Under consideration for publication in J. Functional Programming

A Characterization of Lambda-terms Transforming Numerals

PAWEŁ PARYS∗
University of Warsaw, Warsaw, Poland
(*e-mail:* parys@mimuw.edu.pl)

Abstract

It is well known that simply-typed λ-terms can be used to represent numbers, as well as some other data types. We show that λ-terms of each fixed (but possibly very complicated) type can be described by a finite piece of information (a set of appropriately defined intersection types) and by a vector of natural numbers. On the one hand the description is compositional: having only the finite piece of information for two closed λ-terms M and N, we can determine its counterpart for MN, and a linear transformation that applied to the vectors of numbers for M and N gives us the vector for MN. On the other hand, when a λ-term represents a natural number, then this number is approximated by a number in the vector corresponding to this λ-term.

As a consequence, we prove that in a λ-term of a fixed type we can store only a fixed number of natural numbers, in such a way that they can be extracted using λ-terms. More precisely, while representing k numbers in a closed λ-term of some type we only require that there are k closed λ-terms $M_1,\dots,M_k$ such that M_i takes as argument the λ-term representing the k-tuple, and returns the i-th number in the tuple (we do not require that, using λ-calculus, one can construct the representation of the k-tuple out of the k numbers in the tuple). Moreover, the same result holds when we allow that the numbers can be extracted approximately, up to some error (even when we only want to know whether a set is bounded or not).

All the results remain true when we allow the Y combinator (recursion) in our λ-terms, as well as uninterpreted constants.

1 Introduction

It is well known that simply-typed λ-terms can be used to represent natural numbers, as well as some other data types (for an introduction see e.g. (Barendregt *et al.*, 2013)). Then, we can construct higher-order functions operating on numerals (that is, representations of natural numbers). The goal of this paper is to characterize all such functions.

Consider for example functions of the form

$$g(f) = n_1 + f(n_2 + f(n_3 + f(\cdots + f(n_k)\dots))),$$

where $n_1,\dots,n_k$ are some fixed natural numbers. In order to know precisely the result of such a function g for each argument f, we need to remember all the numbers $n_1,\dots,n_k$ (notice that k may be arbitrarily big). If, however, we allow approximation of the result up

∗ Work supported by the National Science Center (decision DEC-2012/07/D/ST6/02443).

to some error, the situation changes dramatically. Assuming that all n_i are positive, it turns out that the function g is quite well approximated by the function

$$g'(f) = n_1 + f(m),$$

where $m = n_2 + ... + n_k$. For example, if $f(x) = 2 \cdot x$, then $g'(f) \leq g(f) \leq g'(f) \cdot 2^{g'(f)}$. In fact, for each fixed function f definable in simply-typed λ-calculus we can give a similar relationship between $g'(f)$ and $g(f)$ (not depending on the values used in g and g'). Notice that we need to remember n_1 separately in order to handle functions f ignoring its argument, like $f(x) = 0$.

We thus see that all the functions g are "of the same shape" and can be specified using only two numbers: n_1 and m. In this paper we generalize the above example, and we prove that for each sort[1] there exist only finitely many "shapes" of functions, and for each of them we need to specify only a fixed number of constants appearing in the function.

More precisely, to each closed λ-term M of a fixed sort α we assign a finite piece of information $types(M)$, which is a set of appropriately defined types of M (for each α there are only finitely possible values of $types(M)$). We also assign to M a vector of natural numbers, whose length is equal to the size of $types(M)$; thus the maximal length of this vector for each α is fixed. If a λ-term M represents a natural number n (which is possible only for terms of one sort α), then some number m in the vector for M approximates n, that is it holds that $m \leq H(n)$ and $n \leq H(m)$, where H is a fixed, although very fast growing, function. For λ-terms M of other sorts α, the numbers in the vector assigned to M may be seen as approximations of the numbers "appearing in M".

The key reason why such a characterization may be at all interesting is compositionality. Consider an application MN, where M and N are closed λ-terms. Then $types(MN)$ can be determined while knowing only $types(M)$ and $types(N)$. Moreover, there exists a linear transformation L, again depending only on $types(M)$ and $types(N)$, such that L applied to the vectors assigned to M and to N gives us the vector assigned to MN. In fact L will be of a very special form: to obtain a number in the vector for MN, we take one number from the vector for M and some numbers from the vector for N, and we sum them.

We believe that the characterization of higher-order functions is interesting in its own, but we also obtain some consequences. Recall that in λ-calculus we can represent pairs or tuples of representable data types, in particular of natural numbers. Notice however that the sort of terms representing pairs is more complex than the sort of terms representing the elements of pairs. It follows from our result that, indeed, for representing k-tuples of natural numbers for big k, we need terms of complex sort. For this reason, for each sort α we define a number $dim(\alpha)$, the *dimension* of sort α, and we prove that tuples of more than $dim(\alpha)$ natural numbers cannot be represented in terms of sort α. This has to be understood correctly. It is not a problem to pack arbitrarily many natural numbers into a term of some fixed (even very simple) sort, so that for each list (arbitrarily long tuple) of natural numbers we obtain a different term, like in our function g given above. We however consider the opposite direction, that is extracting numbers from terms: we do not require anything about

[1] We use the name "sort" instead of "type" (except in the abstract) to avoid confusion with the types introduced later, used for describing λ-terms more precisely.

how a representation of a tuple can be created out of the numbers in the tuple, but what we require is that using λ-terms we can extract the numbers from the representation of the tuple. One could imagine the following function h:

$$h(x) = \begin{cases} n_1 & \text{if } x = 1, \\ \dots & \\ n_k & \text{if } x = k, \\ 0 & \text{otherwise.} \end{cases}$$

By passing different values of x, we could extract out of h each n_i separately. But we see that, unlike g, this function cannot be rewritten using a fixed amount of numbers, since each n_i is independent. In the light of our results, this means that h cannot be realized in simply-typed λ-calculus.

We also obtain another property. Let $\alpha_{\mathbb{N}}$ be the sort of terms representing natural numbers. Fix a sort α, and consider the following equivalence relation over closed terms of sort $\alpha \to \alpha_{\mathbb{N}}$: we have $M \sim M'$ when for each sequence $N_1, N_2, \dots$ of closed terms of sort α, the sequences of numbers represented by the terms $MN_1, MN_2, \dots$ and $M'N_1, M'N_2, \dots$ are either both bounded or both unbounded (if all these terms indeed represent numbers). We obtain that this relation has at most $dim(\alpha)$ (hence finitely many) equivalence classes.

We should note that our environment is slightly stronger than just higher-order functions on natural numbers using some basic arithmetical operations. It is possible to have terms not representing any number nor a function on natural numbers, but by composing such terms we can later obtain a term representing a number.

Our theorems are independent from the choice of a particular numeral system (a way of representing natural numbers in λ-terms). The reason is that numbers read in one numeral system approximate numbers read in any other numeral system (since there can be only finitely many numerals of at most a particular size, small numerals represent small numbers), and we talk only about asymptotic behavior. A particular numeral system is one in which the number represented by a λ-term equals the size of its β-normal form. Thus the results can be restated using sizes of β-normal forms of terms, without talking about numeral systems. In fact, the core of our proofs refers basically to sizes of β-normal forms, not directly to numeral systems.

Our results hold also

- when we consider the λY-calculus, that is when we allow the Y combinator, introducing infinite recursion, and
- when one is allowed to use uninterpreted constants in λ-terms.

We remark that results of our paper hold only for simply-typed λ-calculus. As we will see on page 32, analogous results become false for polymorphic λ-calculus (already with the Hindley-Milner type system).

Related Work. This is an extended version of a conference paper (Parys, 2014). In the conference paper the main result was slightly weaker: to a term M we were assigning not only a finite piece of information $types(M)$ and a vector of natural numbers, but it was also necessary to know the maximal order k of a subterm appearing in M. For higher k the precision of approximation of the number represented in a term by numbers in the vector

was smaller. Here, the precision is independent of k. Additionally, we have allowed here the Y combinator and arbitrary representations of natural numbers. The two main differences in the proof are the following. First, the type system introduced in (Parys, 2014) was using a productive/nonproductive flag, that is replaced here by a number called duplication order. This change was necessary when we wanted to obtain precision independent of the maximal order of a subterm. Second, in (Parys, 2014) we were using Krivine machines, while here we directly analyze a sequence of left-most reductions. This change is mainly on the level of presentation, and the idea of the proof remains the same (indeed, a Krivine machine performs left-most reductions).

Results in the spirit of this paper were an important part of the proof (Parys, 2012) that collapsible higher-order pushdown systems generate more trees than higher-order pushdown systems without the collapse operation. In that proof, it is first shown that if some particular tree is generated by a pushdown system without collapse, then the stack of the pushdown system can be divided into two parts, one encoding a function, and the other encoding an argument to the function, so that the function has to operate on its argument in some complex way. Then, like in our paper, it is shown that the behavior of every of the two parts can be represented by a type coming from a finite set and by a vector of numbers that are composed in a linear way; such a representation is not strong enough to encode the complex behavior of the function claimed in the former part of the proof, which leads to a contradiction. The appropriate lemmas of (Parys, 2012) were almost completely hidden in the appendix, and stated in the world of stacks of higher-order pushdown systems. Because we think that these results are of independent interest, we present them here, in a more natural variant.

The types defined in our paper resemble the intersection types used in (Kobayashi, 2013). However, comparing to (Kobayashi, 2013), in our types we additionally have a number, called the duplication order.

Schwichtenberg (Schwichtenberg, 1976) and Statman (Statman, 1979) show that the functions over natural numbers representable in the simply-typed λ-calculus are precisely the "extended polynomials". Notice that they describe only first-order functions (functions $\mathbb{N}^k \to \mathbb{N}$), while we analyze arbitrary higher-order functions; on the other hand we are not interested in the precise rate of growth of the functions. Similarly, Zaionc (Zaionc, 1987) characterizes the class of functions over words which are represented by closed λ-terms (for appropriate representation of words in λ-calculus).

We explore here equality of functions up to boundedness: two functions leading into natural numbers are considered equivalent if over each subset of their domain they are either both bounded or both unbounded. Such an equivalence relation, called domination equivalence, is widely used in the context of regular cost functions, see e.g. (Colcombet, 2013).

Structure of the Paper. In Section 2 we define some basic notions. In Section 3 we introduce a type system which has two roles. First, it allows us to determine which arguments of a term will be used (i.e. will not be ignored, as in $\lambda x.\lambda y.x$). Second, the type of a subterm says whether this subterm is productive, that is whether it adds something to the value of the whole term. Then we define the vector of numbers assigned to each term. Finally, we state our main theorem (Theorem 10) and we prove some basic properties. The

major part of the proof of Theorem 10 is contained in Section 4. Section 5 gives a longer example illustrating notions from former sections. Finally, in Section 6 we concentrate on consequences of our result.

Acknowledgement. We thank Igor Walukiewicz and Sylvain Salvati for a discussion on this topic. We also thank anonymous reviewers of this paper as well as of its conference version for useful comments.

2 Preliminaries

The set of *sorts* is constructed from a unique basic sort o using a binary operation $\to$. Thus o is a sort and if α, β are sorts, so is $(\alpha \to \beta)$. The order of a sort is defined by: $ord(o) = 0$, and $ord(\alpha \to \beta) = \max(1 + ord(\alpha), ord(\beta))$.

The set of *simply-typed λY-terms* is defined inductively as follows. For each sort α there is a countable set of variables $x^\alpha, y^\alpha, \ldots$ and a countable set of uninterpreted constants $\mathbf{c}^\alpha, \mathbf{d}^\alpha, \ldots$ that are also terms of sort α. If M is a term of sort β and x^α a variable of sort α then $\lambda x^\alpha.M$ is a term of sort $\alpha \to \beta$. If M is of sort $\alpha \to \beta$ and N is of sort α then MN is a term of sort β. Finally, if a term M is of sort $\alpha \to \alpha$, then YM is a term of sort α.

As usual, we identify λ-terms up to α-conversion. We often omit the sort annotation of variables, but please keep in mind that every variable is implicitly sorted. A term is called *closed* when it does not have free variables. For a term M of sort α we write $ord(M)$ for $ord(\alpha)$. We write $M[N/x]$ for the term obtained from M by substituting N for all appearances of x. The *size* of a sort and of a term is defined as:

$$
\begin{aligned}
&|o| = 0, && |x^\alpha| = |\mathbf{c}^\alpha| = |\alpha|,\\
&|\alpha \to \beta| = |\alpha| + |\beta|, && |\lambda x^\alpha.M| = |\alpha| + |M|,\\
& && |YM| = 1 + |M|,\\
& && |MN| = |M| + |N|.
\end{aligned}
$$

In addition to the standard β-reduction $\to_\beta$, we have *δ-reduction* $\to_\delta$, defined by the following rewriting rule:

$$YM \to_\delta M(YM).$$

A term *is in $\beta\delta$-normal form*, if neither $\to_\beta$ nor $\to_\delta$ can be applied to it; *N is the $\beta\delta$-normal form of M* if M can be reduced to N and N is in $\beta\delta$-normal form. A term M *$\beta\delta$-normalizes* if there exists a term N that is the $\beta\delta$-normal form of M. Notice that not every term $\beta\delta$-normalizes. Recall, however, that the system has the Church-Rosser property, which means that there exists at most one $\beta\delta$-normal form of each term. Even when the $\beta\delta$-normal form exists, not every sequence of reductions leads to it (there may exist infinite sequences of reductions). It is, however, always a good strategy to perform leftmost (outermost) reductions first, as described below. A reduction $\lambda x_1.\cdots.\lambda x_m.MN_1 \ldots N_k \to_{\beta\delta} \lambda x_1.\cdots.\lambda x_m.M'N'_1 \ldots N'_k$ is called a *leftmost reduction* if

- $M = (\lambda x.K)N_0 \to_\beta K[N_0/x] = M'$ and $N_i = N'_i$ for all i, or
- $M = YK \to_\delta K(YK) = M'$ and $N_i = N'_i$ for all i, or

- $M = M'$ is a constant or a variable, and $N_i \to_{\beta\delta} N_i'$ is a leftmost reduction for some i, and $N_i = N_i'$ for all other i.

Fact 1
A λY-term $\beta\delta$-normalizes if a term in $\beta\delta$-normal form can be reached by a finite sequence of leftmost reductions. $\square$

Domination Relation. Given two numbers $n, m \in \mathbb{N}$ and a function $H\colon \mathbb{N} \to \mathbb{N}$ (called a *correction function*) we write $n \preceq_H m$ if $n \leq H(m)$. This relation is extended to functions $f, g\colon X \to \mathbb{N}$: we have $f \preceq_H g$ if $f(x) \preceq_H g(x)$ for every $x \in X$. We also write $f \approx_H g$ if $f \preceq_H g$ and $g \preceq_H f$. Finally, we say that f is *dominated* by g, denoted $f \preceq g$, if $f \preceq_H g$ for some correction function H, and we say that f is *domination-equivalent* to g, denoted $f \approx g$, if $f \approx_H g$ for some correction function H.

It is easy to observe that $f \preceq g$ holds if and only if for every set $Y \subseteq X$, $f(Y)$ is bounded whenever $g(Y)$ is bounded. Moreover $f \preceq g$ and $g \preceq f$ implies $f \approx g$, as well as $f \approx g$ and $g \approx h$ implies $f \approx h$.

Numeral Systems. A *numeral system* is given by a sort $\alpha_{\mathbb{N}}$, by a set $\mathcal{N}$ of closed λY-terms that are of sort $\alpha_{\mathbb{N}}$ and in $\beta\delta$-normal form, and by a function $val\colon \mathcal{N} \to \mathbb{N}$ that is domination-equivalent to the function $size_{\mathcal{N}}\colon \mathcal{N} \to \mathbb{N}$ returning the size of its argument, $size_{\mathcal{N}}(M) = |M|$. The function val is extended to all closed terms M whose $\beta\delta$-normal form N is in $\mathcal{N}$ by $\mathbf{val}(M) = val(N)$; we say that M *represents* the number $\mathbf{val}(M)$. Notice that not every term of sort $\alpha_{\mathbb{N}}$ represents some number: this is certainly the case for terms which do not $\beta\delta$-normalize, but possibly also for other terms.

The assumption that val is domination-equivalent to the size function may look artificial, but in fact every reasonable function aiming to define a numeral system satisfies this condition. The following proposition says that this is the case when every number is represented by only one term in $\beta\delta$-normal form, and these terms use only finitely many constants.

Proposition 2
Let $\mathcal{N}$ be a set of closed λY-terms that are of some sort $\alpha_{\mathbb{N}}$, in $\beta\delta$-normal form, and altogether use only finitely many constants. Then every injective function $val\colon \mathcal{N} \to \mathbb{N}$ defines a numeral system.

Proof
We need to prove that $val \approx size_{\mathcal{N}}$, which means that val and $size_{\mathcal{N}}$ are bounded on the same sets $X \subseteq \mathcal{N}$. Of course on finite sets both functions are bounded. Since val is injective, on every infinite set $X \subseteq \mathcal{N}$ it is unbounded. On the other hand, there are only finitely many closed terms of at most a particular size built using constants from a finite set (recall that terms differing only in names of bound variables are considered to be equal). Thus $size_{\mathcal{N}}$, similarly to val, is unbounded on every infinite set $X \subseteq \mathcal{N}$. $\square$

Another possibility of defining a numeral system is to use a term for zero, and a term for the increment operation, as described by the next proposition.

Proposition 3
Let Z and I be closed terms of sort $\alpha_{\mathbb{N}}$ and $\alpha_{\mathbb{N}}{\to}\alpha_{\mathbb{N}}$, respectively, for some sort $\alpha_{\mathbb{N}}$. Suppose that $\underbrace{I\,(I\,(\ldots\,(I}_{n}\,Z)\ldots))$ $\beta\delta$-normalizes for every $n \in \mathbb{N}$, and let M_n be its $\beta\delta$-normal form. If every M_n is different, the function *val* mapping M_n to n defines a numeral system.

Proof
The terms $M_0, M_1, \ldots$ can only use constants appearing in Z and I, so the thesis follows from the previous proposition. □

Example 1
The standard numeral system (called Church encoding) uses sort $\alpha_{\mathbb{N}} = (o{\to}o){\to}o{\to}o$ for numerals. A number $n \in \mathbb{N}$ is represented by

$$\mathcal{N}_1 \ni \lambda f.\lambda x.\underbrace{f\,(f\,(\ldots\,(f}_{n}\,x)\ldots)) \xmapsto{val_1} n.$$

Example 2
Another possibility is to use sort $\alpha_{\mathbb{N}} = o$ and constants $\mathbf{0}$ of type o, denoting zero, and $\mathbf{1}{+}$ of type $o{\to}o$, denoting an increment. Then a number $n \in \mathbb{N}$ is represented by

$$\mathcal{N}_2 \ni \underbrace{\mathbf{1}{+}\,(\mathbf{1}{+}\,(\ldots\,(\mathbf{1}{+}}_{n}\,\mathbf{0})\ldots)) \xmapsto{val_2} n.$$

Let us remark that the fact that we are fixing a particular simple sort for numerals restricts the arithmetic that can be performed on those numerals, compared to when polymorphic sorts can be used. In particular such numerals cannot be subtracted, in contrast to numerals of polymorphic sort. This is essential for our results, with the following intuition: in order to approximate $n+m$ it is enough to approximate n and m, but in order to approximate $n-m$ it is necessary to know n and m quite precisely (to detect whether $n-m$ is close to 0 we need to know whether n is close to m; knowing that say $n \leq 2m \leq 4n$ is highly insufficient). See also page 32 for a discussion on polymorphic λ-calculus.

3 Type System and Values of Terms

In this section we define types which will be used to precisely describe our terms. Then, we define the vector of numbers assigned to each term. Finally, we state the main theorem of this paper, Theorem 10. The proof of this theorem is postponed to the next section.

Eliminating Higher-Order Constants. Our type system will be defined for terms using only constants of order at most 1. Instead of handling constants of higher orders directly in the type system, we now show how to replace them by combinations of constants of order at most 1.

Arguments of the original constants were of positive order, which means that they were expecting further arguments; now we give them some other constants as arguments. This is described by the following mutual definitions of *flatten* and *carg*, working by induction

on sort:

$$\mathit{flatten}(\mathbf{c}^\alpha) = \begin{cases} \mathbf{c}^\alpha, & \text{if } \mathit{ord}(\alpha) \leq 1, \\ \lambda x_1^{\alpha_1}. \cdots .\lambda x_k^{\alpha_k}.\underbrace{\mathbf{d}_1\,(\ldots\,(\mathbf{d}_1}_{|\alpha|-k-1}\,(\mathbf{d}_k\,\mathit{carg}(x_1^{\alpha_1})\ldots\mathit{carg}(x_k^{\alpha_k})))\ldots), & \\ & \text{otherwise, if } \alpha = \alpha_1 \to \ldots \to \alpha_k \to o, \end{cases}$$

$$\mathit{carg}(x^{\alpha_1\to\ldots\to\alpha_k\to o}) = x^{\alpha_1\to\ldots\to\alpha_k\to o}\,\mathit{flatten}(\mathbf{d}^{\alpha_1})\ldots\mathit{flatten}(\mathbf{d}^{\alpha_k}).$$

In this definition $\mathbf{d}_k$ is a fixed constant of sort $\underbrace{o\to\ldots\to o}_{k}\to o$, and $\mathbf{d}^\alpha$ is a fixed constant of sort α. Notice that $\mathit{flatten}(\mathbf{c}^\alpha)$ is a term of sort α (the sort is unchanged) and $\mathit{carg}(\mathbf{c}^\alpha)$ is a term of sort o. We extend $\mathit{flatten}(\cdot)$ to arbitrary terms in the natural way:

$$\mathit{flatten}(MN) = \mathit{flatten}(M)\mathit{flatten}(N), \qquad \mathit{flatten}(YM) = Y\mathit{flatten}(M),$$
$$\mathit{flatten}(x^\alpha) = x^\alpha, \qquad \mathit{flatten}(\lambda x^\alpha.M) = \lambda x^\alpha.\mathit{flatten}(M).$$

Example 3
Let us see how some example constant will be transformed:

$$\begin{aligned} &\mathit{flatten}(\mathbf{c}^{(((o\to o)\to o)\to o)\to((o\to o)\to o)\to(o\to o)\to o\to o}) \\ &\quad = \lambda x_3^{((o\to o)\to o)\to o}.\lambda x_2^{(o\to o)\to o}.\lambda x_1^{o\to o}.\lambda x_0^o.\mathbf{d}_1\,(\mathbf{d}_1\,(\mathbf{d}_1\,(\mathbf{d}_1\,(\mathbf{d}_1 \\ &\qquad (\mathbf{d}_4\,(x_3\,(\lambda y_1^{o\to o}.\mathbf{d}_1\,(\mathbf{d}_1\,(y_1\,\mathbf{d}_0))))\,(x_2\,\mathbf{d}_1)\,(x_1\,\mathbf{d}_0)\,x_0)))))). \end{aligned}$$

It is easy to see that if $M \to^*_{\beta\delta} N$ then also $\mathit{flatten}(M) \to^*_{\beta\delta} \mathit{flatten}(N)$: we simply replace every constant $\mathbf{c}$ in the sequence of reductions by $\mathit{flatten}(\mathbf{c})$, and we obtain a correct sequence of reductions from $\mathit{flatten}(M)$ to $\mathit{flatten}(N)$. It follows that if M $\beta\delta$-normalizes, then $\mathit{flatten}(M)$ as well. Indeed, the $\beta\delta$-normal form N of M does not contain any Y, as well as $\mathit{flatten}(N)$, so $\mathit{flatten}(N)$ $\beta\delta$-normalizes, and thus $\mathit{flatten}(M)$ $\beta\delta$-normalizes as well thanks to the sequence of reductions $\mathit{flatten}(M) \to^*_{\beta\delta} \mathit{flatten}(N)$.

The key property of the *flatten* operation is that it only slightly changes the size of the $\beta\delta$-normal form of a term (thus also the represented number).[2] Let us denote the size of the $\beta\delta$-normal form of a term M by $\mathbf{size}(M)$.

Lemma 4

1. For every closed term M that is of sort o and in $\beta\delta$-normal form it holds that $|M| \leq \mathbf{size}(\mathit{flatten}(M)) \leq |M|^2$.
2. For every closed term M that is of some sort α and in $\beta\delta$-normal form it holds that $|M| \leq |\alpha| - 1 + \mathbf{size}(\mathit{carg}(x^\alpha)[\mathit{flatten}(M)/x^\alpha]) \leq (|M| + |\alpha|)^2$.

[2] We remark that as a result of applying our encoding and $\beta\delta$-normalizing the term, the information about variable binding is lost (e.g. even if a closed term M of sort o has a lot of variables, all of them are replaced by constants in the $\beta\delta$-normal form of $\mathit{flatten}(M)$), so it is impossible to recover the original term from the encoding. This is irrelevant for us, as we only care about the size of terms, up to domination equivalence. Let us mention, however, in this place, a result (Clairambault & Murawski, 2013) which shows how to encode terms with higher-order constants in terms with constants of order at most 1 in a nice, compact way, so that all information is preserved.

Proof
We use induction, where the induction parameter is $|M|$ for the first point and $|M|+|\alpha|$ for the second point. For the first part, observe that M is of the form $\mathbf{c}^\alpha M_1 \dots M_k$, where $\alpha = \alpha_1 \to \dots \to \alpha_k \to o$. If $ord(\alpha) \leq 1$, it holds that $\mathit{flatten}(M) = \mathbf{c}^\alpha \mathit{flatten}(M_1) \dots \mathit{flatten}(M_k)$, and using the induction assumption for $M_1, \dots, M_k$, that are of sort o, we obtain

$$\begin{aligned}|M| &= |\alpha| + |M_1| + \dots + |M_k| \\ &\leq |\alpha| + \mathbf{size}(\mathit{flatten}(M_1)) + \dots + \mathbf{size}(\mathit{flatten}(M_k)) = \mathbf{size}(\mathit{flatten}(M)) \\ &\leq |\alpha| + |M_1|^2 + \dots + |M_k|^2 \leq (|\alpha| + |M_1| + \dots + |M_k|)^2 = |M|^2.\end{aligned}$$

Next, suppose that $ord(M) > 1$. In this case $\mathit{flatten}(M)$ β-reduces (in k steps) to the term

$$\underbrace{\mathbf{d}_1 (\dots (\mathbf{d}_1}_{|\alpha|-k-1} (\mathbf{d}_k \, carg(x_1^{\alpha_1})[\mathit{flatten}(M_1)/x_1^{\alpha_1}] \dots carg(x_k^{\alpha_k})[\mathit{flatten}(M_k)/x_k^{\alpha_k}])) \dots),$$

thus, recalling that $|\alpha| = |\alpha_1| + \dots + |\alpha_k| + 1$, we have

$$\begin{aligned}\mathbf{size}(\mathit{flatten}(M)) &= 2(|\alpha| - k - 1) + (k+1) + \sum_{i=1}^{k} \mathbf{size}(carg(x_i^{\alpha_i})[\mathit{flatten}(M_i)/x_i^{\alpha_i}]) \\ &= |\alpha| + \sum_{i=1}^{k} \left(|\alpha_i| - 1 + \mathbf{size}(carg(x_i^{\alpha_i})[\mathit{flatten}(M_i)/x_i^{\alpha_i}])\right).\end{aligned}$$

Since $|M_i| + |\alpha_i| < |M|$, we can use the second part of the induction assumption for the subterms $M_1, \dots, M_k$; we obtain

$$\begin{aligned}|M| &= |\alpha| + |M_1| + \dots + |M_k| \\ &\leq |\alpha| + \sum_{i=1}^{k} \left(|\alpha_i| - 1 + \mathbf{size}(carg(x_i^{\alpha_i})[\mathit{flatten}(M_i)/x_i^{\alpha_i}])\right) = \mathbf{size}(\mathit{flatten}(M)) \\ &\leq |\alpha| + \sum_{i=1}^{k} (|M_i| + |\alpha_i|)^2 \leq |\alpha| + (|M_1| + |\alpha_1| + \dots + |M_k| + |\alpha_k|)^2 \\ &= |\alpha| + (|M| - 1)^2 \leq 2|M| - 1 + (|M| - 1)^2 = |M|^2.\end{aligned}$$

For the second point, let us write $\alpha = \alpha_1 \to \dots \to \alpha_k \to o$ and $M = \lambda x_1^{\alpha_1}. \dots .\lambda x_m^{\alpha_m}.N$, where N does not begin with a lambda (we have $m \leq k$); we want to estimate the size of the $\beta\delta$-normal form of the term $K = carg(x^\alpha)[\mathit{flatten}(M)/x^\alpha]$. Observe that

$$\begin{aligned}&K = \mathit{flatten}((\lambda x_1^{\alpha_1}. \dots .\lambda x_m^{\alpha_m}.N)\, \mathbf{d}^{\alpha_1} \dots \mathbf{d}^{\alpha_k}) \to^*_{\beta\delta} \mathit{flatten}(L), \qquad \text{where} \\ &L = N[\mathbf{d}^{\alpha_1}/x_1^{\alpha_1}, \dots, \mathbf{d}^{\alpha_m}/x_m^{\alpha_m}]\, \mathbf{d}^{\alpha_{m+1}} \dots \mathbf{d}^{\alpha_k}.\end{aligned}$$

Notice that L is in $\beta\delta$-normal form, and its size satisfies

$$|N| \leq |L| = |N| + |\alpha_{m+1}| + \dots + |\alpha_k| \leq |N| + |\alpha| - 1 \leq |M| + |\alpha| - 1.$$

In particular we can use the first part of the induction assumption for L; we obtain

$$\begin{aligned}|M| &= |\alpha_1| + \dots + |\alpha_m| + |N| \leq |\alpha| - 1 + |L| \leq |\alpha| - 1 + \mathbf{size}(K) \\ &\leq |\alpha| - 1 + |L|^2 \leq |\alpha| - 1 + (|M| + |\alpha| - 1)^2 \leq (|M| + |\alpha|)^2. \quad \square\end{aligned}$$

Corollary 5

It holds that $\mathbf{size}(M) \approx \mathbf{size}(\mathit{flatten}(M))$, where M ranges over all closed terms of sort o that $\beta\delta$-normalize.

The Type System. Our types differ from sorts in that on the left-hand side of $\to$, instead of a single type, we have a set of pairs (f,τ), where τ is a type, and f is a number called the *duplication order*. The unique atomic type is denoted $\mathbf{r}$. Also we will not only assign a type τ to our terms, but also a duplication flag F (which together form a pair (F,τ)).

More precisely, for each sort α we define the set $\mathscr{T}^\alpha$ of types of sort α, by induction on α, as follows:

$$\mathscr{T}^o = \{\mathbf{r}\}, \qquad \mathscr{T}^{\alpha\to\beta} = \mathscr{P}_{cons}(\{-\infty,0,\dots,ord(\alpha)\}\times\mathscr{T}^\alpha)\times\mathscr{T}^\beta,$$

where $\mathscr{P}_{cons}(\{-\infty,0,\dots,ord(\alpha)\}\times\mathscr{T}^\alpha)$ denotes the set of *consistent* (see below) subsets of $\{-\infty,0,\dots,ord(\alpha)\}\times\mathscr{T}^\alpha$. A type $(T,\tau)\in\mathscr{T}^{\alpha\to\beta}$ is always denoted as $\bigwedge T\to\tau$, or $\bigwedge_{i\in I}(f_i,\tau_i)\to\tau$ when $T=\{(f_i,\tau_i)\mid i\in I\}$.

For $\alpha=\alpha_1\to\dots\to\alpha_k\to o$, a set $T\subseteq\{-\infty,0,\dots,ord(\alpha)\}\times\mathscr{T}^\alpha$ of type pairs is *consistent* if for each tuple of consistent sets $U_1\subseteq\mathscr{T}^{\alpha_1},\dots,U_k\subseteq\mathscr{T}^{\alpha_k}$ there exists at most one pair $(f,\bigwedge T_1\to\dots\to\bigwedge T_k\to\mathbf{r})\in T$ such that $T_i\subseteq U_i$ for each $i\in\{1,\dots,k\}$. Again, this definition is by induction on the sort.

Intuitively, a term has type $\bigwedge T\to\tau$ if it returns a term described by τ, when given an argument for which we can derive all type pairs from T. It will be important to estimate how many constants will be used in the $\beta\delta$-normal form of a term. The duplication order f helps with that. It is used to describe the way in which a term is productive, that is how it increases the number of constants in the $\beta\delta$-normal form. When a term uses a constant, we set $f\geq 0$. Notice however that this constant has to be really used: there exist terms which syntactically contain a constant, but such that it will disappear during reductions, like in $(\lambda x.\lambda y.y)\,\mathbf{c}$. But a subterm can increase the number of constants in the $\beta\delta$-normal form even if it does not contain any constant: it may take a constant as an argument and use this argument more than once. For that reason, when our term takes an argument of some duplication order f', and uses it more than once ("duplicates it"), we set $f\geq f'+1$.

In a type $\bigwedge T\to\tau$ we always assume that the set T is consistent. The reason is that, on the one hand, for each term the set of all its types is consistent, as we will see later, so it does not make sense to expect an argument whose set of types is not consistent. And, on the other hand, by restricting ourselves to types $\bigwedge T\to\tau$ where T is consistent, we obtain better properties of the type system, like uniqueness of derivation.

A *type environment* Γ is a set of bindings of variables of the form $x^\alpha:(f,\tau)$, where $\tau\in\mathscr{T}^\alpha$ and $f\in\{-\infty,0,\dots,ord(\alpha)\}$; for each variable x^α it is required that the set $\{(f,\tau)\mid (x^\alpha:(f,\tau))\in\Gamma\}$ of all type pairs assigned to it is consistent. In Γ we may have multiple bindings for the same variable. By $dom(\Gamma)$ we denote the set of variables x which are bound by Γ.

A *type judgement* is of the form $\Gamma\vdash M:(F,\tau)$, where we require that the type τ and the term M are of the same sort, and where F, called a *duplication flag*, is a function $F\colon\mathscr{P}(dom(\Gamma))\to\{-\infty\}\cup\mathbb{N}$. Here, instead of a single duplication order f, we have a function F which assigns a duplication order separately to each subset of $dom(\Gamma)$. Intu-

itively, $F(X)$ is the duplication order of the term obtained by substituting closed terms (of types corresponding to the type binding in Γ) for all variables in X.

In our type system we assume that only constants of order at most 1 appear in our terms. Constants of higher orders should be replaced by a combination of constants of order 0 and 1, as shown before.

The type system consists of the following rules:

$$\frac{F_x(\emptyset) = -\infty}{x : (F_x(\{x\}), \tau) \vdash x : (F_x, \tau)}\ (\textsc{Var}) \qquad \frac{\alpha = \overbrace{o \to \ldots \to o}^{k} \to o \qquad F_0(\emptyset) = 0}{\emptyset \vdash \mathbf{c}^\alpha : (F_0, \underbrace{(0,\mathbf{r}) \to \ldots \to (0,\mathbf{r})}_{k} \to \mathbf{r})}\ (\textsc{Con})$$

$$\frac{\Gamma \vdash M\,(Y\,M) : (F, \tau)}{\Gamma \vdash Y\,M : (F, \tau)}\ (Y) \qquad \frac{\Gamma \cup \{x : (f_i, \tau_i) \mid i \in I\} \vdash M : (F, \tau) \qquad x \notin dom(\Gamma)}{\Gamma \vdash \lambda x.M : (F\restriction_{\mathscr{P}(dom(\Gamma))}, \bigwedge_{i \in I}(f_i, \tau_i) \to \tau)}\ (\lambda)$$

$$\frac{\Gamma_0 \vdash M : (F_0, \bigwedge_{i \in I}(F_i(dom(\Gamma_i)), \tau_i) \to \tau) \qquad \Gamma_i \vdash N : (F_i, \tau_i) \text{ for each } i \in I \qquad 0 \notin I}{\bigcup_{i \in \{0\} \cup I} \Gamma_i \vdash M\,N : (F, \tau)}\ (@)$$

where in the $(@)$ rule we assume that

- the types τ_i are pairwise different (where $i \in I$), and
- with $\tau = \bigwedge T_1 \to \ldots \to \bigwedge T_k \to \mathbf{r}$ for each $X \subseteq dom(\bigcup_{i \in \{0\} \cup I} \Gamma_i)$ the following holds:

$$\begin{aligned} F(X) = \min\big(&\max(\{F_i(X \cap dom(\Gamma_i)) \mid i \in \{0\} \cup I\} \\ &\qquad \cup \{n \mid \exists i, j \in \{0\} \cup I.\, \exists x \notin X.\, \exists \sigma.\, i \neq j \wedge (x : (n-1, \sigma)) \in \Gamma_i \cap \Gamma_j\}), \\ &\max(\{0\} \cup \{n \mid \exists \sigma.\, (n-1, \sigma) \in \bigcup_{i \in \{1,\ldots,k\}} T_i\} \\ &\qquad \cup \{n \mid \exists x \notin X.\, \exists \sigma.\, (x : (n-1, \sigma)) \in \bigcup_{i \in \{0\} \cup I} \Gamma_i\})\big). \end{aligned} \tag{1}$$

A *derivation tree* is defined as usual: it is a tree labeled by type judgements, such that each node together with its children fit to one of the rules of the type system. For an example of a derivation tree see Section 5.1.

Notice that strengthening of type environments is disallowed (i.e., $\Gamma \vdash M : (F, \tau)$ does not necessarily imply $\Gamma, x : (g, \sigma) \vdash M : (F, \tau)$), but contraction is allowed (i.e., $\Gamma, x : (g, \sigma), x : (g, \sigma) \vdash M : (F, \tau)$ implies $\Gamma, x : (g, \sigma) \vdash M : (F, \tau)$, since a type environment is a set of type bindings); such contractions will be counted by *duplication factors* defined later.

We see that in order to derive a type for $Y\,M$, we first need to derive a type for $M\,(Y\,M)$, but in order to derive a type for $M\,(Y\,M)$ it is not necessary to derive a type for $Y\,M$: the set I in the $(@)$ rule may be empty.

Let us explain condition (1) defining the duplication flag in the $(@)$ rule. As already mentioned, for a fixed set X of variables, the intended meaning of $F(X)$ is that this should be the duplication order of the term obtained from $M\,N$ by substituting appropriate terms for all variables in X. Thus we collect (we take a maximum of) corresponding duplication

orders $F_i(X \cap dom(\Gamma_i))$ from premisses of the rule, where we need to restrict X to those variables which indeed appear in the premiss. Additionally, the duplication order should be at least n if already prior to the application we duplicate some binding of duplication order $n-1$; we are interested only in bindings for variables not in X, since afterwards the imaginary substitution the variables from X will disappear. It is however possible that a premiss has some big duplication order n, but in the conclusion we only have variables and arguments of duplication order smaller than $n-1$. Then this duplication may be only used to duplicate some other premiss and is not visible outside of our conclusion. Thus in the formula we have a minimum, dropping the duplication order down to a greatest n such that there is either an argument or a binding in the environment having duplication order $n-1$.

We notice that it is possible to derive $\Gamma \vdash M : (F, \tau)$ with $F(X) > ord(M)$. However $F(X)$ is bounded by the order in the following sense.

Proposition 6
Let $\Gamma \vdash M : (F, \tau)$ be a derivable type judgement, and let $X \subseteq dom(\Gamma)$. Then $F(X) \leq \max(\{ord(M)\} \cup \{ord(x)+1 \mid x \in dom(\Gamma) \setminus X\})$.

Proof
Induction on the size of the smallest derivation tree for $\Gamma \vdash M : (F, \tau)$. This is immediate when this tree consists of a single (VAR) or (CON) rule (for the (VAR) rule we recall that the definition of a type environment only allows bindings $x : (f, \sigma)$ with $f \leq ord(x)$). When the derivation tree starts by the (Y) rule, we can directly use the induction assumption for the rest of the tree. Similarly for the (λ) rule, where for $M = \lambda y.K$ and $\tau = \bigwedge_{i \in I}(f_i, \tau_i) \to \tau'$ we observe that

$$\begin{aligned}
&\max(\{ord(K)\} \cup \{ord(x)+1 \mid x \in dom(\Gamma \cup \{y : (f_i, \tau_i) \mid i \in I\}) \setminus X\}) \\
&\qquad \leq \max(\{ord(K), ord(y)+1\} \cup \{ord(x)+1 \mid x \in dom(\Gamma) \setminus X\}) \\
&\qquad = \max(\{ord(M)\} \cup \{ord(x)+1 \mid x \in dom(\Gamma) \setminus X\}).
\end{aligned}$$

When the first rule is $(@)$ we (without using the induction assumption) observe the second max in the formula (1) for $F(X)$. This max (and hence $F(X)$) cannot be greater than $\max(\{ord(M)\} \cup \{ord(x)+1 \mid x \in dom(\Gamma) \setminus X\})$. $\square$

Consistency and Uniqueness of Type Derivations. We see that there may be multiple type judgements for the same term. Thus potentially it might be possible to derive the same type judgement in multiple ways, because in the $(@)$ rule different premisses may give the same conclusion. It turns out, however, that it is impossible: each type judgement can be derived in only one way. Even more: for a given term we cannot derive an arbitrary set of type pairs; this set will be always consistent.

For two types $\tau = \bigwedge T_1 \to \ldots \to \bigwedge T_k \to \mathbf{r}$ and $\sigma = \bigwedge U_1 \to \ldots \to \bigwedge U_k \to \mathbf{r}$, we write $\tau \preceq \sigma$ if $T_i \subseteq U_i$ for each $i \in \{1, \ldots, k\}$. Notice that a set $T \subseteq \{-\infty, 0, \ldots, ord(\alpha)\} \times \mathscr{T}^\alpha$ is consistent if and only if for each $\tau_{max} \in \mathscr{T}^\alpha$ there exists at most one pair $(f, \tau) \in T$ such that $\tau \preceq \tau_{max}$.

Lemma 7
Let K be a term of sort α, let Γ_{max} be a type environment, and let $\tau_{max} \in \mathscr{T}^\alpha$. Then there exists at most one derivation tree for a type judgement $\Gamma \vdash K : (F, \tau)$ such that $\Gamma \subseteq \Gamma_{max}$ and $\tau \preceq \tau_{max}$.

In this lemma we claim that both the type judgement and the derivation tree is unique. As written in the definition of a type environment, it is required that the set of all type pairs assigned by Γ_{max} to any variable is consistent.

The intuitive explanation of the lemma is as follows. Suppose that we have fixed some terms used for arguments of a term K, and terms which we will substitute for free variables in K. This determines how the term K will "behave": how the arguments and the free variables will be used, that is which of the type pairs for arguments and for free variables will be used; it also determines which variables will be used more than once, so it determines the duplication order. It turns out that instead of fixing completely the terms used for arguments and for substitution, it is enough to know all type pairs which we can assign to them (they are described by τ_{max} and Γ_{max}).

Proof of Lemma 7

It is enough to prove that for each $m \in \mathbb{N}$ there exists at most one derivation tree of height at most m for a type judgement $\Gamma \vdash K : (F, \tau)$ such that $\Gamma \subseteq \Gamma_{max}$ and $\tau \preceq \tau_{max}$. This is proved by induction on m. We analyze possible shapes of the term K.

Suppose first that $K = x$ is a variable. Then all possible type judgements are of the form $x : (F_x(\{x\}), \tau) \vdash x : (F_x, \tau)$, where $F_x(\emptyset) = -\infty$. Because the set of pairs (f, τ) such that $(x : (f, \tau)) \in \Gamma_{max}$ is consistent, there exists at most one pair (f, τ) such that $\tau \preceq \tau_{max}$ and $(x : (f, \tau)) \in \Gamma_{max}$; we have to take $F_x(\{x\}) = f$ and $F_x(\emptyset) = -\infty$.

Next, suppose that $K = \mathbf{c}^{\alpha}$ is a constant. Then we observe that in the (CON) rule there is no freedom, it is possible to derive only one type judgement for K.

Next, suppose that $K = YM$. Then in order to derive $\Gamma \vdash YM : (F, \tau)$ we need to derive $\Gamma \vdash M(YM) : (F, \tau)$ by a tree of height at most $m - 1$. By the induction assumption there is only one such tree.

Next, suppose that $K = \lambda x.M$. Then in order to derive $\Gamma \vdash \lambda x.M : (F, \tau)$ where $\tau = \bigwedge T \to \tau'$ we need a derivation tree of height at most $m - 1$ for $\Gamma \cup \{x : (f, \sigma) \mid (f, \sigma) \in T\} \vdash M : (F', \tau')$, where $x \notin dom(\Gamma)$. Denote $\tau_{max} = \bigwedge T_{max} \to \tau'_{max}$. The conditions $\Gamma \subseteq \Gamma_{max}$ and $\tau \preceq \tau_{max}$ can be rewritten as $\Gamma \cup \{x : (f, \sigma) \mid (f, \sigma) \in T\} \subseteq \Gamma_{max} \cup \{x : (f, \sigma) \mid (f, \sigma) \in T_{max}\}$ and $\tau' \preceq \tau'_{max}$. By the induction assumption there is at most one derivation tree satisfying these conditions. Notice that F is determined by F': it holds that $F = F' \restriction_{\mathscr{P}(dom(\Gamma))}$.

Finally, suppose that $K = MN$. Consider the set T_{max} containing all pairs $(G(dom(\Sigma)), \sigma)$ such that in at most $m - 1$ steps we can derive a type judgement $\Sigma \vdash N : (G, \sigma)$ where $\Sigma \subseteq \Gamma_{max}$. By the induction assumption, for each σ_{max} there is at most one derivation tree of height at most $m - 1$ for a type judgement $\Sigma \vdash N : (G, \sigma)$ such that $\Sigma \subseteq \Gamma_{max}$ and $\sigma \preceq \sigma_{max}$. This implies that the set T_{max} is consistent. In order to derive a type judgement $\Gamma \vdash MN : (F, \tau)$ such that $\Gamma \subseteq \Gamma_{max}$ and $\tau \preceq \tau_{max}$ in at most m steps we need to derive a type judgement $\Gamma' \vdash M : (F', \bigwedge T \to \tau)$ such that $\Gamma' \subseteq \Gamma_{max}$ and $T \subseteq T_{max}$ and $\tau \preceq \tau_{max}$ in at most $m - 1$ steps. The last two conditions can be rewritten as $(\bigwedge T \to \tau) \preceq (\bigwedge T_{max} \to \tau_{max})$. By the induction assumption, there exists at most one derivation tree of such $\Gamma' \vdash M : (F', \bigwedge T \to \tau)$. The set T determines which derivation trees for type judgements $\Sigma \vdash N : (G, \sigma)$ have to be included in a derivation tree for $\Gamma \vdash MN : (F, \tau)$; this in turn determines Γ and F. $\square$

Lower Cost of a Derivation Tree. Consider a node of a derivation tree in which the (@) rule is used, with type environments Γ_i for $i \in \{0\} \cup I$. The *duplication factor* in such a

node is defined as

$$\left(\sum_{i\in\{0\}\cup I} |\{(x:(f,\sigma))\in\Gamma_i \mid f\geq 0\}|\right) - |\{(x:(f,\sigma))\in \bigcup_{i\in\{0\}\cup I}\Gamma_i \mid f\geq 0\}|.$$

In other words, this is equal to the number of type bindings with nonnegative duplication order together in all the type environments $(\Gamma_i)_{i\in\{0\}\cup I}$, minus the number of such type bindings in their union. That is, we say how many times we have to duplicate a binding in Γ, so that the resulting multiset splits into $(\Gamma_i)_{i\in\{0\}\cup I}$. We ignore here bindings with duplication factor $-\infty$ (intuitively, duplicating them does not increase the number of constants in the $\beta\delta$-normal form of the whole term).

The *lower cost* of a derivation tree D, denoted $low(D)$, equals the sum of duplication factors in all nodes of this tree, plus the number of its nodes using the (CON) rule (we use the name "lower cost" in opposition to the "higher cost", defined later). See Section 5.2 for a calculation of the lower cost of example derivation trees.

Let M be a closed term. We define $types(M)$ to be the set of those type pairs (f,τ) for which we can derive $\vdash flatten(M):(F,\tau)$ with $F(\emptyset)=f$. Recall that for each $(f,\tau)\in types(M)$ we have a unique derivation tree for $\vdash flatten(M):(F,\tau)$ (see Lemma 7). By $\mathbf{low}(M,(f,\tau))$ we denote the lower cost of this tree.

As already observed in Proposition 6, it is always the case that $F(\emptyset)\leq ord(flatten(M))=ord(M)$ for a derivable type judgement $\vdash flatten(M):(F,\tau)$. Thus for terms M of a fixed sort α, the set $types(M)$ is a subset of a fixed finite set $\{-\infty,0,\ldots,ord(\alpha)\}\times\mathscr{T}^{\alpha}$.

Notice that if all constants in a term M are of order at most 1, then $M=flatten(M)$, so for such terms the use of $flatten(\cdot)$ in the definition of *types* and **low** does not change anything.

Compositionality. Let us see how we assign types to a composition MN of two closed terms. Looking at our type system we observe that $types(MN)$ contains a pair (f,τ) if and only if $types(M)$ contains a pair $(f',\bigwedge T\to\tau)$ such that $T\subseteq types(N)$ and

$$f=\min\big(\max(\{f'\}\cup\{g\mid(g,\sigma)\in T\}),\max(\{0\}\cup\{n\mid\exists\sigma.\,(n-1,\sigma)\in\bigcup_{i\in\{1,\ldots,k\}}T_i\})\big),$$

where $\tau=\bigwedge T_1\to\ldots\to\bigwedge T_k\to\mathbf{r}$. Thus $types(MN)$ depends only on $types(M)$ and $types(N)$.

Let us now see how $\mathbf{low}(MN,(f,\tau))$ can be calculated, where $(f,\tau)\in types(MN)$. Notice that the duplication factor in the root of the derivation tree for $\vdash flatten(MN):(f,\tau)$ is always 0, because the type environments are empty. As written above, $(f,\tau)\in types(MN)$ implies that $types(M)$ contains a pair $(f',\bigwedge T\to\tau)$ such that $T\subseteq types(N)$; by uniqueness of the derivation tree, there must be at most one such pair. We have

$$\mathbf{low}(MN,(f,\tau))=\mathbf{low}(M,(f',\bigwedge T\to\tau))+\sum_{(g,\sigma)\in T}\mathbf{low}(N,(g,\sigma)).$$

Thus, if we view $\mathbf{low}(MN,\cdot)$ as on a vector of numbers (whose entries are $\mathbf{low}(MN,(f,\tau))$ for all $(f,\tau)\in types(MN)$, aligned in some fixed order), this vector is obtained by applying a linear function to the vectors $\mathbf{low}(M,\cdot)$ and $\mathbf{low}(N,\cdot)$. Moreover, the linear function itself depends only on $types(M)$ and $types(N)$.

Costs During Reductions. The technical core of our result is contained in the following lemma, which says that the lower costs assigned to a term cannot change too much during

reductions. This means that the number assigned to a closed term M approximates the number assigned to the $\beta\delta$-normal form of M. At the end we will be interested in terms of sort o, so we present the lemma only for such terms.

Lemma 8
There exists a correction function $H\colon \mathbb{N} \to \mathbb{N}$ such that whenever N is the $\beta\delta$-normal form of a closed term M that is of sort o and contains only constants of order at most 1, then $(0,\mathbf{r}) \in \mathit{types}(M) \cap \mathit{types}(N)$ and $\mathbf{low}(M,(0,\mathbf{r})) \approx_H \mathbf{low}(N,(0,\mathbf{r}))$.

This lemma will be proved in Section 4.

Example 4
Let us consider sorts $\alpha_0 = o$ and $\alpha_i = \alpha_{i-1} \to \alpha_{i-1}$ for $i \geq 1$. For $i \geq 1$ we define a term $K_i = \lambda f^{\alpha_i}.\lambda y^{\alpha_{i-1}}.f\,(f\,y)$ of sort α_{i+1}. Concentrate on the term

$$M_n = K_n\,K_{n-1}\,\ldots\,K_1\,(\lambda x^o.\mathbf{c}^{o\to o}\,x)\,\mathbf{d}^o.$$

It is not difficult to see that $\mathbf{low}(M_n,(0,\mathbf{r})) = n+2$: we have two constants, and in each K_i the first argument is used twice. Notice however that the $\beta\delta$-normal form N_n of M_n is very big. Indeed, each K_i causes that its argument will be used twice. Thus K_n causes that K_{n-1} will be used twice; these two K_{n-1} cause that K_{n-2} will be used 2^2 times; in turn these K_{n-2} cause that K_{n-3} will be used 2^{2^2} times, and so on. Altogether, $\mathbf{low}(N_n,(0,\mathbf{r}))$ will be nonelementary in n. In consequence $H(n)$ in Lemma 8 has to be nonelementary in n.

Relating Costs with Represented Numbers. Beside of the fact that the lower cost of a term (closed, of sort o) approximates the lower cost of its $\beta\delta$-normal form, we also need to know that the lower cost of a term in $\beta\delta$-normal form approximates its size (and thus the numeral represented by the term).

Lemma 9
Let M be a closed term that is of sort o, in $\beta\delta$-normal form, and contains only constants of order at most 1. Then $(0,r) \in \mathit{types}(M)$ and $|M| = 2\cdot\mathbf{low}(M,(0,\mathbf{r})) - 1$.

Proof
Induction on $|M|$. Notice that M is necessarily of the form $\mathbf{c}M_1\ldots M_k$, where all M_i are again closed, of sort o, and in $\beta\delta$-normal form. The induction assumption gives us, for each i, a derivation tree D_i for $\vdash M_i : (F_0,\mathbf{r})$, where $F_0(\emptyset) = 0$ and $|M_i| = 2\cdot \mathit{low}(D_i) - 1$. We use the (Con) rule to derive $\vdash \mathbf{c} : (F_0,(0,\mathbf{r})\to\ldots\to(0,\mathbf{r})\to\mathbf{r})$, and then using k times the (@) rule we attach the derivation trees D_i, obtaining a derivation tree for $\vdash M : (F_0,\mathbf{r})$. The lower cost of this tree, that is simultaneously $\mathbf{low}(M,(0,\mathbf{r}))$, is 1 (for the new (Con) rule) plus the lower costs of the trees D_i. We see that

$$\begin{aligned}|M| &= (k+1) + \sum_{i=1}^{k}|M_i| = k+1+\sum_{i=1}^{k}(2\cdot \mathit{low}(D_i)-1)\\ &= 2\cdot(1+\sum_{i=1}^{k}\mathit{low}(D_i)) - 1 = 2\cdot\mathbf{low}(M,(0,\mathbf{r})) - 1. \quad \square\end{aligned}$$

We see that $\mathbf{low}(M,(0,\mathbf{r}))$ counts precisely the number of constants used in M.

The Main Theorem. We are now ready to state and prove the main theorem of our paper, saying that the values assigned to terms approximate the numbers represented by the terms.

Theorem 10
Let *val* be a numeral system using terms of sort $\alpha_{\mathbb{N}}$. Then there exists a (choice) function t mapping each nonempty set $T \subseteq \{-\infty, 0, \ldots, ord(\alpha_{\mathbb{N}})\} \times \mathscr{T}^{\alpha_{\mathbb{N}}}$ to some its element such that $\mathbf{low}(M, t(types(M))) \approx \mathbf{val}(M)$, where M ranges over $dom(\mathbf{val})$ (that is, over all closed terms of sort $\alpha_{\mathbb{N}}$ that represent some number); in particular $types(M)$ is nonempty for these M.

Let us emphasize that the correction function hidden in the domination equivalence $\mathbf{low}(M, t(types(M))) \approx \mathbf{val}(M)$ does not depend on M (but depends on the numeral system *val*).

Proof
Let $\alpha_{\mathbb{N}} = \alpha_1 \to \ldots \to \alpha_k \to o$. Fix some constants $\mathbf{c}_1, \ldots, \mathbf{c}_k$ of sorts $\alpha_1, \ldots, \alpha_k$, respectively. For $i \in \{1, \ldots, k\}$, let T_i be the set of all type pairs $(G(\emptyset), \sigma)$ such that we can derive $\emptyset \vdash flatten(\mathbf{c}_i) : (G, \sigma)$, and let $\tau_{term} = \bigwedge T_1 \to \ldots \to \bigwedge T_k \to \mathbf{r}$. Thanks to Lemma 7 we can notice that each set T_i is consistent, so τ_{term} is indeed a type (belongs to $\mathscr{T}^{\alpha_{\mathbb{N}}}$). We choose t as follows: if $T \subseteq \{-\infty, 0, \ldots, ord(\alpha_{\mathbb{N}})\} \times \mathscr{T}^{\alpha_{\mathbb{N}}}$ contains a pair (f, τ) such that $\tau \preceq \tau_{term}$, we fix one such pair as $t(T)$; otherwise we just fix any element of T as $t(T)$.

Take now some term $M \in dom(\mathbf{val})$. Lemma 8 says that $types(flatten(M\,\mathbf{c}_1 \ldots \mathbf{c}_k))$ contains $(0, \mathbf{r})$, which means that there is a derivation tree for $\emptyset \vdash flatten(M\,\mathbf{c}_1 \ldots \mathbf{c}_k) : (F_0, \mathbf{r})$ where $F_0(\emptyset) = 0$. It ends with k nodes using the $(@)$ rule, above which we have derivation trees for $flatten(\mathbf{c}_i)$ and a derivation tree for $\emptyset \vdash flatten(M) : (F, \tau)$, where necessarily $\tau \preceq \tau_{term}$. In consequence $(F(\emptyset), \tau) \in types(M)$. Due to Lemma 7, this is the only pair (f, τ') in $types(M)$ such that $\tau' \preceq \tau_{term}$, so $(F(\emptyset), \tau) = t(types(M))$. Comparing the lower costs of the derivation trees, we see that

$$\begin{aligned}\mathbf{low}(M, t(types(M)) &\leq \mathbf{low}(flatten(M\,\mathbf{c}_1 \ldots \mathbf{c}_k), (0, \mathbf{r})) \\ &\leq \mathbf{low}(M, t(types(M))) + \sum_{i=1}^{k} \sum_{(g,\sigma) \in T_i} \mathbf{low}(\mathbf{c}_i, (g, \sigma)).\end{aligned}$$

This means that $\mathbf{low}(M, t(types(M)) \approx_{H_1} \mathbf{low}(flatten(M\,\mathbf{c}_1 \ldots \mathbf{c}_k), (0, \mathbf{r}))$ for the correction function $H_1(n) = n + \sum_{i=1}^{k} \sum_{(g,\sigma) \in T_i} \mathbf{low}(\mathbf{c}_i, (g, \sigma))$ independent of M.

Let N be the $\beta\delta$-normal form of $flatten(M\,\mathbf{c}_1 \ldots \mathbf{c}_k)$. Applying Lemma 8 for the term $flatten(M\,\mathbf{c}_1 \ldots \mathbf{c}_k)$ we obtain $\mathbf{low}(flatten(M\,\mathbf{c}_1 \ldots \mathbf{c}_k), (0, \mathbf{r})) \approx_{H_2} \mathbf{low}(N, (0, \mathbf{r}))$ for a correction function H_2 independent of M. Then, from Lemma 9 we obtain $\mathbf{low}(N, (0, \mathbf{r})) \approx_{H_3} |N|$ for $H_3(n) = 2n$, and $|N| = \mathbf{size}(flatten(M\,\mathbf{c}_1 \ldots \mathbf{c}_k))$. Next, due to Lemma 5 it holds that $\mathbf{size}(flatten(M\,\mathbf{c}_1 \ldots \mathbf{c}_k)) \approx_{H_4} \mathbf{size}(M\,\mathbf{c}_1 \ldots \mathbf{c}_k)$ for H_4 independent of M.

Let us now observe that $\mathbf{size}(M\,\mathbf{c}_1 \ldots \mathbf{c}_k) \approx_{H_5} \mathbf{size}(M)$ for $H_5(n) = n + |\alpha_1| + \cdots + |\alpha_k|$. Indeed, let K be the $\beta\delta$-normal form of M, and L the $\beta\delta$-normal form of $M\,\mathbf{c}_1 \ldots \mathbf{c}_k$, thus simultaneously of $K\,\mathbf{c}_1 \ldots \mathbf{c}_k$. Let us represent $K = \lambda x_1. \cdots .\lambda x_m.K'$, where K' does not begin with a lambda (necessarily $m \leq k$). Then $L = K'[\mathbf{c}_1/x_1 \ldots \mathbf{c}_m/x_m]\,\mathbf{c}_{m+1} \ldots \mathbf{c}_k$, and thus $|L| = |K| - |\alpha_1| - \cdots - |\alpha_m| + |\alpha_{m+1}| + \cdots + |\alpha_k|$.

Finally, by the definition of a numeral system we have $\mathbf{size}(M) \approx_{H_6} \mathbf{val}(M)$. Composing all the above equivalences together we obtain $\mathbf{low}(M, t(types(M))) \approx \mathbf{val}(M)$, where the correction function is independent of M. $\square$

Remark 11
Of course while considering a general numeral system we cannot be too specific about the relation between the costs and the represented numbers. We notice however that for the standard numeral systems given in Examples 1 and 2, this relation is completely straightforward. For the first of them the lower cost counts how many times the variable f is duplicated:

$$\mathbf{low}(val_1^{-1}(n), t(types(val_1^{-1}(n)))) = \begin{cases} 0 & \text{if } n = 0, \\ n-1 & \text{if } n \geq 1, \end{cases}$$

where

$$t(types(val_1^{-1}(n))) = \begin{cases} (-\infty, \top \to (0,\mathbf{r}) \to \mathbf{r}) & \text{if } n = 0, \\ (-\infty, (0,(0,\mathbf{r}) \to \mathbf{r}) \to (0,\mathbf{r}) \to \mathbf{r}) & \text{if } n = 1, \\ (1, (0,(0,\mathbf{r}) \to \mathbf{r}) \to (0,\mathbf{r}) \to \mathbf{r}) & \text{if } n \geq 2. \end{cases}$$

For the second numeral system the lower cost counts the number of constants in a term: $\mathbf{low}(val_2^{-1}(n), t(types(val_2^{-1}(n)))) = n+1$ where $t(types(val_2^{-1}(n))) = (0,\mathbf{r})$ for all n.

4 Costs During Reductions

In the previous section we have presented the main theorem, but its key component, that is Lemma 8, is not yet proved. We prove it in this section.

Higher Cost. It turns out that during leftmost reductions the lower cost of a term either increases or stays the same, but it cannot decrease. As observed in Fact 1 it is enough to consider leftmost reductions to obtain the $\beta\delta$-normal form of a term. In consequence, the lower cost of a term M is indeed a lower bound for the lower cost of the $\beta\delta$-normal form K of M. We now define another quantity, the *higher cost*, which will be the upper bound for the lower cost of K. This higher cost behaves in an opposite manner to the lower cost: it either decreases or stays the same during leftmost reductions, but it cannot increase. Additionally, the higher cost will be related to the lower cost by a function H: the higher cost can be very big only when the lower cost is also quite big.

We need to the look at the duplication factor more precisely. Consider a node of a derivation tree in which the $(@)$ rule is used, with type environments Γ_i for $i \in \{0\} \cup I$. For $a \in \mathbb{N}$, the *order-a duplication factor* in such a node is defined as

$$\left(\sum_{i \in \{0\} \cup I} |\{(x : (a-1, \sigma)) \in \Gamma_i\}| \right) - |\{(x : (a-1, \sigma)) \in \bigcup_{i \in \{0\} \cup I} \Gamma_i\}|.$$

In other words, this is equal to the number of type bindings with duplication order $a-1$ together in all the type environments $(\Gamma_i)_{i \in \{0\} \cup I}$, minus the number of such type bindings in their union. Notice that the sum of order-a duplication factors over all $a \in \mathbb{N}$ gives the standard duplication factor, and the order-0 duplication factor is always 0.

Consider now a derivation tree D. For each $a \in \mathbb{N}$ we define a value $loc_a(D)$ (loc stands for "local"). If the root (and thus the only node) of D uses the (CON) rule, $loc_0(D) = 1$ and $loc_a(D) = 0$ for all $a > 0$. If the root of D uses the (@) rule, $loc_a(D)$ equals the order-a duplication factor in the root of D (for all $a \in \mathbb{N}$). If the root of D uses any other rule, $loc_a(D) = 0$ for all $a \in \mathbb{N}$.

We notice that the sum of $loc_a(D')$ over all $a \in \mathbb{N}$ and over all subtrees D' of a tree D gives exactly the lower cost of D. In order to obtain the higher cost we will also collect all $loc_a(D')$, but in a more complicated way than just summing.

The values assigned to different duplication orders will be cumulated in an exponential way as follows. Suppose that we have a sequence $(n_a)_{a\in\mathbb{N}}$ of natural numbers in which only finitely many elements are positive. For such a sequence we define $cum_a((n_b)_{b\in\mathbb{N}})$ for each $a \in \mathbb{N}$ as follows:

$$cum_a((n_b)_{b\in\mathbb{N}}) = \begin{cases} 0 & \text{if } n_b = 0 \text{ for all } b \geq a, \\ (n_a+1)\cdot 2^{cum_{a+1}((n_b)_{b\in\mathbb{N}})} - 1 & \text{otherwise.} \end{cases}$$

We will be performing the cumulation in a situation when a term no longer duplicates any argument of some duplication order. Then we are sure that the duplication concerns only internals of the term, and we may shift the values to a lower order. Thus for a derivation tree D deriving $\Gamma \vdash M : (F, \tau)$ we define the following shifting function:

$$sh_{D,a}((n_b)_{b\in\mathbb{N}}) = \begin{cases} 0 & \text{if } a > F(\emptyset), \\ cum_a((n_b)_{b\in\mathbb{N}}) & \text{if } a = F(\emptyset), \\ n_a & \text{if } a < F(\emptyset). \end{cases}$$

To get used with the cum and sh operations let us observe two easy properties. First, we notice that $cum_a((n_b)_{b\in\mathbb{N}}) = n_a$ if $n_b = 0$ for all $b > a$. Second, $cum_a((sh_{D,b}((n_c)_{c\in\mathbb{N}}))_{b\in\mathbb{N}})$ (with D and F as above) equals $cum_a((n_c)_{c\in\mathbb{N}})$ if $a \leq F(\emptyset)$ and 0 otherwise.

We are now ready to define how the $loc_a(\cdot)$ values are collected. Consider a derivation tree D. For each $a \in \mathbb{N}$ we define a value $high_a(D)$, by induction on the structure of D; these values are called *higher costs*. Let D'_i for $i \in I$ denote all subtrees of D attached just above the root. We take

$$high_a(D) = sh_{D,a}\left(\left(loc_b(D) + \sum_{i\in I} high_b(D'_i)\right)_{b\in\mathbb{N}}\right).$$

See Section 5.3 for a calculation of the higher cost for example derivation trees.

Let us explain the intuitions behind the definition of higher costs. At the end we want to look at $high_0(D)$, where D derives a type for a closed term M of sort o. Notice that for such term values of all orders are shifted using our exponential cumulation to $high_0(D)$ (it holds that $high_a(D) = 0$ for all $a > 0$). This value should give an upper bound for the lower cost of the $\beta\delta$-normal form of M. It is always the case that the higher cost is greater than the lower cost. Thus to achieve our goal, we have to ensure that $high_0(D)$ is never increased during leftmost reductions. It turns out that it is not enough to look at the sum of duplication factors; the orders on which they appear start to play a role. Consider the highest n for which the order-n duplication factor is positive somewhere in D. Then, consider a node of D for a redex $(\lambda x.M)N$ such that in the subtrees corresponding to N the order-n duplication factor is zero (inside we only have duplication factors of smaller

order, and some (CON) nodes), but a binding for x of duplication order $n-1$ is duplicated in M, say once. When this redex is reduced, subtrees describing N are replicated twice, and substituted for x in M; simultaneously x disappears, so the order-n duplication factor decreases by one. Similarly, while reducing the next redex, the subtrees describing the argument can also be replicated twice, and so on. Moreover, the subtrees in each step can contain subtrees replicated in previous steps. Next, analogous analysis for order-$(n-1)$ duplication factors (whose number is already increased by order-n duplication factors) shows that each of them can multiply by two the number of duplication factors of order smaller than $n-1$ (and of (CON) nodes), and so on.

This justifies on the intuitive level the exponential character of the formula for cumulating duplication factors of different orders, but in fact this analysis cannot be formalized (in some sense it is incorrect). One problem is that the argument of a redex is not necessarily a closed term. So a positive duplication factor not only implies that the subtrees describing the argument will be replicated, but also the free variables will be used more times (and we do not know how "big" the terms are that will be substituted there). Another problem is that in general reductions may change the duplication flag of the redex (duplications factor of higher order slowly disappear during reductions), so if the redex is used as an argument, it might be necessary to change completely the rest of the derivation tree. Thus it is important in our correctness proof that we perform only leftmost reductions. Unluckily, then we lose the property that the argument of a redex does not contain duplication factors of the highest order.

However, in the formula, we do not make just one tower of exponentials at the end, but we compute some exponentials already for some subterms. Although this makes the final value $high_0(D)$ even smaller, this "eager cumulation" is essential for the proof of correctness, since otherwise $high_0(D)$ could increase during leftmost reductions. The idea behind that is as follows. Suppose that we have a term M of duplication order f. Then, by definition, M can duplicate only terms of duplication order smaller than f. So if inside M we have order-a duplication factor with $a > f$, it can be used only to duplicate subterms of M. Thus basically loc_a is cumulated recursively along the derivation tree; however when the duplication order of a subtree is f, we can forget about its duplication factors in $high_a$ for $a > f$—they will only be applied to loc_f contained inside this subtree, so we can predict their result in $high_f$.

It is an important detail that we take into account the duplication order instead of the standard order. In the conference version of this paper with loc_a we were computing the number of duplicated type bindings for variables of order $a-1$ (instead of duplicated type bindings with *duplication* order $a-1$). For that reason, we obtained there a weaker result: everything worked well only when the maximal order of a subterm of all considered terms was bounded by some fixed number. The problem was that it is possible to duplicate a variable of some high order without duplicating anywhere else variables of lower orders. On the other hand, if we have a subterm of some duplication order a, then it duplicates something of duplication order $a-1$, so necessarily we have a subterm of duplication order $a-1$, and so on. In consequence, the maximal used duplication order is bounded by the lower cost.

Relation Between the Costs. We now relate our two costs of a derivation tree: the lower cost and the higher cost. Let us start by observing that the first one is always smaller.

Proposition 12
For each derivation tree D it holds that $low(D) \leq \sum_{a\in\mathbb{N}} high_a(D)$.

Proof
Easy induction on the structure of D. $\square$

The key point is that we also have the opposite inequality, in the sense of the domination relation.

Lemma 13
For some correction function H it holds that $high_0(D) \preceq_H low(D)$ for every derivation tree D deriving a type for a closed term of sort o.

As a first ingredient of the proof, we need to observe that the maximal duplication order of a subterm is bounded by the lower cost. We need some definitions. For each type pair (f,τ), where $\tau = \bigwedge T_1 \to \ldots \to \bigwedge T_k \to \mathbf{r}$, we define by induction:

$$mdo(f,\tau) = \max(\{f\} \cup \{mdo(g,\sigma) \mid (g,\sigma) \in \bigcup_{i\in\{1,\ldots,k\}} T_i\}),$$

$$prov(f,\tau) = \{0,\ldots,mdo(f,\tau)\} \setminus \bigcup_{i=1}^{k} \bigcup_{(g,\sigma)\in T_i} prov(g,\sigma).$$

Here, $mdo(f,\tau)$ is the maximal duplication order used in (f,τ); it is just the greatest number written anywhere inside (f,τ). The set $prov(f,\tau)$ contains all duplication orders which have to be *provided* by each term having such a type. It contains all numbers between 0 and $mdo(t,\tau)$ except those which have to be provided by arguments of the term. We have analogous definitions for derivation trees. Let D be a derivation tree for $\Gamma \vdash M : (F,\tau)$, whose subtrees located just above the root are D_i for $i \in I$. Then

$$\mathbf{mdo}(D) = \max\big(\{mdo(\max_{X\subseteq dom(\Gamma)} F(X),\tau)\} \cup \{mdo(g,\sigma) \mid (x:(g,\sigma)) \in \Gamma\} \cup \{\mathbf{mdo}(D_i) \mid i \in I\}\big).$$

$$\mathbf{prov}(D) = prov(\mathbf{mdo}(D),\tau) \setminus \bigcup_{(x:(g,\sigma))\in\Gamma} prov(g,\sigma).$$

In the formula for $\mathbf{mdo}(D)$ we include the numbers used in Γ only for convenience; ignoring them would not change anything, as each type binding from Γ is used in some leaf subtree of D.

Lemma 14
For each derivation tree D, and each $a \in \mathbf{prov}(D)$ there is a subtree D' of D with $loc_a(D') > 0$.

Proof
Induction on the structure of D. Let D be a derivation tree for $\Gamma \vdash M : (F,\tau)$. We have several cases.

Suppose that M is a constant. Then $\Gamma = \emptyset$ and $F(\emptyset) = 0$. It holds that $\mathbf{mdo}(D) = 0$, and surely $loc_0(D) > 0$.

Suppose that $M = x$ is a variable. Then $\Gamma = \{x : (F(\{x\}), \tau)\}$, and $F(\emptyset) = -\infty$. We have $\mathbf{mdo}(D) = F(\{x\})$, so $\mathbf{prov}(D) = prov(\mathbf{mdo}(D), \tau) \setminus prov(F(\{x\}), \tau) = \emptyset$, thus the thesis holds trivially.

Suppose that $M = Y\,K$. Just above the root of D we have a subtree D_Y for $\Gamma \vdash K\,(Y\,K) : (F, \tau)$. Notice that $\mathbf{mdo}(D) = \mathbf{mdo}(D_Y)$, and $\mathbf{prov}(D) = \mathbf{prov}(D_Y)$, so the thesis follows immediately from the induction assumption.

Suppose that $M = \lambda x.K$. Let $\tau = \bigwedge T \to \tau'$. Just above the root of D we have a subtree D_λ for $\Gamma \cup \{x : (g, \sigma) \mid (g, \sigma) \in T\} \vdash K : (F', \tau')$. Obviously $\mathbf{mdo}(D) = \max(\mathbf{mdo}(D_\lambda), \dots) \geq \mathbf{mdo}(D_\lambda)$. It also holds that $\mathbf{mdo}(D) \leq \mathbf{mdo}(D_\lambda)$ (and hence $\mathbf{mdo}(D) = \mathbf{mdo}(D_\lambda)$), since in the root of D we only shift some numbers from the environment to τ, and we have $F = F' \restriction_{\mathscr{P}(dom(\Gamma))}$. In consequence $\mathbf{prov}(D) = \mathbf{prov}(D_\lambda)$ (only the duplication orders provided by T are removed from $\{0, \dots, \mathbf{mdo}(D)\}$ in a different place), thus the thesis follows from the induction assumption.

Finally, suppose that $M = K\,L$. Above the root of D we have a derivation tree D_0 for $\Gamma_0 \vdash K : (F_0, \bigwedge_{i \in I}(F_i(dom(\Gamma_i)), \tau_i) \to \tau)$ and for each $i \in I$ we have a derivation tree D_i for $\Gamma_i \vdash L : (F_i, \tau_i)$. Take some $a \in \mathbf{prov}(D)$; we want to prove that there is a subtree D' of D with $loc_a(D') > 0$. Suppose first that $a \leq \mathbf{mdo}(D_0)$. If $a \in \mathbf{prov}(D_0)$, the thesis follows from the induction assumption. Otherwise $a \in prov(F_i(dom(\Gamma_i)), \tau_i)$ for some $i \in I$ (a was removed from $\{0, \dots, \mathbf{mdo}(D_0)\}$ by $prov(\cdot)$ for some of the arguments, but since $a \in \mathbf{prov}(D)$, it was necessarily the first argument). Then $a \in \mathbf{prov}(D_i)$ since $mdo(F_i(dom(\Gamma_i)), \tau_i) \leq \mathbf{mdo}(D_i)$, and again the thesis follows from the induction assumption. Next, suppose that $\mathbf{mdo}(D_0) < a \leq \mathbf{mdo}(D_i)$ for some $i \in I$. Since τ_i is used in D_0, all numbers appearing in τ_i are smaller than a (are not greater than $\mathbf{mdo}(D_0)$). It follows that $a \in \mathbf{prov}(D_i)$, and we are done again. Finally, suppose that $a > \mathbf{mdo}(D_i)$ for each $i \in \{0\} \cup I$. Since $a \leq \mathbf{mdo}(D)$, the duplication order $\mathbf{mdo}(D)$ comes from the root of D. It cannot come from the environment, since $\Gamma = \bigcup_{i \in \{0\} \cup I} \Gamma_i$. So $\mathbf{mdo}(D) = F(X)$ for some $X \subseteq dom(\Gamma)$. As $F_i(X \cap dom(\Gamma_i)) \leq \mathbf{mdo}(D_i) < \mathbf{mdo}(D)$ for each $i \in \{0\} \cup I$, recalling how F is computed in the $(@)$ rule, necessarily two type environments Γ_i, Γ_j contain the same type binding of duplication order $\mathbf{mdo}(D) - 1$. Then, on the one hand $loc_{\mathbf{mdo}(D)}(D) > 0$. On the other hand, this type binding is in some Γ_i, which causes that $\mathbf{mdo}(D_i) \geq \mathbf{mdo}(D) - 1$, so $a = \mathbf{mdo}(D)$. This finishes the proof. $\square$

Next, we show an inequality concerning numbers, that will be useful in our proof of Lemma 13, as well as later.

Lemma 15

Let $(m_a)_{a \in \mathbb{N}}$, $(m'_a)_{a \in \mathbb{N}}$, $(n_a)_{a \in \mathbb{N}}$ be sequences of natural numbers in which only finitely many elements are positive. Suppose that $cum_a((m_b)_{b \in \mathbb{N}}) \geq cum_a((m'_b)_{b \in \mathbb{N}})$ for each $a \in \mathbb{N}$. Then it is also the case that $cum_a((m_b + n_b)_{b \in \mathbb{N}}) \geq cum_a((m'_b + n_b)_{b \in \mathbb{N}})$ for each $a \in \mathbb{N}$.

Proof

To shorten the notation, in this proof we drop the lower index $b \in \mathbb{N}$ for all sequences. We prove a slightly stronger inequality, that is:

$$cum_a(m_b + n_b) \geq cum_a(m'_b + n_b) + cum_a(m_b) - cum_a(m'_b). \tag{2}$$

This inequality is proved by reversed induction. Of course it holds for big a, where all sequences are zero. Assume that (2) holds for $a+1$. Below we derive (2), where in the first inequality we use the induction assumption, in the second the assumption that $cum_{a+1}(m_b) \geq cum_{a+1}(m'_b)$, and in the third the simple fact that $cum_{a+1}(m'_b + n_b) \geq cum_{a+1}(m'_b)$:

$$\begin{aligned}
cum_a(m_b+n_b) &= (m_a+n_a+1)\cdot 2^{cum_{a+1}(m_b+n_b)} - 1 \\
&\geq (m_a+n_a+1)\cdot 2^{cum_{a+1}(m'_b+n_b)+cum_{a+1}(m_b)-cum_{a+1}(m'_b)} - 1 \\
&\geq (m_a+1)\cdot 2^{cum_{a+1}(m_b)}\cdot 2^{cum_{a+1}(m'_b+n_b)-cum_{a+1}(m'_b)} + n_a\cdot 2^{cum_{a+1}(m'_b+n_b)} - 1 \\
&= (cum_a(m'_b)+1+cum_a(m_b)-cum_a(m'_b))\cdot 2^{cum_{a+1}(m'_b+n_b)-cum_{a+1}(m'_b)} \\
&\qquad\qquad + n_a\cdot 2^{cum_{a+1}(m'_b+n_b)} - 1 \\
&\geq (m'_a+1)\cdot 2^{cum_{a+1}(m'_b)}\cdot 2^{cum_{a+1}(m'_b+n_b)-cum_{a+1}(m'_b)} + cum_a(m_b) - cum_a(m'_b) \\
&\qquad\qquad + n_a\cdot 2^{cum_{a+1}(m'_b+n_b)} - 1 \\
&= (m'_a+n_a+1)\cdot 2^{cum_{a+1}(m'_b+n_b)} - 1 + cum_a(m_b) - cum_a(m'_b) \\
&= cum_a(m'_b+n_b) + cum_a(m_b) - cum_a(m'_b). \quad \square
\end{aligned}$$

We now prove a variant of Lemma 13 suitable for induction.

Lemma 16
Let D be a derivation tree, and let $k \in \mathbb{N}$ be such that $loc_a(D') = 0$ for each $a \geq k$ and for each subtree D' of D. Then for each $a \in \mathbb{N}$ the following holds:

$$cum_a\left((high_b(D))_{b\in\mathbb{N}}\right) \leq cum_a\left((m_{low(D),k,b})_{b\in\mathbb{N}}\right), \qquad \text{where}$$
$$m_{n,k,b} = \begin{cases} 0 & \text{if } b \geq k, \\ n & \text{if } b < k. \end{cases}$$

Proof
Induction on the structure of D. Let D_i for $i \in I$ denote all subtrees of D attached just above the root. We observe that

$$cum_a\left((high_b(D))_{b\in\mathbb{N}}\right) \leq cum_a\left(\left(loc_b(D) + \sum_{i\in I} high_b(D_i)\right)_{b\in\mathbb{N}}\right). \tag{3}$$

Here, the sequence on the left side is obtained by applying a shift function to the sequence on the right side; both sides are equal for small a, and from some moment the left side is always 0.

Next, we use the induction assumption, which for each $i \in I$ says that

$$cum_a\left((high_b(D_i))_{b\in\mathbb{N}}\right) \leq cum_a\left((m_{low(D_i),k,b})_{b\in\mathbb{N}}\right).$$

Thus, using Lemma 15, we can replace $high_b(D_i)$ by $m_{low(D_i),k,b}$ on the right side of (3). Finally, because $loc_b(D) + \sum_{i\in I} low(D_i) \leq low(D)$, we obtain

$$\begin{aligned}
cum_a\left((high_b(D))_{b\in\mathbb{N}}\right) &\leq cum_a\left(\left(loc_b(D) + \sum_{i\in I} m_{low(D_i),k,b}\right)_{b\in\mathbb{N}}\right) \\
&\leq cum_a\left((m_{low(D),k,b})_{b\in\mathbb{N}}\right). \quad \square
\end{aligned}$$

Proof of Lemma 13

We will prove that $high_0(D) \preceq_H low(D)$, where $H(n) = cum_0\left((m_{n,n,b})_{b\in\mathbb{N}}\right)$. Let D be a derivation tree deriving a type for a term of sort o.

Observe first that $loc_a(D') = 0$ for each $a \geq low(D)$ and for each subtree D' of D. Indeed, suppose that $loc_a(D_a) > 0$ for some $a \geq low(D) > 0$ and some subtree D_a of D (for $low(D) = 0$ the thesis is obvious). Then in the root of D_a we duplicate (in particular, we have) a type binding of duplication order $a-1$, so $\mathbf{mdo}(D) \geq a-1$, and thus $\{0,\ldots,a-1\} \subseteq \mathbf{prov}(D)$. Lemma 14 implies that also for each $b < a$ there is a subtree D_b of D with $loc_b(D_b) > 0$. But the lower cost equals the sum of $loc_b(D')$ over all b and all subtrees D' of D, so $low(D) \geq \sum_{b=0}^{a} loc_b(D_b) \geq a+1 > low(D)$, a contradiction.

In order to conclude that $high_0(D) \leq H(low(D))$ we just apply Lemma 16. □

Costs During Head Reductions. It is time to observe how the lower and higher costs of a derivation tree behave during a single leftmost reduction: the lower cost cannot decrease, and the higher cost cannot increase.

Lemma 17

Let K and L be closed terms such that L is obtained from K in a single leftmost reduction, and K does not begin with a lambda (is not of the form $\lambda x.K'$). Suppose that we can derive $\vdash L : (F,\tau)$ with a derivation tree D. Then we can derive $\vdash K : (F',\tau)$ with a derivation tree D' such that $F'(\emptyset) \geq F(\emptyset)$ and $low(D') \leq low(D)$ and $cum_a\left((high_b(D'))_{b\in\mathbb{N}}\right) \geq cum_a\left((high_b(D))_{b\in\mathbb{N}}\right)$ for each $a \in \mathbb{N}$.

We concentrate on the case of β-reduction, as the case of δ-reduction is very easy. The proof is tedious, but rather straightforward: it suffices to perform appropriate surgery on the derivation tree, and then analyze how the costs change. Let us first observe the following simple fact.

Proposition 18

Let D be a derivation tree for a type judgement with derivation flag F. Then $high_a(D) = 0$ for all $a > F(\emptyset)$. If $F(\emptyset) \geq 0$, then additionally $low(D) \geq 1$.

Proof

The part saying that $high_a(D) = 0$ for all $a > F(\emptyset)$ is trivial, since $high_a(D)$ is a result of $sh_{D,a}$, which by definition is 0 for all $a > F(\emptyset)$. We prove the second part by induction on the structure of D. For most rules in the root of D the proof is trivial, only the case of application is interesting. Suppose that $F(\emptyset) \geq 0$. Let D_i for $i \in \{0\} \cup I$ be the subtrees just above the root of D, and let F_i be the duplication flags used in their roots. Looking at formula (1) we see that either $F_i(\emptyset) \geq F(\emptyset)$ for some $i \in \{0\} \cup I$, or some binding with a nonnegative duplication flag appears in type environments in the roots of at least two subtrees D_i. In the former case the induction assumption implies that $low(D_i) \geq 1$ which implies $low(D) \geq 1$. In the latter case the duplication factor is positive, so also $low(D) \geq 1$. □

Next, let us see that rules (Y) and (λ) does not change costs.

Proposition 19
Let D be a derivation tree using either the (Y) rule or the (λ) rule in its root, and let D' be the subtree of D which starts just above the root. Then $low(D) = low(D')$ and $high_a(D) = high_a(D')$ for each $a \in \mathbb{N}$.

Proof
The part about lower costs is obvious. For the other part, we notice that the duplication flag F from the root of D and the duplication flag F' from the root of D' satisfy $F(\emptyset) = F'(\emptyset)$. It holds that $high_a(D) = sh_{D,a}((high_b(D'))_{b\in\mathbb{N}})$, and $high_b(D') = 0$ for all $b > F'(\emptyset) = F(\emptyset)$. In consequence the shift $sh_{D,a}$ does not change anything, and we just have $high_a(D) = high_a(D')$. $\square$

We also need an auxiliary lemma concerning sequences of natural numbers.

Lemma 20
Let $(m_a)_{a\in\mathbb{N}}$, $(n_a)_{a\in\mathbb{N}}$, $(k_a)_{a\in\mathbb{N}}$ be sequences of natural numbers in which only finitely many elements are positive. Suppose that for each $a \in \mathbb{N}$ it holds that $n_a = (1+\sum_{b>a} k_b)\cdot(m_a - k_a)$. Then $cum_a((m_b)_{b\in\mathbb{N}}) \geq cum_a((n_b)_{b\in\mathbb{N}})$ for each $a \in \mathbb{N}$.

Proof
Let us denote $x_a = cum_a((m_b)_{b\in\mathbb{N}})$ and $y_a = cum_a((n_b)_{b\in\mathbb{N}})$. We prove by reversed induction that $x_a \geq y_a + \sum_{b\geq a} k_b$. For very big a all numbers are 0 and the inequality holds. Suppose that $x_{a+1} \geq y_{a+1} + \sum_{b>a} k_b$. Observing that $2^n \geq n+1$ for $n \in \mathbb{N}$, we obtain

$$\begin{aligned} x_a &= (m_a+1)\cdot 2^{x_{a+1}} - 1 \geq (m_a+1)\cdot 2^{y_{a+1}+\sum_{b>a} k_b} - 1 \\ &\geq (m_a - k_a + k_a + 1)(1+\sum_{b>a} k_b)\cdot 2^{y_{a+1}} - 1 \\ &\geq (m_a - k_a)(1+\sum_{b>a} k_b)\cdot 2^{y_{a+1}} + k_a + (\sum_{b>a} k_b) - 1 = y_a + \sum_{b\geq a} k_b. \quad \square \end{aligned}$$

In order to prove Lemma 17, we have to analyze what happens during a substitution.

Lemma 21
Suppose that we can derive $\Gamma \vdash M[N/x] : (F,\tau)$ with a derivation tree D, where N is closed. Then there exist F', a set of types S, and G_σ for each $\sigma \in S$ such that

1. we can derive $\Gamma \cup \{x : (G_\sigma(\emptyset), \sigma) \mid \sigma \in S\} \vdash M : (F', \tau)$ with a derivation tree D', and $\vdash N : (G_\sigma, \sigma)$ for each $\sigma \in S$ with a derivation tree C_σ, and
2. $F(X) = F'(X \cup \bigcup_{\sigma\in S}\{x\})$ for each $X \subseteq dom(\Gamma)$, and[3]
3. $\max(\{F'(\emptyset)\} \cup \{G_\sigma(\emptyset) \mid \sigma \in S\}) \geq F(\emptyset)$, and
4. for each $a \in \mathbb{N}$ the following holds:

$$low(D') + \sum_{\sigma\in S} low(C_\sigma) \leq low(D), \qquad \text{and} \tag{4}$$

$$cum_a\left(\left(high_b(D') + \sum_{\sigma\in S} high_b(C_\sigma)\right)_{b\in\mathbb{N}}\right) \geq cum_a((high_b(D))_{b\in\mathbb{N}}). \tag{5}$$

[3] The set $\bigcup_{\sigma\in S}\{x\}$ is empty if S is empty; otherwise this is $\{x\}$.

Proof

We prove the lemma by induction on the structure of the derivation tree D. If x is not free in M, we have $M[N/x] = M$, so we can take $S = \emptyset$, and $F' = F$, and $D' = D$; then points 1-4 hold trivially (in the inequalities both sides are equal). In all the cases below we assume that x is free in M.

Suppose that $M = x$. Then we take $S = \{\tau\}$, and $G_\tau = F$, and $F'(\emptyset) = -\infty$ and $F'(\{x\}) = F(\emptyset)$. Since $M[N/x] = N$ and N is closed, the original tree D derives $\vdash N : (G_\tau, \tau)$. On the other hand, $\Gamma \cup \{x : (G_\sigma(\emptyset), \sigma) \mid \sigma \in S\} \vdash M : (F', \tau)$ is derived by simply using the (VAR) rule; the higher cost of this derivation tree is 0 at each order. Again, points 2-4 follow immediately, where in all inequalities both sides are equal.

Suppose that $M = Y\,K$. In the root of the derivation tree we use the (Y) rule, above which we have a derivation tree for $\Gamma \vdash (K\,(Y\,K))[N/x] : (F, \tau)$. Using the induction assumption for this subtree, we obtain almost everything as expected, we have only to apply the (Y) rule again to the obtained derivation tree for $\Gamma \cup \{x : (G_\sigma(\emptyset), \sigma) \mid \sigma \in S\} \vdash K\,(Y\,K) : (F', \tau)$, which gives us a derivation tree D' for $\Gamma \cup \{x : (G_\sigma(\emptyset), \sigma) \mid \sigma \in S\} \vdash M : (F', \tau)$. Notice that after appending the (Y) rule the duplication flag and the costs stay the same (Proposition 19), so points 2-4 of the thesis follow immediately from the induction assumption.

The case $M = \lambda y.K$ is very similar. W.l.o.g. we can assume that $y \neq x$. Denote $\tau = \bigwedge_{i \in I}(g_i, \rho_i) \to \tau'$. Our derivation tree starts by the (λ) rule, above which a derivation tree D_λ derives $\Gamma \cup \{y : (g_i, \rho_i) \mid i \in I\} \vdash K[N/x] : (F_\lambda, \tau')$. We use the induction assumption for D_λ, obtaining a derivation tree D'_λ for $\Gamma \cup \{y : (g_i, \rho_i) \mid i \in I\} \cup \{x : (G_\sigma(\emptyset), \sigma) \mid \sigma \in S\} \vdash K : (F'_\lambda, \tau')$ and derivation trees C_σ for $\vdash N : (G_\sigma, \sigma)$ for each $\sigma \in S$. By applying the (λ) rule to D'_λ, we obtain a derivation tree D' for $\Gamma \cup \{x : (G_\sigma(\emptyset), \sigma) \mid \sigma \in S\} \vdash M : (F', \tau)$ where $F' = F'_\lambda \restriction_{\mathscr{P}(dom(\Gamma) \cup \{x\})}$. Using the induction assumption, for each $X \subseteq dom(\Gamma)$ we obtain $F(X) = F_\lambda(X) = F'_\lambda(X \cup \{x\}) = F'(X \cup \{x\})$, as well as

$$\max(\{F'(\emptyset)\} \cup \{G_\sigma(\emptyset) \mid \sigma \in S\}) = \max(\{F'_\lambda(\emptyset)\} \cup \{G_\sigma(\emptyset) \mid \sigma \in S\}) \geq F_\lambda(\emptyset) = F(\emptyset).$$

We also obtain point 4, since the costs does not change after appending the (λ) rule (Proposition 19).

Finally, we have the most complicated case $M = K\,L$. Here, above the $(@)$ rule in the derivation tree D we have a subtree D_0 for $\Gamma_0 \vdash K[N/x] : (F_0, \bigwedge_{i \in I}(F_i(dom(\Gamma_i)), \rho_i) \to \tau)$ and subtrees D_i for $\Gamma_i \vdash L[N/x] : (F_i, \rho_i)$ for each $i \in I$ (where, by convention, $0 \notin I$). We use the induction assumption for all these subtrees. We obtain: sets S_i for $i \in \{0\} \cup I$, a derivation tree D'_0 for $\Gamma_0 \cup \{x : (G_\sigma(\emptyset), \sigma) \mid \sigma \in S_0\} \vdash K : (F'_0, \bigwedge_{i \in I}(F_i(dom(\Gamma_i)), \rho_i) \to \tau)$, and derivation trees D'_i for $\Gamma_i \cup \{x : (G_\sigma(\emptyset), \sigma) \mid \sigma \in S_i\} \vdash L : (F'_i, \rho_i)$ for each $i \in I$. Moreover, for each $\sigma \in \bigcup_{i \in \{0\} \cup I} S_i$ we obtain a derivation tree C_σ for $\vdash N : (G_\sigma, \sigma)$. Potentially the obtained derivation trees C_σ (and duplication flags G_σ) could depend not only on σ, but also on i. However Lemma 7 ensures that for each fixed σ there is at most one derivation tree for a type judgement of the form $\vdash N : (G_\sigma, \sigma)$, so indeed the derivation trees coming from different calls to the induction assumption are identical.

We take $S = \bigcup_{i \in \{0\} \cup I} S_i$. The induction assumption (point 2) ensures that $F_i(dom(\Gamma_i)) = F'_i(dom(\Gamma_i) \cup \bigcup_{\sigma \in S_i}\{x\})$, thus we can apply the $(@)$ rule to our derivation trees D'_i (where $i \in \{0\} \cup I$). We obtain a derivation tree D' for $\Gamma \cup \{x : (G_\sigma(\emptyset), \sigma) \mid \sigma \in S\} \vdash M : (F', \tau)$, for appropriate F'.

In order to prove point 2, take a set $X \subseteq dom(\Gamma)$. Let us compare formula (1) for $F(X)$ with analogous formula for $F'(X \cup \{x\})$. The first set $\{F_i(X \cap dom(\Gamma_i)) \mid i \in \{0\} \cup I\}$ is replaced by $\{F'_i((X \cup \{x\}) \cap dom(\Gamma_i \cup \{x : (G_\sigma(\emptyset), \sigma) \mid \sigma \in S_i\})) \mid i \in \{0\} \cup I\}$, but both sets are identical since $F_i(X \cap dom(\Gamma_i)) = F'_i((X \cap dom(\Gamma_i)) \cup \bigcup_{\sigma \in S_i} \{x\})$ by the induction assumption. Next, in the formula for $F(X)$ we have a set of those n for which some binding $y : (n-1, \rho)$ with $y \notin X$ belongs to at least two type environments Γ_i. For $F'(X \cup \{x\})$ the type environments contain additionally bindings for the variable x. These bindings are not taken into account since $x \in X \cup \{x\}$, so this set also remains unchanged. The situation with the set $\{n \mid \exists y \notin X. \exists \sigma. (y : (n-1, \rho)) \in \Gamma\}$ is the same. The last set $\{n \mid \exists \rho. (n-1, \rho) \in \bigcup_{i \in \{1,\ldots,k\}} T_i\}$ also remains unchanged, since in both D and D' we have the same type τ.

Next, we prove point 3. Again, we compare how $F(\emptyset)$ and $F'(\emptyset)$ and computed by formula (1). Denote the first maximum in this formula for $F(\emptyset)$ by n_1, the second maximum by n_2, and similarly n'_1, n'_2 for $F'(\emptyset)$. To compute n_1 we take into account all $F_i(\emptyset)$ and all numbers n such that some binding with duplication order $n-1$ appears in at least two environments Γ_i. To compute n'_1 we take into account all $F'_i(\emptyset)$, all numbers n as above, and also some other numbers referring to bindings of the variable x. From the induction assumption we have $\max(\{F'_i(\emptyset)\} \cup \{G_\sigma(\emptyset) \mid \sigma \in S_i\}) \geq F_i(\emptyset)$ for each $i \in I$, so we obtain $\max(\{n'_1\} \cup \{G_\sigma(\emptyset) \mid \sigma \in S\}) \geq n_1$. For the second maximum we simply have $n'_2 \geq n_2$, since the roots of D and of D' use the same type τ, and all bindings from Γ are also present in $\Gamma \cup \{x : (F_\sigma(\emptyset), \sigma) \mid \sigma \in S\}$. Thus indeed $\max(\{F'(\emptyset)\} \cup \{G_\sigma(\emptyset) \mid \sigma \in S\}) \geq F(\emptyset)$.

It remains to prove inequalities (4) and (5). On the left side of (4) we have

$$low(D') + \sum_{\sigma \in S} low(C_\sigma) = \left(\sum_{i \in \{0\} \cup I} low(D'_i) \right) + \left(\sum_{a \in \mathbb{N}} loc_a(D') \right) + \left(\sum_{\sigma \in S} low(C_\sigma) \right).$$

The type environment of D' is split into type environments of D'_i almost in the same way as the type environment of D is split into type environments of D_i, only in D' we additionally have bindings for the variable x, thus

$$\begin{aligned} \sum_{a \in \mathbb{N}} loc_a(D') &= \left(\sum_{a \in \mathbb{N}} loc_a(D) \right) + \left(\sum_{i \in \{0\} \cup I} |\{\sigma \in S_i \mid G_\sigma(\emptyset) \geq 0\}| \right) \\ &\qquad - |\{\sigma \in S \mid G_\sigma(\emptyset) \geq 0\}| \\ &\leq \left(\sum_{a \in \mathbb{N}} loc_a(D) \right) + \left(\sum_{i \in \{0\} \cup I} \sum_{\sigma \in S_i} low(C_\sigma) \right) - \left(\sum_{\sigma \in S} low(C_\sigma) \right). \end{aligned}$$

The last inequality holds because $G_\sigma(\emptyset) \geq 0$ implies $low(C_\sigma) \geq 1$ (Proposition 18). In consequence we obtain

$$low(D') + \sum_{\sigma \in S} low(C_\sigma) \leq \left(\sum_{i \in \{0\} \cup I} \left(low(D'_i) + \sum_{\sigma \in S_i} low(C_\sigma) \right) \right) + \left(\sum_{a \in \mathbb{N}} loc_a(D) \right).$$

From the induction assumption we know that $low(D'_i) + \sum_{\sigma \in S_i} low(C_\sigma) \leq low(D_i)$ for each $i \in \{0\} \cup I$. It follows that

$$low(D') + \sum_{\sigma \in S} low(C_\sigma) \leq \left(\sum_{i \in \{0\} \cup I} low(D_i) \right) + \left(\sum_{a \in \mathbb{N}} loc_a(D) \right) = low(D).$$

Before proving (5), let us observe that we always fall into one of the following two cases:

1. $F'(\emptyset) \geq F(\emptyset)$ and $F'(\emptyset) > G_\sigma(\emptyset)$ for each $\sigma \in S$, or
2. $F'(\emptyset) \geq F'_i(\emptyset)$ for each $i \in \{0\} \cup I$ and $loc_b(D') = 0$ for each $b > F'(\emptyset)$.

This dichotomy comes from formula (1) defining $F'(\emptyset)$: either $F'(\emptyset)$ equals the first maximum in this formula, or the second. The first maximum takes into account all $F'_i(\emptyset)$ and all b such that $loc_b(D') > 0$. Thus if $F'(X)$ equals that maximum, we have case 2. On the other hand, the second maximum takes into account in particular all numbers $G_\sigma(\emptyset) + 1$, since in the type environment in the root of D' there are bindings $x : (G_\sigma(\emptyset), \sigma)$. Thus if $F'(X)$ equals that maximum, we have $F'(\emptyset) > G_\sigma(\emptyset)$ for each $\sigma \in S$. The part $F'(\emptyset) \geq F(\emptyset)$ follows from the inequality $\max(\{F'(\emptyset)\} \cup \{G_\sigma(\emptyset) \mid \sigma \in S\}) \geq F(\emptyset)$ proved in point 3.

We are now ready to prove inequality (5). Denote its left side by L_a. By definition we have $high_a(D') = sh_{D',a}\big((loc_b(D') + \sum_{i\in\{0\}\cup I} high_b(D'_i))_{b\in\mathbb{N}}\big)$. We distinguish the two cases mentioned above. Suppose first that $F'(\emptyset) \geq F(\emptyset)$ and $F'(\emptyset) > G_\sigma(\emptyset)$ for each $\sigma \in S$. Then for $a > F'(\emptyset) \geq F(\emptyset)$ both sides of (5) are zero. For $a = F'(\emptyset)$ it holds that $high_a(D') = cum_a\big((loc_b(D') + \sum_{i\in\{0\}\cup I} high_b(D'_i))_{b\in\mathbb{N}}\big)$, and $high_b(D') = 0$ for each $b > a$; moreover $high_b(C_\sigma) = 0$ for each $b \geq a$ and each $\sigma \in S$. Thus

$$L_a = cum_a\left(\Big(loc_b(D') + \big(\sum_{i\in\{0\}\cup I} high_b(D'_i)\big) + \big(\sum_{\sigma\in S} high_b(C_\sigma)\big)\Big)_{b\in\mathbb{N}}\right). \tag{6}$$

For $a < F'(\emptyset)$ the shift in the formula for $high_a(D')$ does not shift anything, so again (6) holds. The opposite possibility is that $F'(\emptyset) \geq F'_i(\emptyset)$ for each $i \in \{0\} \cup I$ and $loc_b(D') = 0$ for each $b > F'(\emptyset)$. Then also $high_b(D'_i) = 0$ for each $b > F'(\emptyset) \geq F'_i(\emptyset)$ and each $i \in \{0\} \cup I$. Thus the shift in the formula for $high_a(D')$ does not shift anything (it shifts zeroes). As a consequence (6) holds for each $a \in \mathbb{N}$.

For each $a \in \mathbb{N}$ let us denote $k_a = (\sum_{i\in\{0\}\cup I} |\{\sigma \in S_i \mid G_\sigma(\emptyset) = a-1\}|) - |\{\sigma \in S \mid G_\sigma(\emptyset) = a-1\}|$. We see that $loc_a(D') = loc_a(D) + k_a$. For each $a \in \mathbb{N}$ the following holds:

$$\begin{aligned}
&\left(1 + \sum_{b>a} k_b\right) \cdot \left(loc_a(D') + \left(\sum_{i\in\{0\}\cup I} high_a(D'_i)\right) + \left(\sum_{\sigma\in S} high_a(C_\sigma)\right) - k_a\right) \\
&\qquad \geq loc_a(D) + \left(\sum_{i\in\{0\}\cup I} high_a(D'_i)\right) + \left(1 + \sum_{b>a} k_b\right) \cdot \sum_{\sigma\in S} high_a(C_\sigma) \\
&\qquad \geq loc_a(D) + \sum_{i\in\{0\}\cup I} \left(high_a(D'_i) + \sum_{\sigma\in S_i} high_a(C_\sigma)\right).
\end{aligned}$$

The last inequality holds because $high_a(C_\sigma)$ may be positive only when $G_\sigma(\emptyset) + 1 > a$. We are ready to apply Lemma 20 to (6): we subtract k_a and we multiply by $1 + \sum_{b>a} k_a$. Using our last observation we obtain

$$L_a \geq cum_a\left(\Big(loc_b(D) + \sum_{i\in\{0\}\cup I} \Big(high_b(D'_i) + \sum_{\sigma\in S_i} high_b(C_\sigma)\Big)\Big)_{b\in\mathbb{N}}\right). \tag{7}$$

The induction assumption tells us that for $i \in \{0\} \cup I$,

$$cum_a\left(\left(high_b(D_i') + \sum_{\sigma \in S_i} high_b(C_\sigma)\right)_{b\in\mathbb{N}}\right) \geq cum_a\left((high_b(D_i))_{b\in\mathbb{N}}\right).$$

Using this inequality and Lemma 15 we replace $high_b(D_i') + \sum_{\sigma\in S_i} high_b(C_\sigma)$ by $high_b(D_i)$ (for each i separately) on the right side of (7), and we obtain

$$\begin{aligned} L_a &\geq cum_a\left(\left(loc_b(D) + \sum_{i\in\{0\}\cup I} high_b(D_i)\right)_{b\in\mathbb{N}}\right) \\ &\geq cum_a\left(\left(sh_{D,b}\left(\left(loc_c(D) + \sum_{i\in\{0\}\cup I} high_c(D_i)\right)_{c\in\mathbb{N}}\right)\right)_{b\in\mathbb{N}}\right) \\ &\geq cum_a\left((high_b(D))_{b\in\mathbb{N}}\right). \quad \square \end{aligned}$$

Proof of Lemma 17

We proceed by induction on the size of K. We have four cases. One possibility is that $K = (\lambda x.M)N$ and $L = M[N/x]$. Then of course we apply Lemma 21, out of which we obtain a derivation tree B for $\{x : (G_\sigma(\emptyset), \sigma) \mid \sigma \in S\} \vdash M : (G, \tau)$ and derivation trees C_σ for $\vdash N : (G_\sigma, \sigma)$ for each $\sigma \in S$. A derivation tree B_λ deriving $\vdash \lambda x.M : (G', \bigwedge_{\sigma\in S}(G_\sigma(\emptyset), \sigma) \to \tau)$ is obtained by appending the (λ) rule to B. To the trees B_λ and C_σ for $\sigma \in S$ we apply the $(@)$ rule, which gives us a derivation tree D' for $\vdash (\lambda x.M)N : (F', \tau)$. Point 3 of Lemma 21 says that $\max(\{G(\emptyset)\} \cup \{G_\sigma(\emptyset) \mid \sigma \in S\}) \geq F(\emptyset)$, and we have $G'(\emptyset) = G(\emptyset)$. Observe how $F'(\emptyset)$ is computed by formula (1). The first maximum is exactly $\max(\{G'(\emptyset)\} \cup \{G_\sigma(\emptyset) \mid \sigma \in S\})$ (since there are no bindings in the environment), so it is not smaller than $F(\emptyset)$. The second maximum is the same as in the formula for $F(\emptyset)$, since the roots of D and of D' use the same type τ. Thus $F'(\emptyset) \geq F(\emptyset)$. Let us compare the costs. For the lower cost we have $low(B_\lambda) = low(B)$, so using (4) we obtain

$$low(D') = low(B_\lambda) + \sum_{\sigma\in S} low(C_\sigma) \leq low(D).$$

For $a > F'(\emptyset) \geq F(\emptyset)$ we have $cum_a\left((high_b(D))_{b\in\mathbb{N}}\right) = 0$. As observed in Proposition 19, $high_b(B_\lambda) = high_b(B)$ for each $b \in \mathbb{N}$. For $a \leq F'(\emptyset)$ using inequality (5) we obtain

$$\begin{aligned} cum_a\left((high_b(D'))_{b\in\mathbb{N}}\right) &= cum_a\left(\left(high_b(B_\lambda) + \sum_{\sigma\in S} high_b(C_\sigma)\right)_{b\in\mathbb{N}}\right) \\ &\geq cum_a\left((high_b(D))_{b\in\mathbb{N}}\right). \end{aligned}$$

Another possibility is that $K = YM$ and $L = M(YM)$. Then to obtain D' we just append one node using the (Y) rule below the root of D. Recall that such D', when compared to D, has the same duplication flag, lower cost, and higher cost at each order.

Next, it is possible that $K = M'N$ and $L = MN$ and M is obtained from M' in a leftmost reduction. Then, above the root of D using the $(@)$ rule we have a subtree D_0 deriving $\vdash M : (F_0, \bigwedge_{i\in I}(F_i(\emptyset), \rho_i) \to \tau)$ and subtrees D_i deriving $\vdash N : (F_i, \rho_i)$ for each $i \in I$. Notice that M' does not begin with a lambda, as then the leftmost reduction should reduce the whole $M'N$. Using the induction assumption we obtain a derivation tree D_0' for $\vdash M' : (F_0', \bigwedge_{i\in I}(F_i(\emptyset), \rho_i) \to \tau)$. We apply again the $(@)$ rule to trees D_0' and D_i for $i \in I$, and we

obtain a derivation tree for $\vdash K : (F', \tau)$. Comparing formulas (1) for $F'(\emptyset)$ and for $F(\emptyset)$ we see that the only difference is that for $F'(\emptyset)$ we take $F'_0(\emptyset)$ instead of $F_0(\emptyset)$. In consequence $F'(\emptyset) \geq F(\emptyset)$, since $F'_0(\emptyset) \geq F_0(\emptyset)$ by the induction assumption. The inequality about lower costs also follows trivially from the induction assumption:

$$low(D') = low(D'_0) + \sum_{i \in I} low(D_i) \leq low(D_0) + \sum_{i \in I} low(D_i) = low(D).$$

The induction assumption says also that $cum_a\left((high_b(D'_0))_{b \in \mathbb{N}}\right) \geq cum_a\left((high_b(D_0))_{b \in \mathbb{N}}\right)$ for each $a \in \mathbb{N}$. For $a > F'(\emptyset) \geq F(\emptyset)$ we have $cum_a\left((high_b(D))_{b \in \mathbb{N}}\right) = 0$. For $a \leq F'(\emptyset)$ using the induction assumption and Lemma 15 we obtain

$$\begin{aligned} cum_a\left((high_b(D'))_{b \in \mathbb{N}}\right) &= cum_a\left(\left(high_b(D'_0) + \sum_{i \in I} high_b(D_i)\right)_{b \in \mathbb{N}}\right) \\ &\geq cum_a\left(\left(high_b(D_0) + \sum_{i \in I} high_b(D_i)\right)_{b \in \mathbb{N}}\right) \geq cum_a\left((high_b(D))_{b \in \mathbb{N}}\right). \end{aligned}$$

Finally, it is possible that $K = \mathbf{c} N_1 \ldots N_k M'$ and $L = \mathbf{c} N_1 \ldots N_k M$ and M is obtained from M' in a leftmost reduction. Recall that we only consider constants of order at most one, so necessarily M' is of order 0; in particular M' does not begin with a lambda. Above the root of D using the $(@)$ rule we have a subtree D_0 deriving $\vdash \mathbf{c} N_1 \ldots N_k : (F_0, \bigwedge_{i \in I}(F_i(\emptyset), \mathbf{r}) \to \tau)$ and subtrees D_i deriving $\vdash M : (F_i, \mathbf{r})$ for each $i \in I$ (in fact I contains exactly one element, but this is not important for us). Using the induction assumption for each $i \in I$ we obtain a derivation tree D'_i for $\vdash M' : (F'_i, \mathbf{r})$. Since the type in this tree is $\mathbf{r}$ (it takes no arguments), and there are no bindings in the type environment, it holds that $F'_i(\emptyset) \leq 0$ and $F_i(\emptyset) \leq 0$. We also know that $F'_i(\emptyset) \geq F_i(\emptyset)$. If $F_i(\emptyset) = 0$, then also $F'_i(\emptyset) = 0$. If $F_i(\emptyset) = -\infty$, then $high_a(D_i) = 0$ for each $a \in \mathbb{N}$, so $0 = low(D_i) \geq low(D'_i)$, which implies $F'_i(\emptyset) = -\infty$. Thus we can apply again the $(@)$ rule to trees D_0 and D'_i for $i \in I$; we obtain a derivation tree for $\vdash K : (F, \tau)$. The duplication flag remains unchanged (it is still F). Like previously, the inequalities about lower costs and higher costs follow from the induction assumption (for the higher cost we use Lemma 15). $\square$

Conclusion of the Proof. We already have all ingredients needed to finish the proof of Lemma 8. As H we take the correction function defined by Lemma 13.

Let now M, N be closed terms that are of sort o and contain only constants of order at most 1, where $M \to^*_{\beta\delta} N$ and N is in $\beta\delta$-normal form. Lemma 9 implies that $(0, \mathbf{r}) \in types(N)$, so we can derive $\vdash N : (F_0, \mathbf{r})$ by a derivation tree D, where $F_0(\emptyset) = 0$. Due to Fact 1, there is a sequence of leftmost reductions leading from M to N. Starting from the end, we consecutively apply Lemma 17 to these reductions, and we obtain a derivation tree D' for $\vdash M : (F', \mathbf{r})$, where $F'(\emptyset) \geq F_0(\emptyset)$, and $low(D') \leq low(D)$, and $cum_a\left((high_b(D'))_{b \in \mathbb{N}}\right) \geq cum_a\left((high_b(D))_{b \in \mathbb{N}}\right)$ for each $a \in \mathbb{N}$. Since the type in the root of M is $\mathbf{r}$ (it takes no arguments), and there are no bindings in the type environment, it holds that $F'(\emptyset) \leq 0$, so in fact $F'(\emptyset) = 0$. It follows that $(0, \mathbf{r}) \in types(M)$.

Because $F'(0) = 0 = F_0(0)$, our inequality about higher costs boils down to $high_0(D') \geq high_0(D)$. Recall that $\mathbf{low}(M, (0, \mathbf{r})) = low(D')$ and $\mathbf{low}(N, (0, \mathbf{r})) = low(D)$. Thus using

Proposition 12 and Lemma 13, we obtain the required inequalities:

$$low(D') \leq low(D) \leq high_0(D) \leq high_0(D') \leq H(low(D')).$$

5 A Longer Example

In this section we present a longer example illustrating notions introduced in the former sections, such as derivation trees, the lower cost and the higher cost.

5.1 Derivation Trees

We begin with few examples of derivation trees. To shorten the notation, a function which maps S_i to n_i for each $i \in \{1,\dots,k\}$ is denoted by $[S_1{\mapsto}n_1,\dots,S_k{\mapsto}n_k]$. We denote by b_x the binding $x:(0,\mathbf{r})$, by $b_y^{-\infty}$ the binding $y:(-\infty,(0,\mathbf{r}){\to}\mathbf{r})$, and by b_y^1 the binding $y:(1,(0,\mathbf{r}){\to}\mathbf{r})$. We also denote $F_x = [\emptyset{\mapsto}-\infty,\ \{x\}{\mapsto}0]$, and $F_y^{-\infty} = [\emptyset{\mapsto}-\infty,\ \{y\}{\mapsto}-\infty]$, and $F_y^1 = [\emptyset{\mapsto}-\infty,\ \{y\}{\mapsto}1]$. The first tree, D_1, derives a type judgement for the term $\lambda y.\lambda x.y\,(yx)$.

$$\dfrac{\dfrac{\dfrac{b_y^1 \vdash y:(F_y^1,(0,\mathbf{r}){\to}\mathbf{r}) \qquad \dfrac{b_y^1 \vdash y:(F_y^1,(0,\mathbf{r}){\to}\mathbf{r}) \qquad b_x \vdash x:(F_x,\mathbf{r})}{b_x,b_y^1 \vdash yx:([\emptyset{\mapsto}-\infty,\ \{x\}{\mapsto}0,\ \{y\}{\mapsto}1,\ \{x,y\}{\mapsto}0],\mathbf{r})}\,(@)}{b_x,b_y^1 \vdash y\,(yx):([\emptyset{\mapsto}2,\ \{x\}{\mapsto}2,\ \{y\}{\mapsto}1,\ \{x,y\}{\mapsto}0],\mathbf{r})}\,(@)}{b_y^1 \vdash \lambda x.y\,(yx):([\emptyset{\mapsto}2,\ \{y\}{\mapsto}1],(0,\mathbf{r}){\to}\mathbf{r})}\,(\lambda)}{\vdash \lambda y.\lambda x.y\,(yx):([\emptyset{\mapsto}2],(1,(0,\mathbf{r}){\to}\mathbf{r}){\to}(0,\mathbf{r}){\to}\mathbf{r})}\,(\lambda)$$

The next tree, D_2, derives another type judgement for the same term.

$$\dfrac{\dfrac{\dfrac{b_y^{-\infty} \vdash y:(F_y^{-\infty},(0,\mathbf{r}){\to}\mathbf{r}) \qquad \dfrac{b_y^{-\infty} \vdash y:(F_y^{-\infty},(0,\mathbf{r}){\to}\mathbf{r}) \qquad b_x \vdash x:(F_x,\mathbf{r})}{b_x,b_y^{-\infty} \vdash yx:([\emptyset{\mapsto}-\infty,\ \{x\}{\mapsto}0,\ \{y\}{\mapsto}-\infty,\ \{x,y\}{\mapsto}0],\mathbf{r})}\,(@)}{b_x,b_y^{-\infty} \vdash y\,(yx):([\emptyset{\mapsto}-\infty,\ \{x\}{\mapsto}0,\ \{y\}{\mapsto}-\infty,\ \{x,y\}{\mapsto}0],\mathbf{r})}\,(@)}{b_y^{-\infty} \vdash \lambda x.y\,(yx):([\emptyset{\mapsto}-\infty,\ \{y\}{\mapsto}-\infty],(0,\mathbf{r}){\to}\mathbf{r})}\,(\lambda)}{\vdash \lambda y.\lambda x.y\,(yx):([\emptyset{\mapsto}-\infty],(-\infty,(0,\mathbf{r}){\to}\mathbf{r}){\to}(0,\mathbf{r}){\to}\mathbf{r})}\,(\lambda)$$

Next, consider the term $M_3 = \lambda z.\lambda t.z\,(\lambda x.x)\,(z\,(\lambda x.\mathbf{c}\,x\,x)\,t)$, where $\mathbf{c}$ is a constant of sort $o{\to}o{\to}o$, and types $\tau_1 = (1,(0,\mathbf{r}){\to}\mathbf{r}){\to}(0,\mathbf{r}){\to}\mathbf{r}$ and $\tau_2 = (-\infty,(0,\mathbf{r}){\to}\mathbf{r}){\to}(0,\mathbf{r}){\to}\mathbf{r}$ (these are the types in the conclusions of D_1 and D_2). Then there exists a derivation tree D_3 for the type judgement $\vdash M_3 : (0,(\tau_1 \wedge \tau_2){\to}(0,\mathbf{r}){\to}\mathbf{r})$. We can merge all these trees into D_5 as follows.

$$\dfrac{\dfrac{D_3 \qquad D_1 \qquad D_2}{\vdash M_3\,(\lambda y.\lambda x.y\,(yx)):([\emptyset{\mapsto}1],(0,\mathbf{r}){\to}\mathbf{r})}\,(@) \qquad \vdash \mathbf{d}:([\emptyset{\mapsto}0],\mathbf{r})}{\vdash M_3\,(\lambda y.\lambda x.y\,(yx))\,\mathbf{d}:([\emptyset{\mapsto}0],\mathbf{r})}\,(@)$$

For further use, denote its subtree ending at the first $(@)$ rule by D_4.

5.2 Lower Cost

Let us see what is the lower cost of the derivation trees from Section 5.1. In D_1 the binding b_y^1 is duplicated in the lower $(@)$ rule, so we have $low(D_1) = 1$. In D_2 the binding $b_y^{-\infty}$ is

also duplicated, but it has duplication order $-\infty$, so this duplication is not included to the duplication factor and hence $low(D_2) = 0$. Let us analyze the tree D_3, which is not written explicitly. There will be a node describing the subterm $\mathbf{c}xx$ in which a binding for x will be duplicated. There will be also a (CON) node for the constant $\mathbf{c}$. Notice that bindings for variable z are not duplicated, since the two appearances of z use two different bindings. Thus $low(D_3) = 2$. Altogether we obtain $low(D_4) = 3$ and $low(D_5) = 4$ (because of another (CON) node).

Notice that the $\beta\delta$-normal form of the term described by D_5 is $\mathbf{c}\,(\mathbf{c}\,\mathbf{d}\,\mathbf{d})\,(\mathbf{c}\,\mathbf{d}\,\mathbf{d})$. It contains 7 constants, so the lower cost of the derivation tree for this term will be 7.

5.3 Higher Cost

Next, let us analyze the higher cost of the derivation trees from Section 5.1. We see that the duplication of b^1_y in D_1 is on duplication order 2 (b^1_y has duplication order 1), that is $high_2(D_1) = 1$ and $high_a(D_1) = 0$ for $a \neq 2$. In D_2 all higher costs are 0. In D_3 the duplication of x is on duplication order 1, and we have a (CON) node for $\mathbf{c}$, so $high_0(D_3) = high_1(D_3) = 1$ and $high_a(D_3) = 0$ for $a \geq 2$. Notice that there were no nontrivial shifts (cumulations) of higher costs in D_1 and in D_3, since always we had either an argument or a type binding of high-enough duplication order. But the conclusion of D_4 has empty type environment and takes only an argument of duplication order 0, thus higher costs from order 2 will be now shifted to order 1: $high_0(D_4) = 1$, and $high_1(D_4) = (1+1)\cdot 2^1 - 1 = 3$, and $high_a(D_4) = 0$ for $a \geq 2$. Finally, in D_5 we add 1 to the higher cost at order 0 because of the constant $\mathbf{d}$, and we shift everything to order 0: $high_0(D_3) = (2+1)\cdot 2^3 - 1 = 23$. Surely this is greater than 7, the lower cost of the $\beta\delta$-normal form of $M_3\,(\lambda y.\lambda x.y\,(yx))\,\mathbf{d}$.

6 Consequences

In this section we present two example consequences of Theorem 10. We define $dim(\alpha)$ to be the number of possible sets $types(M)$ over all closed terms M of sort $\alpha\to\alpha_{\mathbb{N}}$. As already observed, this number is finite (and is not greater than $|\mathscr{P}_{cons}(\{-\infty, 0, \ldots, ord(\alpha\to\alpha_{\mathbb{N}})\} \times \mathscr{T}^{\alpha\to\alpha_{\mathbb{N}}})|$). In particular $dim(\alpha)$ assigns some natural number to each sort α.

Recall that for a term M of sort $\alpha_{\mathbb{N}}$ representing a natural number in a numeral system *val* we write $\mathbf{val}(M)$ for this number. We obtain the following theorem.

Theorem 22
Let *val* be a numeral system using terms of sort $\alpha_{\mathbb{N}}$, let α be a sort, let $\mathscr{F}$ ("functions") be a set of closed terms of sort $\alpha\to\alpha_{\mathbb{N}}$, and let $\mathscr{A}$ ("arguments") be a set of closed terms of sort α such that $val(KN)$ is defined for all $K \in \mathscr{F}$, $N \in \mathscr{A}$. We define an equivalence relation over elements of $\mathscr{F}$: we have $K \sim L$ when for each $\mathscr{M} \subseteq \mathscr{A}$ the set $\{\mathbf{val}(KN) \mid N \in \mathscr{M}\}$ is bounded if and only if the set $\{\mathbf{val}(LN) \mid N \in \mathscr{M}\}$ is bounded. Then this relation has at most $dim(\alpha)$ equivalence classes.

Proof
We will show that when $types(K) = types(L)$ then also $K \sim L$; the thesis of the theorem will follow, since we have at most $dim(\alpha)$ possible sets $types(K)$.

Thus suppose $types(K) = types(L)$, and consider a set $\mathscr{M}$ of terms of sort α such that KN and LN represent a number for each $N \in \mathscr{M}$, and such that $\{\mathbf{val}(KN) \mid N \in \mathscr{M}\}$ is bounded. Theorem 10 gives us a choice function t and a correction function H such that $\mathbf{low}(KN, t(types(KN))) \approx_H \mathbf{val}(KN)$ for each $N \in \mathscr{M}$. This implies that the set $\{\mathbf{low}(KN, t(types(KN))) \mid N \in \mathscr{M}\}$ is bounded as well. Because $types(K) = types(L)$, by compositionality of *types* we have $types(KN) = types(LN)$. Thus Theorem 10 applied to the term LN gives us the equivalence $\mathbf{val}(LN) \approx_H \mathbf{low}(LN, t(types(KN)))$. Recall the property that the vector of lower costs assigned to a composition KN is computed basing on the lower costs assigned to K and to N in a way depending only on $types(K)$ and $types(N)$: there is an element u_N and a set U_N such that $\mathbf{low}(KN, t(types(KN))) = \mathbf{low}(K, u_N) + \sum_{u \in U_N} \mathbf{low}(N, u)$ and $\mathbf{low}(LN, t(types(KN))) = \mathbf{low}(L, u_N) + \sum_{u \in U_N} \mathbf{low}(N, u)$. In consequence $\mathbf{low}(LN, t(types(KN))) = \mathbf{low}(KN, t(types(KN))) + \mathbf{low}(L, u_N) - \mathbf{low}(K, u_N)$. Because u_N comes from a finite set, the set $\{\mathbf{low}(LN, t(types(KN))) \mid N \in \mathscr{M}\}$ is bounded, so the set $\{\mathbf{val}(LN) \mid N \in \mathscr{M}\}$ is bounded as well thanks to the equivalence $\mathbf{val}(LN) \approx_H \mathbf{low}(LN, t(types(KN)))$. The opposite implication (from L to K) is completely symmetric. □

The second theorem says that in terms of type α we can represent k-tuples of natural numbers only for $k \leq dim(\alpha)$ in such a way that the numbers can be extracted out of the k-tuples using λ-terms. That is, while representing k-tuples in terms of sort α, we want to have closed terms $M_1, \ldots, M_k$, all of the same sort $\alpha \to \alpha_{\mathbb{N}}$. Then the k-tuple represented by a term N of sort α will be $(\mathbf{val}(M_1 N), \ldots, \mathbf{val}(M_k N))$. Our result is described by the following theorem.

Theorem 23
Let *val* be a numeral system using terms of sort $\alpha_{\mathbb{N}}$, and let α be a sort. Let $M_1, \ldots, M_k$ be closed terms of sort $\alpha \to \alpha_{\mathbb{N}}$, for $k > dim(\alpha)$. Let X be the set of k-tuples

$$(\mathbf{val}(M_1 N), \ldots, \mathbf{val}(M_k N))$$

over all closed terms N of sort α such that $\mathbf{val}(M_i N)$ is defined for each i. Then $X \neq \mathbb{N}^k$. Moreover, there exist at most $dim(\alpha)$ indices $i \in \{1, \ldots, k\}$ for which there exists a subset $X_i \subseteq X$ containing tuples with arbitrarily big numbers on the i-th coordinate, but such that all numbers on all other coordinates are bounded.

Proof
This is an immediate consequence of Theorem 22. Suppose that for some i there exists a set X_i as in the statement of the theorem. This means that there is a set $\mathscr{M}_i$ of terms such that the set $\{\mathbf{val}(M_i N) \mid N \in \mathscr{M}_i\}$ is unbounded, but the sets $\{\mathbf{val}(M_j N) \mid N \in \mathscr{M}_i\}$ are bounded for each $j \neq i$. Then, by definition $M_i \not\sim M_j$ for each $j \neq i$. Since we only have $dim(\alpha)$ equivalence classes of $\sim$, we can have at most $dim(\alpha)$ such indices i. In particular it holds that $X \neq \mathbb{N}^k$. □

Polymorphic Types. We notice that the results given in our paper for simply-typed λ-calculus do not extend to polymorphic λ-calculus, already with the Hindley-Milner type system. We will show that we can represent tuples of arbitrarily many natural numbers in terms of the polymorphic sort $\forall\beta.(\beta \to \alpha_{\mathbb{N}}) \to (\alpha_{\mathbb{N}} \to \beta \to \beta) \to \beta \to \alpha_{\mathbb{N}}$. This would contradict an appropriate formulation of Theorem 23 for polymorphic λ-calculus.

For a number n, let $[\![n]\!]$ be some term representing n. We represent a k-tuple $(n_1,\ldots,n_k)$ in the term $list_{n_1,\ldots,n_k} = \lambda e.\lambda f.\lambda z.e(f\,[\![n_1]\!]\,(f\,[\![n_2]\!]\,(\ldots\,(f\,[\![n_k]\!]\,z)\ldots)))$, having the polymorphic sort as declared above.

Now we show how to extract particular numbers from such a tuple. We need the term $tup_{n_1,\ldots,n_k} = \lambda f.f\,[\![n_1]\!]\ldots[\![n_k]\!]$ (whose sort depends on k), that is usually used to represent a k-tuple $(n_1,\ldots,n_k)$. Then the term $extr_{i,k} = \lambda t.t\,(\lambda x_1.\cdots.\lambda x_k.x_i)$ extracts the number n_i, that is $extr_{i,k}\,tup_{n_1,\ldots,n_k} \to_\beta^* [\![n_i]\!]$. We also need a term that shifts numbers in a k-tuple, ignores the last number, and places a new one on the first position: $shift_k = \lambda t.\lambda n.\lambda f.f\,n\,(extr_{1,k}\,t)\ldots(extr_{k-1,k}\,t)$. To extract the i-th element from our list we use the term $M_i = \lambda x.x\,extr_{i,i}\,shift_i\,tup_{1,\ldots,i}$. It is easy to see that for $i \le k$ we indeed have

$$\begin{aligned} M_i\,list_{n_1,\ldots,n_k} &\to_\beta^* extr_{i,i}\,(shift_i\,[\![n_1]\!]\,(shift_i\,[\![n_2]\!]\,(\ldots\,(shift_i\,[\![n_k]\!]\,tup_{1,\ldots,i})\ldots))) \\ &\to_\beta^* extr_{i,i}\,tup_{n_1,\ldots,n_i} \to_\beta^* [\![n_i]\!]. \end{aligned}$$

References

Barendregt, Henk, Dekkers, Wil, & Statman, Richard. (2013). *Lambda calculus with types*. Perspectives in Logic. Cambridge University Press.

Clairambault, Pierre, & Murawski, Andrzej S. (2013). Böhm trees as higher-order recursive schemes. *Pages 91–102 of:* Seth, Anil, & Vishnoi, Nisheeth K. (eds), *IARCS annual conference on foundations of software technology and theoretical computer science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*. LIPIcs, vol. 24. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Colcombet, Thomas. (2013). Regular cost functions, part I: logic and algebra over words. *Logical methods in computer science*, **9**(3).

Kobayashi, Naoki. (2013). Pumping by typing. *Pages 398–407 of: Lics*. IEEE Computer Society.

Parys, Paweł. (2012). On the significance of the collapse operation. *Pages 521–530 of: Lics*. IEEE.

Parys, Paweł. (2014). How many numbers can a lambda-term contain? *Pages 302–318 of:* Codish, Michael, & Sumii, Eijiro (eds), *Flops*. Lecture Notes in Computer Science, vol. 8475. Springer.

Schwichtenberg, Helmut. (1976). Definierbare funktionen im lambda-kalkl mit typen. *Archiv logic grundlagenforsch*, **17**, 113–114.

Statman, Richard. (1979). The typed lambda-calculus is not elementary recursive. *Theor. comput. sci.*, **9**, 73–81.

Zaionc, Marek. (1987). Word operation definable in the typed lambda-calculus. *Theor. comput. sci.*, **52**, 1–14.