How Many Numbers Can a Lambda-term Contain?

Paweł Parys[⋆]

University of Warsaw, Warsaw, Poland
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 prove, however, 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).

1 Introduction

It is well known, that simply-typed λ-terms can be used to represent numbers, as well as some other data types (for an introduction see e.g. [1]). In particular we can represent pairs or tuples of representable data types. Notice however that the sort[1] of terms representing pairs is more complex than the sort of terms representing the elements of pairs. We prove 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. It gives an upper bound on how large tuples of natural numbers can be represented by a term of sort α.

To represent a natural number in a λ-term we use two constants: $\mathbf{0}$ of sort o, and $\mathbf{1+}$ of sort $o \to o$. We define the *value* of a closed term M of sort o as the natural number saying how many times the constant $\mathbf{1+}$ appears in the β-normal form of M. Notice that each β-normalized closed term of sort o is of the form $\mathbf{1+}\ (\mathbf{1+}\ (\dots(\mathbf{1+}\ \mathbf{0})\dots))$. Of course the number of constants $\mathbf{1+}$ used in a term may change during β-reduction; we count it in the β-normal form of a term.

[⋆] The author holds a post-doctoral position supported by Warsaw Center of Mathematics and Computer Science. Work supported by the National Science Center (decision DEC-2012/07/D/ST6/02443).

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

It is not a problem to pack arbitrarily many natural numbers into a term, so that for each list (arbitrarily long tuple) of natural numbers we obtain a different term, even of a very simple sort. We however consider the opposite direction, that is extracting numbers from terms. We do not require anything about 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. That is, while representing k-tuples in terms of sort α, we want to have closed terms $M_1, \dots, M_k$, all of the same sort $\alpha \to o$. Then the k-tuple *extracted* by $M_1, \dots, M_k$ from a closed term N (representing a k-tuple) of sort α is defined as the k-tuple of values of $M_1\ N, \dots, M_k\ N$. Our main result is described by the following theorem.

Theorem 1. *Let $M_1, \dots, M_k$ be closed terms of sort $\alpha \to o$, for $k > dim(\alpha)$. Let X be the set of all k-tuples which are extracted by $M_1, \dots, M_k$ from any closed term of sort α. Then $X \neq \mathbb{N}^k$. Moreover, there exist at most $dim(\alpha)$ indices $i \in \{1, \dots, k\}$ 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.*

In the last sentence of the theorem we say that the set X is, in some sense, really at most $dim(\alpha)$-dimensional. It follows that it is impossible to represent k-tuples in terms of sort α with $k > dim(\alpha)$ even when we allow some approximation of the numbers in tuples. The next theorem states a similar property.

Theorem 2. *Fix a sort α. We define an equivalence relation over closed terms of sort $\alpha \to o$: we have $M \sim M'$ when for each sequence $N_1, N_2, \dots$ of closed terms of sort α, the sequences of values of the terms $M\ N_1, M\ N_2, \dots$ and $M'\ N_1, M'\ N_2, \dots$ are either both bounded or both unbounded. Then this relation has at most $dim(\alpha)$ equivalence classes.*

Beside of the final result, we believe that the techniques used in the proofs are interesting on their own. First, we introduce a type system which describes, intuitively, whether a subterm adds something to the value of a term, or not. Second, we describe a closed term of any sort α by a tuple (of arity depending only on α) of natural numbers, which approximate all possible values which can be extracted from the term. This description is compositional: the tuple for MN depends only on the tuples for M and for N.

Related Work. Results in the spirit of this paper (but with significant differences) were an important part of the proof [2] that Collapsible Higher-Order Pushdown Systems generate more trees than Higher-Order Pushdown Systems without the collapse operation. However the appropriate lemmas of [2] 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 [3]. However, comparing to [3], we additionally have a productive/nonproductive flag in our types.

One may wonder why we represent natural numbers using constants **1+** and **0**, instead of using the standard representation as terms of sort $(o \to o) \to o \to o$, where the representation $[k]$ of a number k is defined by $[0] = \lambda f.\lambda x.x$ and $[k+1] = \lambda f.\lambda x.f\ ([k]\ f\ x)$. Observe, however, that a number in the "standard" representation can be easily converted to a number in our representation: the term $[k]$ **1+** **0** has value k. Since in this paper we only talk about extracting numbers from terms (we never start from representations of numbers), all our results also hold for the standard representation. We believe that thanks to distinguishing the constants **0** and **1+** from other variables, the argumentation in the paper becomes more clear.

Schwichtenberg [4] and Statman [5] show that the functions over natural numbers representable in the simply-typed λ-calculus are precisely the "extended polynomials". Notice that our results does not follow from this characterization, since they describe only first-order functions (functions $\mathbb{N}^k \to \mathbb{N}$). Similarly, Zaionc [6] characterizes the class of functions over words which are represented by closed λ-terms (for appropriate representation of words in λ-calculus).

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 $\lambda x.\mathbf{0}$). 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. In Section 4 we introduce the Krivine machine, and we define its variant which beside of terms stores type judgements, and even derivation trees for them. This machine allows us to trace how a derivation tree changes during β-reductions. Next, in Section 5, to configurations of the Krivine machine we assign some numbers, which give a lower and an upper bound on the value of the term. To obtain them, we basically count in how many places in derivation trees for type judgements something "productive" happens. Finally, in Section 6 we conclude the proof of our main theorems, and in Section 7 we give some further remarks.

Acknowledgement. We thank Igor Walukiewicz and Sylvain Salvati for a discussion on this topic.

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 $ord(\alpha \to \beta) = \max(1 + ord(\alpha), ord(\beta))$.

A *signature* is a set of typed constants, that is symbols with associated sorts. In our paper we use a signature consisting of two constants: **0** of sort o, and **1+** of sort $o \to o$.

The set of *simply-typed λ-terms* is defined inductively as follows. A constant of sort α is a term of sort α. For each sort α there is a countable set of variables $x^\alpha, y^\alpha, \dots$ 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 $\alpha \to \beta$. Finally, if M is of sort $\alpha \to \beta$ and N is of sort α then MN is a term of sort β. 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 $ord(\alpha)$.

3 Type System

In this section we define types which will be used to describe our terms. These 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 flag from $\{\mathsf{pr}, \mathsf{np}\}$ (where pr stands for productive, and np for nonproductive). The unique atomic type is denoted $\mathbf{r}$. More precisely, for each sort α we define the set $\mathcal{T}^\alpha$ of types of sort α as follows:

$$\mathcal{T}^o = \{\mathbf{r}\}, \qquad \mathcal{T}^{\alpha \to \beta} = \mathcal{P}(\{\mathsf{pr}, \mathsf{np}\} \times \mathcal{T}^\alpha) \times \mathcal{T}^\beta,$$

where $\mathcal{P}$ denotes the powerset. A type $(T, \tau) \in \mathcal{T}^{\alpha \to \beta}$ is denoted as $\bigwedge T \to \tau$, or $\bigwedge_{i \in I}(f_i, \tau_i) \to \tau$ when $T = \{(f_i, \tau_i) \mid i \in I\}$. Moreover, to our terms we will not only assign a type τ, but also a flag $f \in \{\mathsf{pr}, \mathsf{np}\}$ (which together form a pair (f, τ)).

Intuitively, a term has type $\bigwedge T \to \tau$ when it can return τ, while taking an argument for which we can derive all pairs (of a flag and a type) from T. And, we assign the flag pr (productive), when this term (while being a subterm of a term of sort o) increases the value. To be more precise, a term is productive in two cases. First, when it uses the constant $\mathbf{1}+$. Notice however that this $\mathbf{1}+$ has to be really used: there exist terms which syntactically contain $\mathbf{1}+$, but the result of this $\mathbf{1}+$ is then ignored, like in $(\lambda x.\mathbf{0})\mathbf{1}+$. Second, a term which takes a productive argument and uses it at least twice is also productive.

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. The *type environment* Γ is a set of bindings of variables of the form $x^\alpha : (f, \tau)$, where $\tau \in \mathcal{T}^\alpha$. In Γ we may have multiple bindings for the same variable. By $dom(\Gamma)$ we denote the set of variables x which are binded by Γ, and by $\Gamma{\restriction}_{\mathsf{pr}}$ we denote the set of those binding from Γ which use flag pr.

The type system consists of the following rules:

$$\emptyset \vdash \mathbf{0} : (\mathsf{np}, \mathbf{r}) \qquad \emptyset \vdash \mathbf{1}+ : (\mathsf{pr}, (f, \mathbf{r}) \to \mathbf{r}) \qquad x : (f, \tau) \vdash x : (\mathsf{np}, \tau)$$

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

$$\frac{\Gamma \vdash M : (f', \bigwedge_{i \in I}(f_i^\bullet, \tau_i) \to \tau) \qquad \Gamma_i \vdash N : (f_i^\circ, \tau_i) \text{ for each } i \in I}{\Gamma \cup \bigcup_{i \in I} \Gamma_i \vdash MN : (f, \tau)} \ (@)$$

where in the (@) rule we assume that

– each pair $(f_i^\bullet, \tau_i)$ is different (where $i \in I$), and
– for each $i \in I$, $f_i^\bullet = \mathsf{pr}$ if and only if $f_i^\circ = \mathsf{pr}$ or $\mathit{\Gamma}_i{\restriction_{\mathsf{pr}}} \neq \emptyset$, and
– $f = \mathsf{pr}$ if and only if $f' = \mathsf{pr}$, or $f_i^\circ = \mathsf{pr}$ for some $i \in I$, or $|\mathit{\Gamma}{\restriction_{\mathsf{pr}}}| + \sum_{i\in I} |\mathit{\Gamma}_i{\restriction_{\mathsf{pr}}}| > |(\mathit{\Gamma} \cup \bigcup_{i\in I} \mathit{\Gamma}_i){\restriction_{\mathsf{pr}}}|$.

Let us explain the conditions of the (@) rule. The second condition says that when M requires a "productive" argument, either we can apply an N which is itself productive, or we can apply a nonproductive N which uses a productive variable; after substituting something for the variable N will become productive. The third condition says that MN is productive if M is productive, or if N is productive, or if some productive free variable is duplicated.

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

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. Consider a node of a derivation tree in which the (@) rule is used, with type environments $\mathit{\Gamma}$ and $\mathit{\Gamma}_i$ for $i \in I$. For $a \in \mathbb{N}$, the *order-a duplication factor* in such a node is defined as

$$|\{(x : (\mathsf{pr}, \sigma)) \in \mathit{\Gamma} \mid \mathit{ord}(x) = a\}| + \sum_{i\in I} |\{(x : (\mathsf{pr}, \sigma)) \in \mathit{\Gamma}_i \mid \mathit{ord}(x) = a\}| - \\ - |\{(x : (\mathsf{pr}, \sigma)) \in \mathit{\Gamma} \cup \bigcup_{i\in I} \mathit{\Gamma}_i \mid \mathit{ord}(x) = a\}|.$$

In other words, this is equal to the number of productive type bindings for variables of order a together in all the type environments $\mathit{\Gamma}$, $(\mathit{\Gamma}_i)_{i\in I}$, minus the number of such type bindings in their union.

Example 3. Below we give two example derivation trees. In the first tree, we denote by b_y the binding $y : (\mathsf{pr}, (\mathsf{pr}, \mathbf{r}) \to \mathbf{r})$, and by b_z the binding $z : (\mathsf{pr}, \mathbf{r})$.

$$\dfrac{b_y \vdash y : (\mathsf{np}, (\mathsf{pr}, \mathbf{r}) \to \mathbf{r}) \qquad \dfrac{b_y \vdash y : (\mathsf{np}, (\mathsf{pr}, \mathbf{r}) \to \mathbf{r}) \qquad b_z \vdash z : (\mathsf{np}, \mathbf{r})}{b_z,\ b_y \vdash y\ z : (\mathsf{np}, \mathbf{r})}\,(@)}{b_z,\ b_y \vdash y\ (y\ z) : (\mathsf{pr}, \mathbf{r})}\,(@)$$

$$\dfrac{\dfrac{\vdash \mathbf{1}\!\!+ : (\mathsf{pr}, (\mathsf{pr}, \mathbf{r}) \to \mathbf{r}) \qquad \dfrac{\vdash \mathbf{1}\!\!+ : (\mathsf{pr}, (\mathsf{pr}, \mathbf{r}) \to \mathbf{r}) \qquad x : (\mathsf{pr}, \mathbf{r}) \vdash x : (\mathsf{np}, \mathbf{r})}{x : (\mathsf{pr}, \mathbf{r}) \vdash \mathbf{1}\!\!+\ x : (\mathsf{pr}, \mathbf{r})}\,(@)}{x : (\mathsf{pr}, \mathbf{r}) \vdash \mathbf{1}\!\!+\ (\mathbf{1}\!\!+\ x) : (\mathsf{pr}, \mathbf{r})}\,(@)}{\vdash \lambda x.\mathbf{1}\!\!+\ (\mathbf{1}\!\!+\ x) : (\mathsf{pr}, (\mathsf{pr}, \mathbf{r}) \to \mathbf{r})}\,(\lambda)$$

The order-1 duplication factor of the root node of the first tree is 1, because the binding for y is used in both subtrees (and y is of order 1); the other nodes have duplication factors 0.

It is possible to derive six other type judgements containing the term $y\ (y\ z)$:

$$y : (\mathsf{np}, (\mathsf{pr}, \mathbf{r}) \to \mathbf{r}),\ z : (\mathsf{pr}, \mathbf{r}) \vdash y\ (y\ z) : (\mathsf{np}, \mathbf{r}),$$
$$y : (\mathsf{pr}, (\mathsf{pr}, \mathbf{r}) \to \mathbf{r}),\ y : (\mathsf{np}, (\mathsf{pr}, \mathbf{r}) \to \mathbf{r}),\ z : (\mathsf{pr}, \mathbf{r}) \vdash y\ (y\ z) : (\mathsf{np}, \mathbf{r}),$$
$$y : (\mathsf{pr}, (\mathsf{pr}, \mathbf{r}) \to \mathbf{r}),\ y : (\mathsf{pr}, (\mathsf{np}, \mathbf{r}) \to \mathbf{r}),\ z : (\mathsf{np}, \mathbf{r}) \vdash y\ (y\ z) : (\mathsf{np}, \mathbf{r}),$$
$$y : (\mathsf{pr}, (\mathsf{np}, \mathbf{r}) \to \mathbf{r}),\ y : (\mathsf{np}, (\mathsf{np}, \mathbf{r}) \to \mathbf{r}),\ z : (\mathsf{np}, \mathbf{r}) \vdash y\ (y\ z) : (\mathsf{np}, \mathbf{r}),$$
$$y : (\mathsf{np}, (\mathsf{pr}, \mathbf{r}) \to \mathbf{r}),\ y : (\mathsf{pr}, (\mathsf{np}, \mathbf{r}) \to \mathbf{r}),\ z : (\mathsf{np}, \mathbf{r}) \vdash y\ (y\ z) : (\mathsf{np}, \mathbf{r}),$$
$$y : (\mathsf{np}, (\mathsf{np}, \mathbf{r}) \to \mathbf{r}),\ z : (\mathsf{np}, \mathbf{r}) \vdash y\ (y\ z) : (\mathsf{np}, \mathbf{r}).$$

4 Krivine Machine

The Krivine machine [7] is an abstract machine that computes the weak head normal form of a λ-term, using explicit substitutions, called environments. Two properties of the Krivine machine are important for us. First, the Krivine machine performs β-reductions starting from the head redex; this redex is always a closed term. Second, the Krivine machine isolates closed subterms of a term using closures; we will be deriving types for each closure separately. We could perform β-reductions in this order and identify closed subterms also without the Krivine machine, but we believe that using it simplifies the presentation.

An *environment* is a function mapping some variables into closures. A *closure* is a pair $C = (M, \rho)$, where M is a term and ρ is an environment. We use the notation $\mathit{term}(C) := M$ and $\mathit{env}(C) := \rho$. A *configuration* of the Krivine machine is a pair (C, S), where C is a closure and S is a *stack*, which is a sequence of closures (with the topmost element on the left).

We require that in a closure (M, ρ), the environment is defined for every free variable of M; moreover $\mathit{term}(\rho(x))$ has to be of the same sort as x. We also require that in a configuration (C, S), when $\mathit{term}(C)$ is of sort $\alpha_1 \to \dots \to \alpha_k \to o$, then the stack S has k elements $C_1, \dots, C_k$, where $\mathit{term}(C_i)$ is of sort α_i, for each i. Let us also emphasize that we only consider "finite" closures, environments, configurations: an environment binds only finitely many variables, and after going repeatedly to a closure in the environment of a closure we will find an empty environment after finitely many steps.

The rules of the Krivine machine are as follows:

$$((\lambda x.M, \rho), CS) \xrightarrow{\lambda} ((M, \rho[x \mapsto C]), S),$$
$$((MN, \rho), S) \xrightarrow{@} ((M, \rho), (N, \rho)S),$$
$$((x, \rho), S) \xrightarrow{\mathit{Var}} (\rho(x), S),$$
$$((\mathbf{1}{+}, \rho), C) \xrightarrow{\mathbf{1}+} (C, \varepsilon).$$

Intuitively, a closure $C = (M, \rho)$ denotes the closed λ-term $[\![C]\!]$ which is obtained from M by substituting for every its free variable x the λ-term $[\![\rho(x)]\!]$. Also a configuration (C, S) denotes a closed λ-term $[\![C, S]\!]$ of sort o; this is the application $[\![C]\!][\![C_1]\!] \dots [\![C_k]\!]$, where $S = C_1 \dots C_k$. It is not difficult to see that

- when $(C,S) \xrightarrow{@} (C',S')$ or $(C,S) \xrightarrow{Var} (C',S')$, then $[\![C,S]\!] = [\![C',S']\!]$;
- when $(C,S) \xrightarrow{\lambda} (C',S')$, then $[\![C,S]\!]$ β-reduces to $[\![C',S']\!]$ (the head redex is eliminated);
- when $(C,S) \xrightarrow{\mathbf{1}+} (C',S')$, then $[\![C,S]\!] = (\mathbf{1}+\ [\![C',S']\!])$ (in particular the value of the new term is smaller by one than that of the old term).

From each configuration (C,S), as long as $\mathit{term}(C) \neq \mathbf{0}$, a (unique) step can be performed. Next, observe that each computation terminates after finite time. Indeed, the $\mathbf{1}+$ rule changes the denoted term into one with smaller value (and the value is not changed by the other rules). The λ rule performs β-reduction (and the term is not changed by the @ and *Var* rules), so as well it can be applied only finitely many times. The *Var* rule removes one closure from the configuration; the total number of closures (computed recursively) in the configuration decreases. The @ rule does not change this number, but increases the size of the stack, which is necessarily bounded by the number of closures. It follows that to compute the value of the term $[\![C,S]\!]$, it is enough to start the Krivine Machine from (C,S), and count how many times the $\mathbf{1}+$ rule was used.

In this paper we use an extension of the Krivine Machine, which also stores derivation trees. An *extended closure* is a triple (M,D,ρ), where M is a term of some sort α, and ρ is an environment (mapping variables to extended closures), and D is a partial function from $\{\mathsf{pr},\mathsf{np}\} \times \mathcal{T}^\alpha$ to derivation trees. Beside of $\mathit{term}(C)$ and $\mathit{env}(C)$ we use the notation $\mathit{der}(C) := D$, as well as $\mathit{tp}(C) := \mathit{dom}(D)$. The root of the tree assigned by D to a pair $(f^\bullet,\tau)$ has to be labeled by $\Gamma \vdash M : (f^\circ,\tau)$ such that $f^\bullet = \mathsf{pr}$ if and only if $f^\circ = \mathsf{pr}$ or $\Gamma\restriction_{\mathsf{pr}} \neq \emptyset$. Moreover, for each binding $(x : (g,\sigma)) \in \Gamma$ we require that $(g,\sigma) \in \mathit{tp}(\rho(x))$; denote this condition by $(\star)$. The partial function D can be also seen as a set of derivation trees: the pair (f,τ) to which a tree is assigned is determined by its root (however this is not an arbitrary set: to each pair we assign at most one tree).

A *configuration* of the extended Krivine machine is a pair (C,S), where C is an extended closure such that $|\mathit{tp}(C)| = 1$, and $S = C_1 \dots C_k$ is a *stack* of extended closures. We require that, when $\mathit{tp}(C) = \{(f, \bigwedge T_1 \to \dots \to \bigwedge T_k \to \mathbf{r})\}$, it holds $T_i \subseteq \mathit{tp}(C_i)$ for each i; denote this condition by $(\star\star)$.

The rules of the extended Krivine machine are as follows:

- $((\lambda x.M,D,\rho),CS) \xrightarrow{\lambda} ((M,D',\rho[x \mapsto C]),S)$, where the only tree in D' is obtained from the only tree in D by cutting off the root;
- $((MN,D,\rho),S) \xrightarrow{@} ((M,D_M,\rho),(N,D_N,\rho)S)$, where D_M contains the subtree of the tree in D which derives a type for M, and D_N contains all other subtrees (rooted in children of the root) of the tree in D;
- $((x,D,\rho),S) \xrightarrow{Var} ((\mathit{term}(\rho(x)), \mathit{der}(\rho(x))\restriction_{\mathit{dom}(D)}, \mathit{env}(\rho(x))),S)$;
- $((\mathbf{1}+,D,\rho),(M,D',\rho')) \xrightarrow{\mathbf{1}+} ((M,D'\restriction_{\{(f,\mathbf{r})\}},\rho'),\varepsilon)$, when the only element of $\mathit{dom}(D)$ is $(\mathsf{pr},(f,\mathbf{r}) \to \mathbf{r})$.

Let π be the projection from configurations of the extended machine to configurations of the standard one, which just drops the "*der*" component of every extended closure. Notice that when $(C,S) \to (C',S')$ in the extended machine, then $\pi(C,S) \to \pi(C',S')$ in the standard machine. Next, we observe that from each configuration, as long as the term in its main closure is not $\mathbf{0}$, we can perform a step (in particular, the result of the step satisfies all conditions of a configuration).

- In the case of $\lambda x.M$, the root of the derivation tree in D is labeled by $\Gamma \vdash \lambda x.M : (f, \bigwedge_{i\in I}(f_i,\tau_i) \to \tau)$. This tree begins by the (λ) rule, so $x \notin \mathit{dom}(\Gamma)$, and the only child of the root (which becomes the root of the new tree) is labeled by $\Gamma \cup \{x : (f_i,\tau_i) \mid i \in I\} \vdash M : (f,\tau)$. Notice that (due to conditions $(\star)$ and $(\star\star)$) for each binding $(y : (g,\sigma)) \in \Gamma$ we have $(g,\sigma) \in \mathit{tp}(\rho(y)) = \mathit{tp}(\rho[x \mapsto C](y))$, and for each $i \in I$ we have $(f_i,\tau_i) \in \mathit{tp}(C) = \mathit{tp}(\rho[x \mapsto C](x))$, which gives condition $(\star)$ for the new closure.
- In the application case, the derivation tree in D uses the $(@)$ rule in the root. Thus one child of the root is labeled by $\Gamma \vdash M : (f', \bigwedge_{i\in I}(f_i^\bullet,\tau_i) \to \tau)$, and the other children by $\Gamma_i \vdash N : (f_i^\circ,\tau_i)$ for each $i \in I$, where $f_i^\bullet = \mathsf{pr}$ if and only if $f_i^\circ = \mathsf{pr}$ or $\Gamma_i{\restriction_{\mathsf{pr}}} \neq \emptyset$. It follows that $\mathit{dom}(D_N) = \{(f_i^\bullet,\tau_i) \mid i \in I\}$. Simultaneously $\mathit{dom}(D_M) = \{(f'^\bullet, \bigwedge_{i\in I}(f_i^\bullet,\tau_i) \to \tau)\}$ for some $f'^\bullet$, so condition $(\star\star)$ holds for the new configuration. The definition of the $(@)$ rule ensures that each pair $(f_i^\bullet,\tau_i)$ is different, so D_N is really a (partial) function. Condition $(\star)$ for both the new closures is ensured by condition $(\star)$ for the original closure, since the type environment in the root of the derivation tree in D is a superset of Γ and of each Γ_i.
- In the *Var* case, the root of the tree in D is labeled by $x : (f,\tau) \vdash x : (\mathsf{np},\tau)$, where $\mathit{dom}(D) = \{(f,\tau)\}$. Thus condition $(\star)$ for (x, D, ρ) ensures that $\mathit{dom}(D) \subseteq \mathit{tp}(\rho(x))$.
- In the $\mathbf{1}\!+$ case, the root of the tree in D is labeled by $\emptyset \vdash \mathbf{1}\!+ : (\mathsf{pr}, (f,\mathbf{r}) \to \mathbf{r})$, so $\mathit{dom}(D)$ is as in the rule, and condition $(\star\star)$ ensures that $(f,\mathbf{r}) \in \mathit{dom}(D')$.

Example 4. We give an example computation of the extended Krivine machine. In our closures we use fragments of the derivation trees given in Example 3. By T_1, T_2, T_3, U_1, U_2, U_3, U_4 we denote the subtrees of these trees, where:

- T_1 derives $y : (\mathsf{pr}, (\mathsf{pr},\mathbf{r}) \to \mathbf{r}),\ z : (\mathsf{pr},\mathbf{r}) \vdash y\ (y\ z) : (\mathsf{pr},\mathbf{r})$,
- T_2 derives $y : (\mathsf{pr}, (\mathsf{pr},\mathbf{r}) \to \mathbf{r}) \vdash y : (\mathsf{np}, (\mathsf{pr},\mathbf{r}) \to \mathbf{r})$,
- T_3 derives $y : (\mathsf{pr}, (\mathsf{pr},\mathbf{r}) \to \mathbf{r}),\ z : (\mathsf{pr},\mathbf{r}) \vdash y\ z : (\mathsf{np},\mathbf{r})$,
- U_1 derives $\vdash \lambda x.\mathbf{1}\!+\ (\mathbf{1}\!+\ x) : (\mathsf{pr}, (\mathsf{pr},\mathbf{r}) \to \mathbf{r})$,
- U_2 derives $x : (\mathsf{pr},\mathbf{r}) \vdash \mathbf{1}\!+\ (\mathbf{1}\!+\ x) : (\mathsf{pr},\mathbf{r})$,
- U_3 derives $\vdash \mathbf{1}\!+ : (\mathsf{pr}, (\mathsf{pr},\mathbf{r}) \to \mathbf{r})$,
- U_4 derives $x : (\mathsf{pr},\mathbf{r}) \vdash \mathbf{1}\!+\ x : (\mathsf{pr},\mathbf{r})$.

Additionally, we use the following derivation tree, which we denote V_1.

$$\frac{\vdash \mathbf{1}\!+ : (\mathsf{pr}, (\mathsf{np},\mathbf{r}) \to \mathbf{r}) \qquad \vdash \mathbf{0} : (\mathsf{np},\mathbf{r})}{\vdash \mathbf{1}\!+\ \mathbf{0} : (\mathsf{pr},\mathbf{r})}\ (@)$$

To shorten the notation, we denote:

$$\begin{aligned}
&\rho := [y \mapsto (\lambda x.\mathbf{1}\!+\ (\mathbf{1}\!+\ x), \{U_1\}, \emptyset),\ z \mapsto (\mathbf{1}\!+\ \mathbf{0}, \{V_1\}, \emptyset)],\\
&\eta := [x \mapsto (y\ z, \{T_3\}, \rho)].
\end{aligned}$$

The extended Krivine machine can transition as follows:

$$\begin{aligned}
&((y\ (y\ z), \{T_1\}, \rho), \varepsilon) \xrightarrow{@} ((y, \{T_2\}, \rho), (y\ z, \{T_3\}, \rho)) \xrightarrow{Var}\\
&\xrightarrow{Var} ((\lambda x.\mathbf{1}\!+\ (\mathbf{1}\!+\ x), \{U_1\}, \emptyset), (y\ z, \{T_3\}, \rho)) \xrightarrow{\lambda} ((\mathbf{1}\!+\ (\mathbf{1}\!+\ x), \{U_2\}, \eta), \varepsilon) \xrightarrow{@}\\
&\xrightarrow{@} ((\mathbf{1}\!+, \{U_3\}, \eta), (\mathbf{1}\!+\ x, \{U_4\}, \eta)) \xrightarrow{\mathbf{1}+} ((\mathbf{1}\!+\ x, \{U_4\}, \eta), \varepsilon) \xrightarrow{@} \dots
\end{aligned}$$

Next, we observe that we can really add some derivation trees to a configuration.

Lemma 5. *For each configuration (C, S) of the standard Krivine machine there exists a configuration (C', S') of the extended machine such that $\pi(C', S') = (C, S)$.*

Proof. We use induction on the length of the longest computation starting from the configuration (C, S) (this computation is unique and has finite length). We have five cases depending on the form of the term in the main closure. Before starting the case analysis, we observe that for each closure B there exists an extended closure B' such that $\pi(B') = B$. To construct such B' we can add the partial function with empty domain everywhere inside B. (This cannot be applied for a configuration: the *tp* of the main closure of a configuration is required to have size 1).

Consider first a configuration of the form $((\mathbf{0}, \rho_s), \varepsilon)$. Then to the main closure we can add the derivation tree using the rule $\emptyset \vdash \mathbf{0} : (\mathsf{np}, \mathbf{r})$, and everywhere inside ρ_s we can add the partial function with empty domain.

Next, consider a configuration of the form $((\lambda x.M, \rho_s), C_s S_s)$. Its successor is $((M, \rho_s[x \mapsto C_s]), S_s)$, which by the induction assumption can be extended to a configuration $((M, D', \rho[x \mapsto C]), S)$ of the extended machine. Potentially $\rho_s(x)$ can be defined. In such situation we can assume that $\rho(x)$ is defined and $\pi(\rho(x)) = \rho_s(x)$; otherwise we assume that $\rho(x)$ is undefined. Notice that these assumptions do not change $\rho[x \mapsto C]$, where $\rho(x)$ is overwritten. The label of the root of the tree in D' can be denoted as $\Gamma \cup \{x : (f_i, \tau_i) \mid i \in I\} \vdash M : (f, \tau)$, where $x \notin dom(\Gamma)$. We can apply the (λ) rule, and obtain a tree rooted by $\Gamma \vdash \lambda x.M : (f, \bigwedge_{i \in I}(f_i, \tau_i) \to \tau)$. The thesis is satisfied by the configuration $((\lambda x.M, D, \rho), CS)$, where D contains this new tree. Notice that conditions $(\star)$ and $(\star\star)$ are satisfied for this configuration, since they were satisfied for $((M, D', \rho[x \mapsto C]), S)$.

Before considering the next case, notice that any two extended closures C_1, C_2 such that $\pi(C_1) = \pi(C_2)$ can be merged into one extended closure C such that $\pi(C) = \pi(C_1)$ and $tp(C) = tp(C_1) \cup tp(C_2)$. To do that, by induction we create an environment ρ, which maps each variable $x \in dom(env(C_1))$ into the extended closure obtained by merging $env(C_1)(x)$ and $env(C_2)(x)$. We also create D which is equal to $\mathit{der}(C_1)$ on $\mathit{tp}(C_1)$, and is equal to $\mathit{der}(C_2)$ on $\mathit{tp}(C_2) \setminus \mathit{tp}(C_1)$. As C we take $(\mathit{term}(C_1), D, \rho)$; notice that condition $(\star)$ remains satisfied.

Next, consider a configuration of the form $((MN, \rho_s), S_s)$. Its successor is $((M, \rho_s), (N, \rho_s)S_s)$, which by the induction assumption can be extended to a configuration $((M, D_M, \rho_M), (N, D_N, \rho_N)S)$ of the extended machine. Let ρ be obtained by merging ρ_M and ρ_N, as described in the previous paragraph. Denote $\mathit{dom}(D_M)$ as $\{(f'^{\bullet}, \bigwedge_{i\in I}(f_i^{\bullet}, \tau_i) \to \tau)\}$, where each pair $(f_i^{\bullet}, \tau_i)$ is different. Then $(f_i^{\bullet}, \tau_i) \in \mathit{dom}(D_N)$ for each $i \in I$, by condition $(\star\star)$. Let $\Gamma \vdash M : (f', \bigwedge_{i\in I}(f_i^{\bullet}, \tau_i) \to \tau)$ be the label of the root of the tree in D_M, and let $\Gamma_i \vdash N : (f_i^{\circ}, \tau_i)$ be the label of the root of $D_N(f_i^{\bullet}, \tau_i)$ for each i; recall that $f_i^{\bullet} = \mathsf{pr}$ if and only if $f_i^{\circ} = \mathsf{pr}$ or $\Gamma_i\restriction_{\mathsf{pr}} \neq \emptyset$. We can apply the (@) rule to these roots, and obtain a derivation tree with root labeled by $\Gamma \cup \bigcup_{i\in I}\Gamma_i \vdash MN : (f, \tau)$ (for some f). Then $((MN, D, \rho), S)$, where D contains this new tree, is a correct configuration and satisfies the thesis.

Next, consider a configuration $((x, \rho_s), S_s)$. Its successor is $(\rho_s(x), S_s)$, which by the induction assumption can be extended to a configuration (C, S) of the extended machine. Let $\{(f, \tau)\} := \mathit{tp}(C)$. We take D containing the derivation $x : (f, \tau) \vdash x : (\mathsf{np}, \tau)$, and we take ρ mapping x to C, and each other variable $y \in \mathit{dom}(\rho_s)$ into any extended closure E_y such that $\pi(E_y) = \rho_s(y)$. Then $((x, D, \rho), S)$ is a configuration and satisfies the thesis.

Finally, consider a configuration of the form $((\mathbf{1}{+}, \rho_s), C_s)$. Its successor is (C_s, ε), which by the induction assumption can be extended to a configuration (C, ε) of the extended machine. Let $\{(f, \mathbf{r})\} := \mathit{tp}(C)$. We take D containing the derivation $\emptyset \vdash \mathbf{1}{+} : (\mathsf{pr}, (f, \mathbf{r}) \to \mathbf{r})$, and we take ρ mapping each variable $x \in \mathit{dom}(\rho_s)$ into any extended closure E_x such that $\pi(E_x) = \rho_s(x)$. Then $((\mathbf{1}{+}, D, \rho), C)$ is a configuration and satisfies the thesis. □

5 Assigning Values to Configurations

To a configuration of a Krivine machine we assign two numbers, *low* and *high*, which estimate (from below and from above, respectively) the value of the term represented by the configuration.

Let C be an extended closure, and let $(f, \tau) \in \mathit{tp}(C)$. By $\mathit{inc}_0(C, f, \tau)$ we denote the number of leaves of $\mathit{der}(C)(f, \tau)$ using the $\mathbf{1}{+}$ rule, and by $\mathit{inc}_a(C, f, \tau)$ for $a > 0$ we denote the sum of order-$(a-1)$ duplication factors of all (@) nodes of $\mathit{der}(C)(f, \tau)$.

We define $\mathit{low}(C, f, \tau)$, and $\mathit{high}_a(C, f, \tau)$, and $\mathit{rec}_a(C, f, \tau)$, and $\mathit{ext}_a(C, f, \tau)$ for each $a \in \mathbb{N}$ by induction on the structure of C (where *rec* stands for "recursive" and *ext* for "external"). Let Γ be the type environment used in the root of $\mathit{der}(C)(f, \tau)$. We take

$$\mathit{low}(C, f, \tau) := \sum_{a\in\mathbb{N}} \mathit{inc}_a(C, f, \tau) + \sum_{(x:(g,\sigma))\in\Gamma} \mathit{low}(\mathit{env}(C)(x), g, \sigma),$$

$$\mathit{rec}_a(C, f, \tau) := \mathit{inc}_a(C, f, \tau) + \sum_{(x:(g,\sigma))\in\Gamma} \mathit{ext}_a(\mathit{env}(C)(x), g, \sigma),$$

$$\begin{aligned}
&\mathit{high}_a(C,f,\tau) := (\mathit{rec}_a(C,f,\tau)+1)\cdot 2^{\mathit{high}_{a+1}(C,f,\tau)}-1,\\
&\mathit{high}_a(C,f,\tau) := 0 \qquad \text{if } \mathit{rec}_b(C,f,\tau)=0 \text{ for all } b\geq a,\\
&\mathit{ext}_a(C,f,\tau) := \begin{cases} 0 & \text{if } a > \mathit{ord}(\mathit{term}(C)),\\ \mathit{high}_a(C,f,\tau) & \text{if } a = \mathit{ord}(\mathit{term}(C)),\\ \mathit{rec}_a(C,f,\tau) & \text{if } a < \mathit{ord}(\mathit{term}(C)).\end{cases}
\end{aligned}$$

For a configuration (C_0,S) with $S=C_1\dots C_k$ and $\mathit{tp}(C_0)=\{(f,\bigwedge T_1\to\dots\to\bigwedge T_k\to\mathbf{r})\}$ we define, denoting $T_0:=\mathit{tp}(C_0)$:

$$\begin{aligned}
&\mathit{low}(C_0,S) := \sum_{i=0}^{k}\sum_{(g,\sigma)\in T_i}\mathit{low}(C_i,g,\sigma),\\
&\mathit{rec}_a(C_0,S) := \sum_{i=0}^{k}\sum_{(g,\sigma)\in T_i}\mathit{ext}_a(C_i,g,\sigma),\\
&\mathit{high}_a(C_0,S) := (\mathit{rec}_a(C_0,S)+1)\cdot 2^{\mathit{high}_{a+1}(C_0,S)}-1,\\
&\mathit{high}_a(C_0,S) := 0 \qquad \text{if } \mathit{rec}_b(C_0,S)=0 \text{ for all } b\geq a,\\
&\mathit{high}(C_0,S) := \mathit{high}_0(C_0,S).
\end{aligned}$$

Example 6. We will compute these numbers for the first configuration from Example 4. Denoting

$$C_2=(\lambda x.\mathbf{1}\!+\,(\mathbf{1}\!+\,x),\{U_1\},\emptyset), \qquad C_3=(\mathbf{1}\!+\,\mathbf{0},\{V_1\},\emptyset),$$

$$C_1=(y\ (y\ z),\{T_1\},[y\mapsto C_2,\ z\mapsto C_3]),$$

this configuration is (C_1,ε). It denotes the term

$$(\lambda x.\mathbf{1}\!+\,(\mathbf{1}\!+\,x))\ ((\lambda x.\mathbf{1}\!+\,(\mathbf{1}\!+\,x))\ (\mathbf{1}\!+\,\mathbf{0})),$$

which has value 5. It holds $\mathit{tp}(C_1)=\mathit{tp}(C_3)=\{(\mathsf{pr},\mathbf{r})\}$ and $\mathit{tp}(C_2)=\{(\mathsf{pr},\tau)\}$, where $\tau=((\mathsf{pr},\mathbf{r})\to\mathbf{r})$. Because we have two $\mathbf{1}\!+$ nodes in U_1, and one in V_1, it holds $\mathit{inc}_0(C_2,\mathsf{pr},\tau)=2$ and $\mathit{inc}_0(C_3,\mathsf{pr},\mathbf{r})=1$. The order-1 duplication factor in the root of T_1 causes that $\mathit{inc}_2(C_1,\mathsf{pr},\mathbf{r})=1$. All other $\mathit{inc}_i(\cdot,\cdot,\cdot)$ equal 0. It follows that

$$\begin{aligned}
&\mathit{low}(C_2,\mathsf{pr},\tau)=\mathit{ext}_0(C_2,\mathsf{pr},\tau)=\mathit{high}_0(C_2,\mathsf{pr},\tau)=\mathit{rec}_0(C_2,\mathsf{pr},\tau)=2,\\
&\mathit{low}(C_3,\mathsf{pr},\mathbf{r})=\mathit{ext}_0(C_3,\mathsf{pr},\mathbf{r})=\mathit{high}_0(C_3,\mathsf{pr},\mathbf{r})=\mathit{rec}_0(C_3,\mathsf{pr},\mathbf{r})=1,\\
&\mathit{low}(C_1,\mathsf{pr},\mathbf{r})=\mathit{low}(C_2,\mathsf{pr},\tau)+\mathit{low}(C_3,\mathsf{pr},\mathbf{r})+\mathit{inc}_2(C_1,\mathsf{pr},\mathbf{r})=4,\\
&\mathit{rec}_0(C_1,\mathsf{pr},\mathbf{r})=3, \qquad \mathit{high}_2(C_1,\mathsf{pr},\mathbf{r})=\mathit{rec}_2(C_1,\mathsf{pr},\mathbf{r})=1,\\
&\mathit{high}_1(C_1,\mathsf{pr},\mathbf{r})=(0+1)\cdot 2^1-1=1,\\
&\mathit{ext}_0(C_1,\mathsf{pr},\mathbf{r})=\mathit{high}_0(C_1,\mathsf{pr},\mathbf{r})=(3+1)\cdot 2^1-1=7,\\
&\mathit{low}(C,\varepsilon)=4, \qquad \mathit{high}(C,\varepsilon)=7.
\end{aligned}$$

In the second configuration of the computation we do not have any duplication factor, and we count five $\mathbf{1}\!+$ nodes. Notice that both C_2 and C_3 appear in two environments, but C_3 is not used in the first of them (more precisely, no binding for z is appears in the type environment of T_2), so the **1**+ node in C_3 is counted only once. Thus both *low* and *high* of this configuration are 5, which is equal to its value.

Let us explain the intuitions behind the definitions of *low* and *high*. First, concentrate on *low*. It counts the number of **1**+ leaves of our derivation trees. Our type system ensures that each such **1**+ will be used (and thus it will add 1 to the value of the term). It also counts duplication factors of (@) nodes of derivation trees. When a duplication factor in some node is 1 (and similarly for any positive number), some "productive" subtree of the (@) node will be used twice. And such a subtree increases the value of the term at least by one—it either contains some **1**+, or some other duplication, which will be now performed twice instead of once.

In the formula for *high*, which is going to be an upper bound for the value, we have to overapproximate. For that reason, it is not enough to look on the sum of duplication factors; the orders on which they appear start to play a role. Consider the highest k for which the order-k duplication factor is positive in some (@) node, and consider an innermost node such that it is positive; say, it is equal to 1. Inside, we only have duplication factors of smaller order, and some **1**+ nodes. When the application described by the (@) node is performed, they will be replicated twice. Similarly, the next (@) node also can multiply their number by two, and so on. Next, analogous analysis for order-$(k-1)$ duplication factors (whose number is already increased by order-k duplication factors) shows that each of them can multiply by two the number of duplication factors of order smaller than $k-1$ (and of **1**+ nodes), and so on. This justifies on the intuitive level the exponential character of the formula[2] for *high*, but in fact this analysis cannot be formalized (in some sense it is incorrect). The problem is that the innermost node with positive duplication factor for the highest order does not necessarily denote a closed term. So a positive duplication factor not only implies that the subterms will be replicated, but also the free variables will be used more times (and we do not know how "big" terms will be substituted there). Thus it is important in our correctness proof that we reduce only such redexes $(\lambda x.M)N$ in which N is closed; this is always the case for the head redex, which is reduced by the Krivine machine.

However in the formula we do not make just one tower of exponentials at the end, but we compute some exponentials already for some inner closures. This is essential for the proof of correctness, since otherwise Lemma 9 would be false (although this modification makes the *high* value even smaller). The idea behind that is as follows. When we have a closed term M, its subterm of order $a \geq ord(M)$ cannot be duplicated by anything outside M; only the whole M can be duplicated (or subterms of M which are of order smaller order than M). Oppositely, a subterm of order $a < ord(M)$ can be duplicated by things from

[2] One can observe that the order-0 duplication factor always equals 0 (an order-0 term can be used only once). Thus in $high_0$ we could multiply rec_0 directly by 2^{high_2}. However this observation would only complicate the proof.

outside of M, because we can pass this subterm as an argument to an argument of M. Thus basically inc_a is cumulated recursively along closures; however for a closure of some order k we can forget about its duplication factors in inc_a for $a > k$—they will only be applied to inc_k contained inside this closure, so we can predict their result in $high_k$.

The next two propositions state that *low* and *high* extend quantitatively the information in the pr/np flag: 0 corresponds to np, and positive numbers to pr.

Proposition 7. *Let C be a closure, and let $(\mathsf{np}, \tau) \in tp(C)$. Then it holds $rec_a(C, \mathsf{np}, \tau) = 0$ for each $a \in \mathbb{N}$.*

Proof. The root of $der(C)(\mathsf{np}, \tau)$ is labeled by a type judgement $\Gamma \vdash term(C) : (\mathsf{np}, \tau)$, where $\Gamma{\restriction_{\mathsf{pr}}} = \emptyset$. It is easy to see by induction on the tree structure, that a derivation tree ending with the np flag has duplication factors of each (@) node (and each order) equal to zero, as well as it does not contain **1+** leaves. It follows that $inc_a(C, \mathsf{np}, \tau) = 0$. Because $\Gamma{\restriction_{\mathsf{pr}}} = \emptyset$, the added rec_a components are also equal to 0, by induction on the structure of the closure. □

Proposition 8. *Let C be a closure, and let $(\mathsf{pr}, \tau) \in tp(C)$. Then it holds $low(C, \mathsf{pr}, \tau) > 0$.*

Proof. When $der(C)(\mathsf{pr}, \tau)$ is labeled by a type judgement $\Gamma \vdash term(C) : (f, \tau)$, we have one of two cases. One possibility is that $f = \mathsf{pr}$. Then it is easy to see by induction on the tree structure, that a derivation tree ending with the pr flag either has a **1+** leaf, or an (@) node with a positive duplication factor for some order. Otherwise we have $\Gamma{\restriction_{\mathsf{pr}}} \neq \emptyset$. Then by induction on the structure of the closure we obtain that some of the added *low* components (for closures in the environment) is positive. □

Below we have the key lemma about the *low* and *high* numbers.

Lemma 9. *Let (C, S) be a configuration of the extended Krivine machine, which evolves to (C', S') in one step. If this was the* **1+** *step, we have $low(C, S) \leq 1 + low(C', S')$ and $high(C, S) \geq 1 + high(C', S')$; otherwise we have $low(C, S) \leq low(C', S')$ and $high(C, S) \geq high(C', S')$.*

Proof (sketch). The proof consists of tedious but straightforward calculations. We have four rules of the Krivine machine, which we have to analyze. We will see that only in the application rule we can have inequalities; for the other rules we have equalities. In all cases only the "front" of the configuration changes. In *low* and *high* for the old configuration we include some *low* and *high* of closures in the environment or on the stack, for some pairs (g, σ). We see that for the new configuration we include exactly the same closures with the same (g, σ) pairs. Thus we have to locally analyze what changes only near the "front" of the configuration.

For the **1+** rule this is immediate. We remove a closure $(\mathbf{1+}, D, \rho)$, where the only tree in D uses the rule $\emptyset \vdash \mathbf{1+} : (\mathsf{pr}, (f, \mathbf{r}) \to \mathbf{r})$. Since inc_0 for this closure is 1, and inc_a for $a > 0$ is 0, during the step we subtract 1 from *low* and *high*.

Also the case of the *Var* rule is very easy. This time we remove a closure (x, D, ρ), where the only tree in D uses the rule $x : (f, \tau) \vdash x : (\mathsf{np}, \tau)$. This closure has inc_a equal to 0 for each a, so *low* and *high* do not change.

In the λ rule we only move one closure from the stack to the environment, so *low* and *high* do not change as well. It can happen that the order of $\lambda x.M$ and of M is different, and the definition of ext_a is sensitive for that. But, since all other terms in the stack are of order smaller than the order of M (and the order of $\lambda x.M$), this change of order does not influence the result: some exponents which were computed outside of the closure with $\lambda x.M$ will be now computed inside the closure with M.

Finally, consider the case $((MN, D, \rho), S) \xrightarrow{@} ((M, D_M, \rho), (N, D_N, \rho)S)$. For *low* the analysis is quite simple. The root of the tree in D had some duplication factor, which were added to *low* in the old configuration, but is not in the new one. But such duplication factor counts how many times a productive binding of a variable in the type environment in D is replicated in the type environments of the trees in D_M and D_N. In the new configuration, the *low* for these bindings will be added for each copy. Since by Proposition 8 these *low* are positive, they will compensate the duplication factor of the root, which is subtracted.

For *high* we have two phenomena. The first concerns the replication of variable bindings in the type environments. We do not have to care about nonproductive bindings, since by Proposition 7 their rec_a is 0. Let dp_a be the order-a duplication factor at the root of the tree in D. A productive binding for a variable of order a is replicated at most dp_a times (instead of once in D it appears at most $dp_a + 1$ times in D_M and D_N). Notice the shift of orders: in the old configuration we were adding dp_a to inc_{a+1}. Thus without it, $high_a$ decreases 2^{dp_a} times. On the other hand, a closure (from ρ) of order a adds something to rec_b only for $b \leq a$ (otherwise its ext_b is 0), and now this rec_b will be multiplied by (at most) $dp_a + 1$. Due to the inequality $2^{dp_a} \geq dp_a + 1$, we see that $high_a$ will not increase. In fact it decreases by at least dp_a, thanks to the $+1$ in the formula for $high_a$. Thus we can repeat the same argument for $a - 1$, and continue by induction for all $b \leq a$. The second phenomenon is that $ord(N)$ is smaller than $ord(M)$. This implies that previously we were first adding together some ext_a for elements of ρ, and then making a tower of exponents in the closure (MN, D, ρ), while now we are making the tower of exponents inside (N, D_N, ρ), separately for each pair $(g, \sigma) \in dom(D_N)$, and then we are summing the results. But this can only decrease the result, as described by the inequality

$$(a + b + 1) \cdot 2^{c+d} - 1 \geq (a + 1) \cdot 2^c - 1 + (b + 1) \cdot 2^d - 1.$$

We also notice that $ord(MN)$ can be smaller than $ord(M)$, but this does not influence the result, since all other elements on the stack are of smaller order (similarly to the λ case). □

Corollary 10. *Let (C, S) be a configuration of the extended Krivine machine. Then the value of the term $[\![C, S]\!]$ is not smaller than $low(C, S)$, and not greater than $high(C, S)$.*

Proof. Induction on the length of the maximal computation from (C, S). If this length is 0, we have $\mathit{term}(C) = \mathbf{0}$, and $\mathit{tp}(C) = \{(\mathsf{np}, \mathbf{r})\}$, and $\mathit{der}(C)(\mathsf{np}, \mathbf{r})$ consists of the rule $\emptyset \vdash \mathbf{0} : (\mathsf{np}, \mathbf{r})$, so $\mathit{low}(C, S) = 0 = \mathit{high}(C, S)$, and $[\![C, S]\!] = \mathbf{0}$ has value 0. Otherwise we use the induction assumption and Lemma 9.

Next, we state that if $\mathit{low}(C, S)$ is small, then also $\mathit{high}(C, S)$ is small, so $\mathit{low}(C, S)$ (and $\mathit{high}(C, S)$ as well) really approximates the value of $[\![C, S]\!]$.

Lemma 11. *For all $k, L \in \mathbb{N}$ there exists a number $H_{k,L}$ such that for each configuration (C, S) such that $\mathit{low}(C, S) \le L$ and such that each variable appearing anywhere inside (C, S) (inside a term or an environment) is of order at most k, it holds $\mathit{high}(C, S) \le H_{k,L}$.*

Proof. We define

$$H_{0,L} := (L + 1) \cdot 2^L - 1,$$
$$H_{a+1,L} := (L + 1) \cdot 2^{H_{a,L}} - 1 \qquad \text{for each } a \in \mathbb{N}.$$

Let $\#_{\mathit{cl}}(C, S)$ denote the number of closures everywhere (recursively) inside (C, S), and let $|S|$ denote the length of the stack. We prove the inequality by induction on $2 \cdot \#_{\mathit{cl}}(C, S) - |S|$.

Assume first that S and $\mathit{env}(C)$ are empty. Let $\mathit{tp}(C) = \{(f, \tau)\}$. Then $\mathit{low}(C, S) = \mathit{low}(C, f, \tau) = \sum_{a \in \mathbb{N}} \mathit{inc}_a(C, f, \tau)$, and $\mathit{high}(C, S) = \mathit{high}_0(C, f, \tau)$ with $\mathit{rec}_a(C, f, \tau) = \mathit{inc}_a(C, f, \tau) \le \mathit{low}(C, S)$ for each $a \in \mathbb{N}$. Since each variable in $\mathit{term}(C)$ is of order at most k, for $a > k + 1$ (which gives $a - 1 > k$) the order-$(a - 1)$ duplication factor of any (@) node in $\mathit{der}(C)(f, \tau)$ is zero, thus also $\mathit{inc}_a(C, f, \tau) = 0$. We see for $a \in \{0, 1, \dots, k\}$ that $\mathit{high}_a(C, f, \tau) \le H_{k-a,L}$.

Next, assume that S is nonempty. Denote $((M, D_M, \rho_M), (N, D_N, \rho_N)S') := (C, S)$. W.l.o.g. we can assume that $\mathit{dom}(\rho_M) \cap \mathit{dom}(\rho_N) = \emptyset$; otherwise we can rename the variables in M, D_M, ρ_M so that they are different from the variables in $\mathit{dom}(\rho_N)$, and such renaming does not change the *low* and *high* values. Denote $\rho := \rho_M \cup \rho_N$. Notice that $((M, D_M, \rho), (N, D_N, \rho)S')$ is a configuration with the same *low* and *high* as (C, S). The tree in D_M has root's label of the form $\Gamma \vdash M : (f', \bigwedge_{i \in I}(f_i^\bullet, \tau_i) \to \tau)$, where each pair $(f_i^\bullet, \tau_i)$ is different. Moreover, for each $i \in I$, we have a derivation tree $D_N(f_i^\bullet, \tau_i)$ rooted by some $\Gamma_i \vdash N : (f_i^\circ, \tau_i)$ such that $f_i^\bullet = \mathsf{pr}$ if and only if $f_i^\circ = \mathsf{pr}$ or $\Gamma_i{\restriction_{\mathsf{pr}}} \neq \emptyset$. Thus we can apply the (@) rule to these trees, and obtain a tree rooted by $\Gamma \cup \bigcup_{i \in I} \Gamma_i \vdash MN : (f, \tau)$ for some f. Let $C' := (MN, D, \rho)$, where D contains this new tree. We notice that (C', S') is a configuration (satisfies conditions ($\star$) and ($\star\star$)), and the machine can make a step from it to $((M, D_M, \rho), (N, D_N, \rho)S')$. Lemma 9 implies that $\mathit{low}(C', S') \le \mathit{low}(C, S) \le L$ and $\mathit{high}(C, S) \le \mathit{high}(C', S')$. It holds $\#_{\mathit{cl}}(C', S') = \#_{\mathit{cl}}(C, S) - 1$, and $|S'| = |S| - 1$, and the maximal order of a variable in these two configurations is the same. The induction assumption for (C', S') tells us that $\mathit{high}(C', S') \le H_{k,L}$.

Finally, assume that S is empty, but $\mathit{env}(C)$ is nonempty. Fix some variable $x \in \mathit{dom}(\mathit{env}(C))$, and denote $(M, D, \rho[x \to C_x]) := C$, where $x \notin \mathit{dom}(\rho)$. Let $\Gamma \cup \{x : (f_i, \tau_i) \mid i \in I\} \vdash M : (f, \tau)$ with $x \notin \mathit{dom}(\Gamma)$ be the label of the root of the tree in D. We can append the (λ) rule to this tree, and obtain a tree with root labeled by $\Gamma \vdash \lambda x.M : (f, \bigwedge_{i\in I}(f_i, \tau_i) \to \tau)$. Let $C' := (\lambda x.M, D', \rho)$, where D' contains this new tree. We notice that (C', C_xS') is a configuration (satisfies $(\star)$ and $(\star\star)$), and the machine can make a step from it to (C, S). Lemma 9 implies that $\mathit{low}(C', C_xS') \leq \mathit{low}(C, S) \leq L$ and $\mathit{high}(C, S) \leq \mathit{high}(C', C_xS')$. Notice that $\#_{cl}(C', C_xS') = \#_{cl}(C, S)$, and $|S'| = |S| + 1$, and the maximal order of a variable in these two configurations is the same. The induction assumption for (C', C_xS') tells us that $\mathit{high}(C', C_xS') \leq H_{k,L}$. □

6 Representing Tuples

In this section conclude the proof of Theorems 1 and 2.

Proof (Theorem 2). We define $\mathit{dim}(\alpha) = |\mathcal{P}(\{\mathsf{pr}, \mathsf{np}\} \times \mathcal{T}^{\alpha\to o})|$. For a closed term M of sort $\alpha \to o$, let $\mathit{types}(M)$ be the set of pairs $(f, \bigwedge T \to \mathbf{r})$ such that we can derive $\emptyset \vdash \mathit{nf}(M) : (f, \bigwedge T \to \tau)$, where $\mathit{nf}(M)$ is the β-normal form of M. We will show that when $\mathit{types}(M) = \mathit{types}(M')$ then also $M \sim M'$; the thesis of the theorem will follow, since we have at most $\mathit{dim}(\alpha)$ possible sets $\mathit{types}(M)$.

Thus suppose $\mathit{types}(M) = \mathit{types}(M')$, and consider a sequence $N_1, N_2, \dots$ of terms of sort α, such that the sequence of values of $M\ N_1, M\ N_2, \dots$ is bounded. W.l.o.g. we can assume that M, M', and all N_i are in β-normal form (since the value of $M\ N_i$ and of $\mathit{nf}(M)\ \mathit{nf}(N_i)$ is exactly the same). For each $i \in \mathbb{N}$, there exists a correct configuration of the form $((M, D_i^M, \emptyset), (N_i, D_i^N, \emptyset))$, denote it (C_i, E_i) (we use Lemma 5 for $((M, \emptyset), (N_i, \emptyset))$). Let $\{(f_i, \bigwedge T_i \to \mathbf{r})\} := \mathit{dom}(D_i^M)$, and let $D_i^{M'}$ contain a derivation tree rooted by $\emptyset \vdash M' : (f_i, \bigwedge T_i \to \mathbf{r})$, which exists thanks to equality of types. Let $C_i' := (M', D_i^{M'}, \emptyset)$. Then (C_i', E_i) is a correct configuration as well. Since $\mathit{low}(C_i, E_i)$ is not greater than the value of $M\ N_i$ (Corollary 10), also $\mathit{low}(C_i, E_i)$ is bounded (when ranging over $i = 1, 2, \dots$). Next, we see that

$$\begin{aligned}
&\mathit{low}(C_i', E_i) + \mathit{low}(C_i, f_i, \textstyle\bigwedge T_i \to \mathbf{r}) = \\
&= \mathit{low}(C_i', f_i, \textstyle\bigwedge T_i \to \mathbf{r}) + \displaystyle\sum_{(g,\sigma)\in T_i} \mathit{low}(E_i, g, \sigma) + \mathit{low}(C_i, f_i, \textstyle\bigwedge T_i \to \mathbf{r}) = \\
&= \mathit{low}(C_i, E_i) + \mathit{low}(C_i', f_i, \textstyle\bigwedge T_i \to \mathbf{r}).
\end{aligned}$$

Since C_i, f_i, T_i, C_i' come from a finite set, we obtain that $\mathit{low}(C_i', E_i)$ is bounded as well (by some L). Notice that the maximal order of a variable appearing anywhere inside M' or some N_i is $\mathit{ord}(\alpha)$, because these terms are in β-normal form. Thus $\mathit{high}(C_i', E_i)$ is bounded by $H_{\mathit{ord}(\alpha),L}$ (Lemma 11). It follows that the sequence of values of $M'\ N_1, M'\ N_2, \dots$ is bounded by $H_{\mathit{ord}(\alpha),L}$ as well (Corollary 10). The opposite implication (from M' to M) is completely symmetric. □

Proof (Theorem 1). This is an immediate consequence of Theorem 2. Assume that for some i there exists a set X_i as in the statement of the theorem. This means that there is a sequence of terms $N_1, N_2, \dots$, such that the values of

$M_i\ N_1, M_i\ N_2, \ldots$ are unbounded, but the values of $M_j\ N_1, M_j\ N_2, \ldots$ 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 $X \neq \mathbb{N}^k$. □

7 Future Work

One can consider λ-calculus enriched by the Y combinator, describing recursion. Then, instead of a finite β-normal form of a term, we may obtain an infinite limit tree, called the Böhm tree. An algorithmic question arises: given a λY-term, how to calculate its "value", that is the "value" of its Böhm tree. In particular, can we decide whether this value is finite? (It turns out that when the value is finite, one can compute it precisely, using standard techniques.) The question starts to become interesting when we can have arbitrary constants of order 0 and 1, instead of just **0** and **1**+, and the value (of a Böhm tree) is defined by a finite tree automaton with counters (e.g. a parity B-automaton), given as a part of the input. (Notice that the value can be finite even when the tree is infinite.) This question (in several variants) were approached only for order 1 (all subterms of the input term are of order 1), that is for pushdown systems [8, 9]; in general it remains open.

References

1. Barendregt, H., Dekkers, W., Statman, R.: Lambda calculus with types. Perspectives in Logic. Cambridge University Press (2013)
2. Parys, P.: On the significance of the collapse operation. In: LICS, IEEE (2012) 521–530
3. Kobayashi, N.: Pumping by typing. In: LICS, IEEE Computer Society (2013) 398–407
4. Schwichtenberg, H.: Definierbare funktionen im lambda-kalkl mit typen. Archiv Logic Grundlagenforsch **17** (1976) 113–114
5. Statman, R.: The typed lambda-calculus is not elementary recursive. Theor. Comput. Sci. **9** (1979) 73–81
6. Zaionc, M.: Word operation definable in the typed lambda-calculus. Theor. Comput. Sci. **52** (1987) 1–14
7. Krivine, J.L.: A call-by-name lambda-calculus machine. Higher-Order and Symbolic Computation **20**(3) (2007) 199–207
8. Lang, M.: Resource-bounded reachability on pushdown systems. Master's thesis, RWTH Aachen (2011)
9. Chatterjee, K., Fijalkow, N.: Infinite-state games with finitary conditions. In Rocca, S.R.D., ed.: CSL. Volume 23 of LIPIcs., Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2013) 181–196