. Since the result must be of the form $\overline{m-n}$, we also know that $i \leq 3$. There are therefore three cases to consider:

- If $i = 1$, then $\overline{M} = \lambda x.xt_1 \dots t_k$, and by linearity each term $t_1, \dots, t_k$ must be closed. Now $\overline{M}\,\overline{m}\,\overline{n} \to^* \overline{m}t_1 \dots t_k\overline{n}$, and my Lemma 5, $\overline{m}t_1 \to^* I$, so $\overline{m}$ is lost and thus M where $i = 1$ cannot compute subtraction.
- If $i = 2$, then there are two cases: $j = 1$ or $j = 2$. If $j = 1$, then $\overline{M}$ is the term $\lambda x_1 x_2.x_1 t_1 \dots t_l \dots t_k$, where $a \leq l \leq k$, and by linearity each term $t_1, \dots, t_l, \dots, t_k$ must be closed, with the exception of t_l that has the free variable x_2. Now $\overline{M}\,\overline{m}\,\overline{n} \to^* \overline{m}t_1 \dots t_l[\overline{n}/x_2] \dots t_k$. The term t_l, being a normal form, must be of the shape $x_2 u_1 \dots n_p$, where u_1 is closed, thus $\overline{n}$ is lost and thus $\overline{M}$ where $i = 2$ and $j = 1$ cannot compute subtraction. The case for $j = 2$ is similar.
- If $i = 3$, then there are three cases to consider ($1 \leq j \leq 3$) which are all similar to the previous case.

Interestingly, there is a linear term $\overline{M_n}$ such that $\overline{m} \circ \overline{M_n} \to^* \overline{m-n}$. This is given by $\overline{M_0} = \lambda x.x$ and $\overline{M_{n+1}} = \lambda yx.x(\overline{M_n}y)$. This has a cost in the size of n, and it is one of the very best subtractions for numeral systems. However, it is not so useful here since we cannot construct the term $\overline{M_n}$ from $\overline{n}$, as shown in the previous result, with a linear function. We will come back to this result later in the third attempt at building a linear numeral system.

Remark 1 All the linear systems that we define can also be expressed using the linear combinators C, B and I. For example, the system above can be written as: $\overline{0} = I$, $\overline{1} = CII$, $\overline{2} = C(CII)I, \dots, \overline{n+1} = C\overline{n}I$. Then each of the linear operators can be written using these combinators, for example $\overline{S} = C(BC(BI))I$.

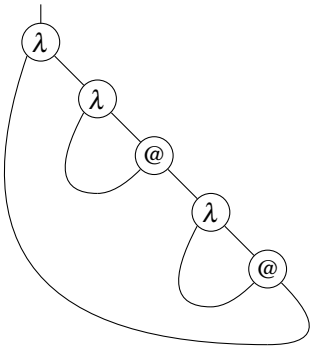
4.4 Second candidate

Our second attempt to build a linear numeral system is based on alternating abstractions and applications. This system builds terms in a similar style to Scott numerals in fact, but has the pointer to the end. This way of representing numbers was also found by Parigot and Rozière [8], although linearity was not a concern in their case, so the operations are different. In this system there is no way to erase a number however, so the test for zero will have to be simulated in a specific way using pairs as discussed earlier, in Section 4.1. We will harness this idea in the next system to allow numbers (and pairs) to be erased.

We define: $d = \langle\,\rangle, \lambda x.\langle x\rangle, \lambda x.\langle\langle x\rangle\rangle \ldots$, i.e.,

$$\begin{array}{ll} \underline{0} & = \langle\,\rangle = \lambda x.x \\ \underline{1} & = \lambda x.\langle x\rangle = \lambda yx.xy \\ \underline{2} & = \lambda x.\langle\langle x\rangle\rangle = \lambda yx.x(\lambda x.xy) \\ & \vdots \\ \underline{n+1} & = \lambda yx.x(\underline{n}y) \end{array}$$

We can see how the system is working, using a diagram. The following represents $2 = \lambda yx.x(\lambda x.xy)$:



Here we can see that each successor is represented by an abstraction, a variable and an application ($\lambda x.x-$, where $-$ is the rest of the number represented as a list. Just like in the previous linear system, we have access to the first and last element of the list, so we can add to either end of this list of successors. Each numeral is represented by a linear term, and the following operations are also linear:

Definition 9 Successor, addition, predecessor, subtraction and test for zero can be defined as follows:

$$\begin{array}{ll} \underline{S} & = \lambda nyx.x(ny) \\ \underline{A} & = \lambda mny.m(ny) \\ \underline{P} & = \lambda ny.nyI \\ \underline{M} & = \lambda mny.nI(my) \\ \underline{Z} & = \lambda x.x(\lambda x.\langle T_2, xII\rangle)(\lambda x.\langle F_2, xII\rangle) \end{array}$$

where $T_2 = \lambda xy.yIIx$ and $F_2 = \lambda xy.xIIy$ are the linear Booleans as defined previously. There is a possible variant if we take $\underline{S'} = \lambda xy.x(\lambda x.xy)$.

Proposition 3 *The arithmetic functions compute the expected results:*

$$\begin{array}{ll}
\underline{S}\,\underline{n} & \to^* \underline{n+1} \\
\underline{A}\,\underline{m}\,\underline{n} & \to^* \underline{m+n} \\
\underline{P}\,\underline{n+1} & \to^* \underline{n} \\
\underline{M}\,\underline{m}\,\underline{n} & \to^* \underline{m-n} \\
\underline{Z}\,\underline{0} & \to^* \langle T_2, I\rangle \\
\underline{Z}\,\underline{n+1} & \to^* \langle F_2, \underline{n}(\lambda x.\langle T_2, xII\rangle)II\rangle
\end{array}$$

Proof All cases shown by straightforward β-reductions.

$\underline{Z}$ in this system does not meet our requirement fully, since the result is a pair. It allows us to simulate the test for zero, and if we had enough I's, then we would be able to consume the number. It does however meet all the other requirements, and in particular, we now have subtraction. The situation is again not recoverable:

Proposition 4 *There is no linear term $\underline{Z}$ such that $\underline{Z}\,\underline{n} \to^* T_2/F_2$*

Proof Wadsworth [10] has shown that in any numeral system the test for zero has a specific form: $\underline{Z} = \lambda x_1 \ldots x_i.x_1 t_1 \ldots t_k$, where $i \geq 1$, $k \geq 0$, and the principal variable is the first bound variable. So $\underline{Z}\underline{n} \to^* \lambda x_2 \ldots x_i.\underline{n}t_1 \ldots t_k$. Since $\underline{n}$ is of the form $\lambda yx.x(\lambda x.x(\ldots xy)\ldots)$, there are $n+1$ abstractions, so if $k < n$ part of n remains untouched. Since $\underline{n}$ cannot be known in advance it will not reduce to the representation of a Boolean.

Corollary 2 *1. $\underline{S}$, $\underline{A}$ and $\underline{P}$ are all constant time operations.*
2. The test for zero is linear, but we have a pair as a result.

4.5 Third candidate

Checking carefully, we note that in the previous two system $\overline{P} = \underline{S}$, and $\overline{S} = \underline{P}$ (almost, because we need to use one of the variants given). Actually, these two systems could be part of one system that represents the positive and negative numbers. Let us arbitrarily decide that $\overline{n}$ is negative, and $\underline{n}$ is positive, then:

$$\begin{array}{r}
\vdots \\
-2 = \lambda x.xII \\
-1 = \lambda x.xI \\
0 = I \\
1 = \lambda yx.xy \\
2 = \lambda yx.x(\lambda x.xy) \\
\vdots
\end{array}$$

Thus $\overline{S}/\underline{P}$ moves us up, $\overline{P}/\underline{S}$ moves us down the integers. However neither test for zero will work for these: either $\overline{Z}$ or $\underline{Z}$ work fine for each half, but neither for both. We can define a uniform test for zero though, which returns a pair:

$$Z = \lambda x.xF_3(\lambda xy.\langle F_2, xy\rangle)(\lambda x.\langle T_2, x\rangle)(\lambda xy.\langle F_2, xy\rangle)$$

However, representing the integers is not the direction we want to go, and there are additional problems that would need to be solved (related to problems with predecessor indicated in Section 3). There is an alternative way to combine these two systems in a much more useful way though. The linchpin for the third system comes from the following property, which is a key result of this paper:

Proposition 5 *Let $m,n \geq 0$, then:*

1. *If $m = n$ then $\overline{m} \circ \underline{n} \to^* I$*
2. *If $n < m$ then $\overline{m} \circ \underline{n} \to^* \overline{m-n}$*
3. *If $m < n$ then $\overline{m} \circ \underline{n} \to^* \underline{n-m}$*

Proof 1. By induction on n. When $n = 0$, $I \circ I \to^* I$ (base case). Next assume $\overline{k} \circ \underline{k} \to^* I$, we show the case for $k+1$.

$$\begin{aligned}\overline{k+1} \circ \underline{k+1} &= \lambda z.(\lambda x.\overline{k}(xI))((\lambda yx.x(\underline{k}y))z) \\ &\to \lambda z.(\lambda x.\overline{k}(xI))(\lambda x.x(\underline{k}z)) \\ &\to \lambda z.\overline{k}((\lambda x.x(\underline{k}z))I) \\ &\to \lambda z.\overline{k}(\underline{k}z))\end{aligned}$$

Now $\overline{k} \circ \underline{k} \to^* \lambda z.\overline{k}(\underline{k}z))$, and so by the induction hypothesis, $\lambda z.\overline{k}(\underline{k}z)) \to^* I$ as required.

2. The other two cases are a consequence of the following three results which can be shown is the same way as the inductive step above.
   (a) $\overline{m+1} \circ \underline{n+1} = \overline{m} \circ \underline{n}$
   (b) $\overline{m-n} \circ \underline{0} = \overline{m-n}$
   (c) $\overline{0} \circ \underline{n-m} = \underline{n-m}$

□

This result gives a way to erase fully in the second system, but more importantly, it gives a way to compute subtraction in $min(m,n)$. We harness this idea to build a third numeral system that is a combination of the previous two. We do this by building pairs of numbers. Actually, representing integers using pairs is not a new idea. A number n is a pair (a,b) which we interpret as $a-b$. It is then possible to define operations over these pairs, for example: $(a,b) +_p (c,d) = (a+c,b+d)$ and $(a,b) -_p (c,d) = (a+d,b+c)$. Thus we can add two pairs ($+_p$) using addition on numbers, and we can also subtract pairs ($-_p$) using addition. We will make use of a variant of this idea and build the pair $(n,-n)$, where each number will be represented from a different numeral system. The operators for the first two systems together with Proposition 5 then allow all the functions to be computed. We remark however that storing the number twice means that there is some redundancy in the system. It is also worth pointing out that Wadsworth's system needed two copies of the number for zero test. The same is true for this system, but we are in a linear framework so need to keep hold of the second number, and no η-collapse is needed.

We now build a fully linear numeral system, which is main contribution of this paper. We define $d = \langle I,I\rangle$, $\langle \lambda yx.xy, \lambda x.xI\rangle$, $\langle \lambda yx.x(\lambda x.xy), \lambda x.xII\rangle$, ..., i.e., we have

a sequence:

$$\begin{array}{lll}
[\![0]\!] = \langle I, I\rangle & = \lambda x.xII \\
[\![1]\!] = \langle\langle I\rangle, \lambda x.\langle x\rangle\rangle & = \lambda x.x(\lambda x.xI)(\lambda yx.xy) \\
[\![2]\!] = \langle\langle I, I\rangle, \lambda x.\langle\langle x\rangle\rangle\rangle & = \lambda x.x(\lambda x.xII)(\lambda yx.x(\lambda x.xy)) \\
\quad\vdots & \\
[\![n]\!] = \langle \overline{n}, \underline{n}\rangle & = \lambda x.x\overline{n}\underline{n}
\end{array}$$

Each number is represented by a linear term, and we can define all the operations in terms of the operators of the two previous numeral systems. The following operations are all linear.

Definition 10 Successor, addition, predecessor, subtraction and test for zero are defined as follows.

$$\begin{array}{ll}
[\![S]\!] & = \langle \lambda ab.\langle \overline{S}a, \underline{S}b\rangle\rangle = \lambda x.x(\lambda abc.c(\overline{S}a)(\underline{S}b)) \\
[\![A]\!] & = \lambda mn.m(\lambda ab.n(\lambda cd.\langle \overline{A}ac, \underline{A}bd\rangle)) \\
[\![P]\!] & = \langle \lambda ab.\langle \overline{P}a, \underline{P}b\rangle\rangle = \lambda x.x(\lambda abc.c(\overline{P}a)(\underline{P}b)) \\
[\![M]\!] & = \lambda mn.m(\lambda ab.n(\lambda cd.\langle \overline{A}da, \underline{A}bc\rangle)) \\
[\![Z]\!] & = \lambda n.n(\lambda ab.(\underline{Z}\, b)(\lambda cd.adc))
\end{array}$$

Note that we use Z from the second system here, and we now have enough I's to fully erase the terms from the first system.

Proposition 6 *The arithmetic functions compute results as required:*

$$\begin{array}{ll}
[\![S]\!][\![n]\!] & \to^* [\![n+1]\!] \\
[\![A]\!][\![m]\!][\![n]\!] & \to^* [\![m+n]\!] \\
[\![P]\!][\![n+1]\!] & \to^* [\![n]\!] \\
[\![M]\!][\![m]\!][\![n]\!] & \to^* [\![m-n]\!] \\
[\![Z]\!][\![0]\!] & \to^* T_2 \\
[\![Z]\!][\![n+1]\!] & \to^* F_2
\end{array}$$

Proof We show a selection of cases.

$$\begin{array}{rl}
[\![S]\!][\![n]\!] = [\![S]\!]\langle \overline{n}, \underline{n}\rangle & = (\lambda x.x\overline{n}\underline{n})(\lambda abc.c(\overline{S}a)(\underline{S}b)) \\
& \to (\lambda abc.c(\overline{S}a)(\underline{S}b))\overline{n}\underline{n} \to \lambda c.c(\overline{S}\,\overline{n})(\underline{S}\,\underline{n}) \to \langle \overline{n+1}, \underline{n+1}\rangle = [\![n+1]\!]
\end{array}$$

$$\begin{array}{rl}
[\![P]\!][\![n]\!] = [\![P]\!]\langle \overline{n}, \underline{n}\rangle & = (\lambda x.x\overline{n}\underline{n})(\lambda abc.c(\overline{P}a)(\underline{P}b)) \\
& \to (\lambda abc.c(\overline{P}a)(\underline{P}b))\overline{n}\underline{n} \\
& \to \lambda c.c(\overline{P}\,\overline{n})(\underline{P}\,\underline{n}) \to \langle \overline{n-1}, \underline{n-1}\rangle = [\![n-1]\!]
\end{array}$$

$$\begin{array}{rl}
[\![Z]\!][\![0]\!] & = (\lambda n.n(\lambda ab.(\underline{Z}\, b)(\lambda cd.adc)))(\lambda z.zII) \to (\lambda z.zII)(\lambda ab.(\underline{Z}\, b)(\lambda cd.adc)) \\
& \to (\lambda ab.(\underline{Z}\, b)(\lambda cd.adc))II \to^2 \underline{Z}\, I(\lambda cd.Idc) \to^2 \langle T_2, I\rangle(\lambda cd.dc) \\
& \to (\lambda cd.dc)T_2I \to IT_2 \to^2 T_2
\end{array}$$

$$\begin{array}{rl}
[\![Z]\!][\![n+1]\!] & = (\lambda n.n(\lambda ab.(\underline{Z}\, b)(\lambda cd.adc)))(\lambda z.z(\overline{n+1})(\underline{n+1})) \\
& \to (\lambda z.z(\overline{n+1})(\underline{n+1}))(\lambda ab.(\underline{Z}\, b)(\lambda cd.adc)) \\
& \to (\lambda ab.(\underline{Z}\, b)(\lambda cd.adc))(\overline{n+1})(\underline{n+1}) \to (\underline{Z}\,(\underline{n+1}))(\lambda cd.(\overline{n+1})dc) \\
& \to \langle F_2, \underline{n}(\lambda x.\langle T, xII\rangle)II\rangle(\lambda cd.(\overline{n+1})dc) \to (\overline{n+1})\underline{n}(\lambda x.\langle T, xII\rangle)IIF_2 \to^* F_2
\end{array}$$

□

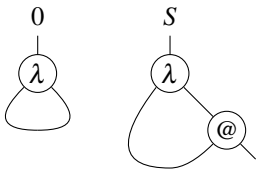
Corollary 3 – *Successor, addition and predecessor are all constant time operations.*
– *Test for zero is a linear time operation (it requires that we erase fully a number to give a Boolean result).*
– *The subtraction function requires min (m,n) reductions to compute $[\![m-n]\!]$.*

5 Characterising linear numeral systems

There are many other candidates for a linear numeral system. One of the easiest is given by:

$$\begin{aligned} 0 &= \langle\,\rangle = I \\ 1 &= \langle I\rangle = \lambda x.xI \\ 2 &= \langle\langle I\rangle\rangle = \lambda x.x(\lambda x.xI) \\ &\vdots \\ n+1 &= \lambda x.xn \end{aligned}$$

In general we have $n = \langle\cdots\langle I\rangle\ldots\rangle$. Each numeral is represented by a linear term, and this corresponds directly to the inductive definition of numbers given earlier in the paper. Here is a diagram showing the building blocks 0 and S:



Thus, we have the same successor and zero as the second candidate, but constructed differently. In this representation, the following operations are also linear:

Definition 11 We define successor, predecessor, subtraction and test for zero as follows:

$$\begin{aligned} S &= \lambda nx.xn \\ P &= \lambda n.nI \\ M &= \lambda mn.nm \\ Z &= \lambda x.x(\lambda x.\langle T_2,x\rangle)(\lambda xy.\langle F_2,xyI\rangle) \end{aligned}$$

We remark that successor, predecessor and subtraction are beautifully simple, and we can encode the test for zero with pairs as done before. However, this time there is no addition.

Proposition 7 *There is no linear term A such that $A\ m\ n \to^* m+n$.*

The reason for this is the same as why there is no constant time addition for the algebraic data type for numbers given earlier, or similarly for the concatenation of linked lists: in all cases, we need to traverse the first number (list) to join it to the second one. Of course, not being able to do addition is really a failure of this system, but it can be defined outside the linear framework (by using an encoding of recursion, as for Scott numerals for instance).

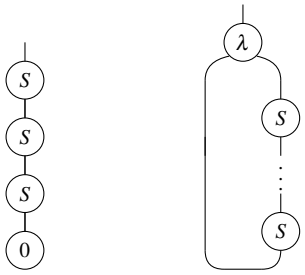


Fig. 3 Alternative number systems

We have now collected a number of "failed" linear numeral systems, and we can start to ask which systems therefore do have addition, which have predecessor, which have subtraction, etc., and then we can try to understand when this arises.

We begin by observing that in a closed linear λ-term:

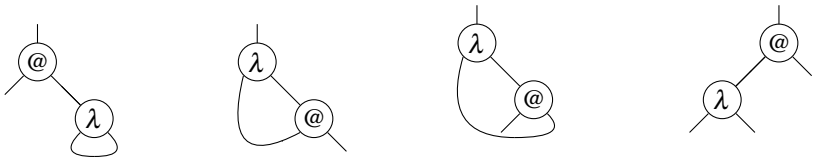
- each λ binds one occurrence of a variable;
- each application has disjoint free variables;
- each variable is bound by a λ.

Therefore, each "successor" that we build in a linear system must be built from equal numbers of variables, abstractions and applications, and at least one of each is needed each time. For example, if we want to add an application, then we need to introduce a new variable, and this can only be added if we add an abstraction to bind this variable. We cannot create a numeral system which is not balanced in this way. Thus, we immediately find a characterisation of the size of linear terms as seen in Lemma 3, Part 1. As a consequence we can understand the size of linear numerals:

- Zero is represented as a closed linear term of size c, where $c = 2+3k$ for some k. $k = 0$ in all the systems we have presented in this paper so far, because the representation of zero is just $\lambda x.x$, which has size 2.
- Each successor causes the size to increase by $3k'$, where $k' \geq 1$ is the number of abstractions (or variables or applications) used. The smallest value of k' is 1, which we have used several times already for the first, second and fourth candidates at a linear numeral system. For the third system, we used $k' = 2$.
- The size of a numeral n is therefore given by: $2+3k+3nk'$, for some $k \geq 0, k' \geq 1$.

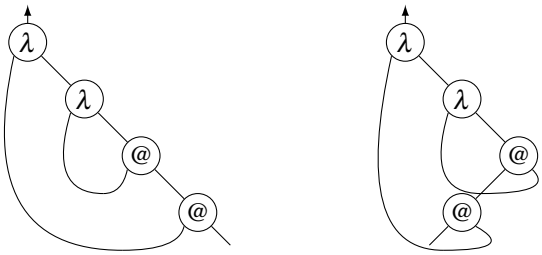
Consider the smallest systems, where $k = 0$ and $k' = 1$ (these correspond to the values for the first, second and fourth systems presented above). We now ask what are all the possible linear systems that can be built in this way. We have found that diagrams are useful for this purpose, and we will demonstrate the ideas through the diagrams. First, we remark that there are essentially two different structures we have used, depending on whether we have access to the end of the list or not. These are represented by the two diagrams in Figure 3, where 0 and S represent the building blocks, used previously, defined using linear λ-terms. The left-hand side corresponds to the standard inductive definition of numbers, so the fourth system, and the right-hand side corresponds to systems 2 and 3, where we have access to the end of the

list. All systems that use the left-hand approach will not have a linearly definable addition, whereas those on the right will. Choices of S then give different systems that we can define. Since we are assuming that $k' = 1$, each successor will be built out of an abstraction, application and a variable. We next consider all the ways that these can be put together. There are essentially 4 ways to do this. The first two shown below are used in the first and second system respectively:



For the first one, there is no choice: the abstraction here must be the identity function, as there are no other variables to bind. The second one however has some more choices that we will mention later. The next two ways in which we can combine an abstraction and application are less useful as we cannot allow terms to be η- or β-redexes.

There are however additional choices. In the right-most two configurations above where there are two edges connecting the abstraction and application nodes we can split one of them to make alternative constructions. There is not much choice: we cannot have a list of abstractions as there is no way to erase. We may try to build systems like this for example:



Some of the above are useful, if we pair them with the first system - this can erase the abstraction, but none of these can lead to a linear numeral system. We can exhaust all possibilities, and establish that there is no linear numeral system when $k = 0$, $k' = 1$, thus the first is with characteristic values $k = 0$, $k' = 2$.

To summarise, the following table shows the four candidates for being a linear numeral system, and we indicate which operations are linearly definable (all are λ-definable if we remove the linear constraint). ✓ indicates that the operation was linearly definable, $\times$ indicates that the operation was not linearly definable, and we write $\frac{1}{2}$ if we needed to use pairs to encode erasing (thus we consider this a partial solution).

	S	P	Z	A	M
1	✓	✓	✓	✓	$\times$
2	✓	✓	$\frac{1}{2}$	✓	✓
3	✓	✓	✓	✓	✓
4	✓	✓	$\frac{1}{2}$	$\times$	✓

Systems 1, 2 and 4 are all characterised by $k = 0$, $k' = 1$, but system 3 is characterised by $k = 0$, $k' = 2$.

The following proposition summarises the points above.

Proposition 8 *For any linear numeral system:*

- *The size of the term representing the number n is proportional to n. Specifically, n would be represented by a term of size $nk + c$.*
- *Successor, addition and predecessor will be constant time operations.*
- *Test for zero and subtraction will be linear time operations.*

Proof
- Let c be the size of the representation of zero. Each successor must add the same size term, say k, which gives the result immediately.
- The successor S of a number n will be built from an application Sn, where $|Sn| = |S| + |n| + 1$. Since each β-reduction reduces the size of a term by 3, we know that the extra structure needed to represent $n+1$ must come from the term S. Thus not only must this be a constant number of β-reductions, but we know the size of the term S also.
  The same reasoning applies to the addition and predecessor operations.
- Erasing a term can never be a constant time operation and therefore must be linear. Subtraction and test for zero both need to erase part of a number of a whole number.

Because of this characterisation, there cannot be a more compact representation of numbers in the linear λ-calculus. In particular, we cannot represent a linear version of binary numerals (see for example [7]) for instance.

6 Conclusion

One of the main results in the λ-calculus is that numerals can be defined, together with successor, predecessor and test for zero functions, and the system is adequate to represent all computable functions. The numeral system presented in this paper can be understood as numeral systems in the usual λ-calculus, and replacing the linear Booleans by the standard ones, immediately eliminates the inefficiency issues associated with tests for zero. In this way, all the systems defined here are adequate. Building numeral systems in a constrained calculus allowed us to get insight into numeral systems generally. Interestingly the linearity constraints imply that we are limited in choices but also ensure that the operations are efficient: it is not possible to define a non-constant time addition for instance.

We have shown that the linear λ-calculus can be used to represent numbers in an efficient way. Specifically, successor, predecessor and addition are all constant time functions, and moreover they *have to be* constant time. Subtraction, frequently omitted in numeral systems, and especially cumbersome in Church style representations, can also be done efficiently. The graphical representation of linear terms has been useful to identify new representations.

One aspect of this work, and indeed one of the reasons for investigating linear numeral systems in the first place, was to find a more efficient representation of predecessor and subtraction operations for System F [4] by potentially finding new ways to represent the data. All linear terms are typeable, so all numbers, and all operations we have presented in this paper also have a type. However, each number has a different type—recursive types are therefore needed—so this work does not offer any improvement for System F. We leave this aspect however for further investigation.

References

1. Barendregt, H.P.: The Lambda Calculus: Its Syntax and Semantics, *Studies in Logic and the Foundations of Mathematics*, vol. 103, second, revised edn. North-Holland Publishing Company (1984)
2. Church, A.: The Calculi of lambda-conversion. No. 6 in Annals of Mathematics Studies. Princeton University Press (1941)
3. Curry, H.B., Feys, R.: Combinatory Logic, Volume I. North-Holland (1958). Second printing 1968
4. Girard, J.Y., Lafont, Y., Taylor, P.: Proofs and Types, *Cambridge Tracts in Theoretical Computer Science*, vol. 7. Cambridge University Press (1989)
5. Hankin, C.: An Introduction to Lambda Calculi for Computer Scientists, *Texts in Computing*, vol. 2. King's College Publications (2004)
6. Klop, J.W.: Combinatory Reduction Systems, *Mathematical Centre Tracts*, vol. 127. Mathematischen Centrum, 413 Kruislaan, Amsterdam (1980)
7. Mogensen, T.Æ.: An investigation of compact and efficient number representations in the pure lambda calculus. In: D. Bjørner, M. Broy, A.V. Zamulin (eds.) Perspectives of System Informatics, 4th International Andrei Ershov Memorial Conference, PSI 2001, Akademgorodok, Novosibirsk, Russia, July 2-6, 2001, Revised Papers, *Lecture Notes in Computer Science*, vol. 2244, pp. 205–213. Springer (2001)
8. Parigot, M., Rozière, P.: Constant time reductions in lambda-calculus. In: A.M. Borzyszkowski, S. Sokolowski (eds.) Mathematical Foundations of Computer Science 1993, 18th International Symposium, MFCS'93, Gdansk, Poland, August 30 - September 3, 1993, Proceedings, *Lecture Notes in Computer Science*, vol. 711, pp. 608–617. Springer (1993)
9. Rezus, A.: Lambda-conversion and logic. Ph.D. thesis, University of Utrecht (1981)
10. Wadsworth, C.P.: Some unusual λ-calculus numeral systems. In: J.P. Seldin, J.R. Hindley (eds.) To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, pp. 215–230. Academic Press, London (1980)