The Weak Lambda Calculus as a Reasonable Machine ⋆

Ugo Dal Lago and Simone Martini *

Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy.

Abstract

We define a new cost model for the call-by-value lambda-calculus satisfying the invariance thesis. That is, under the proposed cost model, Turing machines and the call-by-value lambda-calculus can simulate each other within a polynomial time overhead. The model only relies on combinatorial properties of usual beta-reduction, without any reference to a specific machine or evaluator. In particular, the cost of a single beta reduction is proportional to the difference between the size of the redex and the size of the reduct. In this way, the total cost of normalizing a lambda term will take into account the size of all intermediate results (as well as the number of steps to normal form).

Key words: Lambda calculus, Computational complexity, Invariance thesis, Cost model

1 Introduction

Any computer science student knows that all computational models are extensionally equivalent, each of them characterizing the same class of computable functions. However, the definition of *complexity classes* by means of computational models must take into account several differences between these models, in order to rule out unrealistic assumptions about the cost of the computation steps. It is then usual to consider only *reasonable* models, in such a way that the definition of complexity classes remain invariant when given with reference to any such reasonable model. If polynomial time is the main concern, this requirement takes the form of the *invariance thesis* [15]:

⋆ Extended and revised version of [5], presented at *Computability in Europe 2006*.

* Corresponding author.

Reasonable machines can simulate each other within a polynomially-bounded overhead in time and a constant-factor overhead in space.

Once we agree that Turing machines provide the basic computational model, then many other machine models are proved to satisfy the invariance thesis. Preliminary to the proof of polynomiality of the simulation on a given machine, is the definition of a *cost model*, stipulating when and how much one should account for time and/or space during the computation. For some machines (e.g., Turing machines) this cost model is obvious; for others it is much less so. An example of the latter kind is the type-free lambda-calculus, where there is not a clear notion of *constant time* computational step, and it is even less clear how one should count for consumed space.

The idea of counting the number of beta-reductions [8] is just too naïve, because beta-reduction is inherently too complex to be considered as an atomic operation, at least if we stick to explicit representations of lambda terms. Indeed, in a beta step

$$(\lambda x.M)N \to M\{x/N\},$$

there can be as many as $|M|$ occurrences of x inside M. As a consequence, $M\{x/N\}$ can be as big as $|M||N|$. As an example, consider the term $\underline{n}\ \underline{2}$, where $\underline{n} \equiv \lambda x.\lambda y.x^n y$ is the Church numeral for n. Under innermost reduction this term reduces to normal form in $3n-1$ beta steps, but there is an exponential gap between this quantity and the time needed to write the normal form, that is $\underline{2^n}$. Under outermost reduction, however, the normal form is reached in an exponential number of beta steps. This simple example shows that taking the number of beta steps to normal form as the cost of normalization is at least problematic. Which strategy should we choose [1] ? How do we account for the size of intermediate (and final) results?

Clearly, a viable option consists in defining the cost of reduction as the time needed to normalize a term by another reasonable abstract machine, e.g. a Turing machine. However, in this way we cannot compute the cost of reduction from the structure of the term, and, as a result, it is difficult to compute the cost of normalization for particular terms or for classes of terms. Another invariant cost model is given by the actual cost of outermost (normal order) evaluation, naively implemented [11]. Despite its invariance, it is a too generous cost model (and in its essence not much different from the one that counts the numbers of steps needed to normalize a term on a Turing machine). What is needed is a machine-independent, *parsimonious*, and invariant cost model. Despite some attempts [9,11,12] (which we will discuss shortly), a cost model

[1] Observe that we cannot take the length of the longest reduction sequence, because in several cases this would involve too much useless work. Indeed, there are non-strongly-normalizing terms that can be normalized in one single step, and yet do not admit a longest finite reduction sequence (e.g., $F\ (\Delta\ \Delta)$).

of this kind has not appeared yet.

To simplify things, we attack in this paper the problem for the weak call-by-value lambda-calculus, where we do not reduce under an abstraction and we always fully evaluate an argument before firing a beta redex. Although simple, it is a calculus of paramount importance, since it is the reduction model of any call-by-value functional programming language. For this calculus we define a new, machine-independent cost model and we prove that it satisfies the invariance thesis for time. The proposed cost model only relies on combinatorial properties of usual beta-reduction, without any reference to a specific machine or evaluator. The basic idea is to let the cost of performing a beta-reduction step depend on the size of the involved terms. In particular, the cost of $M \to N$ will be related to the *difference* $|N| - |M|$. In this way, the total cost of normalizing a lambda term will take into account the size of all intermediate results (as well as the number of steps to normal form). The last section of the paper will apply this cost model to the combinatory algebra of closed lambda-terms, to establish some results needed in [4]. We remark that in this algebra the universal function (which maps two terms M and N to the normal form of MN) adds only a *constant* overhead to the time needed to normalize MN. This result, which is almost obvious when viewed from the perspective of lambda-calculus, is something that cannot be obtained in the realm of Turing machines.

1.1 Previous Work

The two main attempts to define an invariant cost model share the reference to optimal lambda reduction à la Lévy [13], a parallel strategy minimizing the number of (parallel) beta steps (see [2]).

Frandsen and Sturtivant [9] propose a cost model essentially based on the number of parallel beta steps to normal form. Their aim is to propose a measure of efficiency for functional programming language implementations. They show how to simulate Turing machines in the lambda calculus with a polynomial overhead. However, the paper does not present any evidence on the existence of a polynomial simulation in the other direction. As a consequence, it is not known whether their proposal is invariant.

More interesting contributions come from the literature of the nineties on optimal lambda reduction. Lamping [10] was the first to operationally present this strategy as a graph rewriting procedure. The interest of this technique for our problem stems from the fact that a single beta step is decomposed into several elementary steps, allowing for the duplication of the argument, the computation of the levels of nesting inside abstractions, and additional book-

keeping work. Since any such elementary step is realizable on a conventional machine in constant time, Lamping's algorithm provides a theoretical basis for the study of complexity of a single beta step. Lawall and Mairson [11] give results on the efficiency of optimal reduction algorithms, highlighting the so-called bookkeeping to be the bottleneck from the point ot view of complexity. A consequence of Lawall and Mairson's work is evidence on the inadequacy of the cost models proposed by Frandsen and Sturtivant and by Asperti [1], at least from the point of view of the invariance thesis. In subsequent work [12], Lawall and Mairson proposed a cost model for the lambda calculus based on Lévy's labels. They further proved that Lamping's *abstract* algorithm satisfies the proposed cost model. This, however, does not imply by itself the existence of an algorithm normalizing *any* lambda term with a polynomial overhead (on the proposed cost). Moreover, studying the dynamic behaviour of Lévy labels is clearly more difficult than dealing directly with the number of beta-reduction steps.

1.2 Research Directions

The results of this paper should be seen as a first step of a broader project to study invariant cost models for several λ-calculi. In particular, we have evidence that for some calculi and under certain conditions we can take *the number of beta reduction steps* as an invariant cost model. Indeed, take the weak call-by-value calculus and suppose you are allowed to exploit sharing during reduction by adopting implicit representations of lambda-terms. We conjecture that the normal form of any term M can be computed in time $O(p(|M|, n))$, where p is a fixed polynomial and n is the number of beta steps to normal form. This result crucially depends on the ability of exploiting sharing and cannot be obtained when working with explicit representations of lambda-terms. But at this point the question is the following: should we take into account the time needed to produce the explicit normal form from its implicit representation?

Moreover, we are confident that simple cost models could be defined for the head linear reduction [6] and for the weak calculus with sharing [3].

2 Syntax

The language we study is the pure untyped lambda-calculus endowed with weak (that is, we never reduce under an abstraction) and call-by-value reduction. Constants are not needed, since usual data structures can be encoded in the pure fragment of the calculus, as we are going to detail in Section 4.

Adding constants would bring the calculus even closer to actual functional programming, where we are interested in the complexity of closed programs applied to data of base types.

Definition 1 *The following definitions are standard:*

- *Terms are defined as follows:*

$$M ::= x \mid \lambda x.M \mid MM.$$

  Λ *denotes the set of all lambda terms.*
- *Values are defined as follows:*

$$V ::= x \mid \lambda x.M.$$

  Ξ *denotes the set of all closed values.*
- *Call-by-value reduction is denoted by* $\rightarrow$ *and is obtained by closing the rule* $(\lambda x.M)V \rightarrow M\{V/x\}$ *under all applicative contexts. Here* M *ranges over terms, while* V *ranges over values.*
- *The length* $|M|$ *of* M *is the number of symbols in* M*.*

Following [14] we consider this system as a complete calculus and not as a mere strategy for the usual lambda-calculus. Indeed, respective sets of normal forms are different. Moreover, the relation $\rightarrow$ is not deterministic although, as we are going to see, this non-determinism is completely harmless.

The way we have defined beta-reduction implies a strong correspondence between values and closed normal forms:

Lemma 1 *Every value is a normal form and every closed normal form is a value.*

Proof. By definition, every value is a normal form, because reduction is weak and, as a consequence, every abstraction is a normal form. For the other direction, we have to prove that if M is a closed normal form, then M is a value. We proceed by induction on M:

- If M is a variable, it is not closed.
- If M is an abstraction then, by definition, it is a value.
- If M is an application NL and M is a closed normal form, then both N and L are closed and in normal form. By induction hypothesis, both N and L are closed values. As a consequence, M cannot be a normal form, because any closed value is an abstraction.

This concludes the proof. □

The prohibition to reduce under abstraction enforces a strong notion of confluence, the so-called one-step diamond property, which instead fails in the usual lambda-calculus.

Proposition 1 (Diamond Property) *If $M \to N$ and $M \to L$ then either $N \equiv L$ or there is P such that $N \to P$ and $L \to P$.*

Proof. By induction on the structure of M. Clearly, M cannot be a variable nor an abstraction so $M \equiv QR$. We can distinguish two cases:

- If M is a redex $(\lambda x.T)R$ (where R is a value), then $N \equiv L \equiv T\{x/R\}$, because we cannot reduce under lambdas.
- If M is not a redex, then reduction must take place inside Q or R. We can distinguish four sub-cases:
  - If $N \equiv TR$ and $L \equiv UR$, where $Q \to T$ and $Q \to U$, then we can apply the induction hypothesis.
  - Similarly, if $R \to T$ and $R \to U$, where $N \equiv QT$ and $L \equiv QU$, then we can apply the induction hypothesis.
  - If $N \equiv QT$ and $L \equiv UR$, where $R \to T$ and $Q \to U$, then $N \to UT$ and $L \to UT$.
  - Similarly, if $N \equiv UR$ and $L \equiv QT$, where $R \to T$ and $Q \to U$, then $N \to UT$ and $L \to UT$.

This concludes the proof. □

As an easy corollary of Proposition 1 we get an equivalence between all normalization strategies— once again a property which does not hold in the ordinary lambda-calculus. This is a well-known consequence of the diamond property:

Corollary 1 (Strategy Equivalence) *M has a normal form iff M is strongly normalizing.*

Proof. Observe that, by Proposition 1, if M is diverging and $M \to N$, then N is diverging, too. Indeed, if $M \equiv M_0 \to M_1 \to M_2 \to \ldots$, then we can build a sequence $N \equiv N_0 \to N_1 \to N_2 \to \ldots$ in a coinductive way:

- If $M_1 \equiv N$, then we define N_i to be M_{i+1} for every $i \geq 1$.
- If $M_1 \not\equiv N$ then by proposition 1 there is N_1 such that $M_1 \to N_1$ and $N_0 \to N_1$.

Now, we prove by induction on n that if $M \to^n N$, with N normal form, then M is strongly normalizing. If $n = 0$, then M is a normal form, then strongly normalizing. If $n \geq 1$, assume, by way of contraddiction, that M is not strongly normalizing. Let L be a term such that $M \to L \to^{n-1} N$. By the above observation, L cannot be strongly normalizing, but this goes against the induction hypothesis. This concludes the proof. □

But we can go even further: in this setting, the number of beta-steps to the normal form does not depend on the evaluation strategy.

Lemma 2 (Parametrical Diamond Property) *If $M \to^n N$ and $M \to^m L$ then there is a term P such that $N \to^l P$ and $L \to^k P$ where $l \leq m$, $k \leq n$*

and $n + l = m + k$.

Proof. We will proceed by induction on $n + m$. If $n + m = 0$, then P will be $N \equiv L \equiv M$. If $n + m > 0$ but either $n = 0$ or $m = 0$, the thesis easily follows. So, we can assume both $n > 0$ and $m > 0$. Let now Q and R be terms such that $M \to Q \to^{n-1} N$ and $M \to R \to^{m-1} L$. From Proposition 1, we can distinguish two cases:

- $Q \equiv R$, By induction hypothesis, we know there is T such that $N \to^{l} T$ and $L \to^{k} T$, where $l \leq m - 1 \leq m$, and $k \leq n - 1 \leq n$. Moreover $(n - 1) + l = (m - 1) + k$, which yields $n + l = m + k$.
- There is T with $Q \to T$ and $R \to T$. By induction hypothesis, there are two terms U, W such that $T \to^{i} U$, $T \to^{j} W$, $N \to^{p} U$, $L \to^{q} W$, where $i \leq n - 1$, $j \leq m - 1$, $p, q \leq 1$, $n - 1 + p = 1 + i$ and $m - 1 + q = 1 + j$. By induction hypothesis, there is P such that $U \to^{r} P$ and $W \to^{s} P$, where $r \leq j$, $s \leq i$ and $r + i = s + j$. But, summing up, this implies

$$
\begin{array}{rcl}
N \to^{p+r} P & & \\
L \to^{q+s} P & & \\
p + r & \leq & 1 + j \leq 1 + m - 1 = m \\
q + s & \leq & 1 + i \leq 1 + n - 1 = n \\
p + r + n & = & (n - 1 + p) + 1 + r = 1 + i + 1 + r \\
 & = & 2 + r + i = 2 + s + j = 1 + 1 + j + s \\
 & = & 1 + m - 1 + q + s = q + s + n.
\end{array}
$$

This concludes the proof. □

The following result summarizes what we have obtained so far:

Proposition 2 *For every term M, there are at most one normal form N and one integer n such that $M \to^{n} N$.*

Proof. Suppose $M \to^{n} N$ and $M \to^{m} L$, with N and L normal forms. Then, by lemma 2, there are P, k, l such that $N \to^{l} P$ and $L \to^{k} P$ and $n+l = m+k$. But since N and L are normal forms, $P \equiv N$, $P \equiv L$ and $l = k = 0$, which yields $N \equiv L$ and $n = m$. □

Given a term M, the number of steps needed to rewrite M to its normal form (if the latter exists) is uniquely determined. As we sketched in the introduction, however, this cannot be considered as the cost of normalizing M, at least if an explicit representation of the normal form is to be produced as part of the computational process.

3 An Abstract Time Measure

We can now define an abstract time measure and prove some of its properties. The rest of the paper is devoted to proving the invariance of the calculus with respect to this computational model.

Intuitively, every beta-step will be endowed with a positive integer cost bounding the difference (in size) between the reduct and the redex. The sum of costs of all steps to normal form gives the cost of normalizing a term.

Definition 2 • *Concatenation of $\alpha, \beta \in \mathbb{N}^*$ is simply denoted as $\alpha\beta$.*

- $\twoheadrightarrow$ *will denote a subset of $\Lambda \times \mathbb{N}^* \times \Lambda$. In the following, we will write $M \overset{\alpha}{\twoheadrightarrow} N$ standing for $(M, \alpha, N) \in \twoheadrightarrow$. The definition of $\twoheadrightarrow$ (in SOS-style) is the following:*

$$\frac{}{M \overset{\varepsilon}{\twoheadrightarrow} M} \qquad \frac{M \to N \quad n = \max\{1, |N| - |M|\}}{M \overset{(n)}{\twoheadrightarrow} N} \qquad \frac{M \overset{\alpha}{\twoheadrightarrow} N \quad N \overset{\beta}{\twoheadrightarrow} L}{M \overset{\alpha\beta}{\twoheadrightarrow} L}$$

  Observe we charge $\max\{1, |N| - |M|\}$ for every step $M \to N$. In this way, the cost of a beta-step will always be positive.
- *Given $\alpha = (n_1, \ldots, n_m) \in \mathbb{N}^*$, define $||\alpha|| = \sum_{i=1}^{m} n_i$.*

Observe that $M \overset{\alpha}{\twoheadrightarrow} N$ iff $M \to^{|\alpha|} N$, where $|\alpha|$ is the length of α as a sequence of natural numbers. This can easily be proved by induction on the derivation of $M \overset{\alpha}{\twoheadrightarrow} N$.

In principle, there could be M, N, α, β such that $M \overset{\alpha}{\twoheadrightarrow} N$, $M \overset{\beta}{\twoheadrightarrow} N$ and $||\alpha|| \neq ||\beta||$. The confluence properties we proved in the previous section, however, can be lifted to this new notion of weighted reduction. First of all, the diamond property and its parameterized version can be reformulated as follows:

Proposition 3 (Diamond Property Revisited) *If $M \overset{(n)}{\twoheadrightarrow} N$ and $M \overset{(m)}{\twoheadrightarrow} L$, then either $N \equiv L$ or there is P such that $N \overset{(m)}{\twoheadrightarrow} P$ and $L \overset{(n)}{\twoheadrightarrow} P$.*

Proof. We can proceed as in Proposition 1. Observe that if $M \overset{\alpha}{\twoheadrightarrow} N$, then $ML \overset{\alpha}{\twoheadrightarrow} NL$ and $LM \overset{\alpha}{\twoheadrightarrow} LN$. We go by induction on the structure of M. Clearly, M cannot be a variable nor an abstraction so $M \equiv QR$. We can distinguish five cases:

- If $Q \equiv \lambda x.T$ and R is a value, then $N \equiv L \equiv T\{x/R\}$, because R is a variable or an abstraction.
- If $N \equiv TR$ and $L \equiv UR$, where $Q \overset{(n)}{\twoheadrightarrow} T$ and $Q \overset{(m)}{\twoheadrightarrow} U$, then we can apply the induction hypothesis, obtaining that $T \overset{(m)}{\twoheadrightarrow} W$ and $U \overset{(n)}{\twoheadrightarrow} W$. This, in

turn, implies $N \overset{(m)}{\twoheadrightarrow} WR$ and $L \overset{(n)}{\twoheadrightarrow} WR$.

- Similarly, if $N \equiv QT$ and $L \equiv QU$, where $R \overset{(n)}{\twoheadrightarrow} T$ and $R \overset{(m)}{\twoheadrightarrow} U$, then we can apply the induction hypothesis.
- If $N \equiv QT$ and $L \equiv UR$, where $R \overset{(n)}{\twoheadrightarrow} T$ and $Q \overset{(m)}{\twoheadrightarrow} U$, then $N \overset{(m)}{\twoheadrightarrow} UT$ and $L \overset{(n)}{\twoheadrightarrow} UT$.
- Similarly, if $N \equiv UR$ and $L \equiv QT$, where $R \overset{(n)}{\twoheadrightarrow} T$ and $Q \overset{(m)}{\twoheadrightarrow} U$, then $N \overset{(m)}{\twoheadrightarrow} UT$ and $L \overset{(n)}{\twoheadrightarrow} UT$.

This concludes the proof. □

Lemma 3 (Parametrical Diamond Property Revisited) *If $M \overset{\alpha}{\twoheadrightarrow} N$ and $M \overset{\beta}{\twoheadrightarrow} L$, then there is a term P such that $N \overset{\gamma}{\twoheadrightarrow} P$ and $L \overset{\delta}{\twoheadrightarrow} P$ where $||\alpha\gamma|| = ||\beta\delta||$.*

Proof. We proceed by induction on $\alpha\beta$. If $\alpha = \beta = \varepsilon$, then P will be $N \equiv L \equiv M$. If $\alpha\beta \neq \varepsilon$ but either $\alpha = \varepsilon$ or $\beta = \varepsilon$, the thesis easily follows. So, we can assume both $\alpha \neq \varepsilon$ and $\beta \neq \varepsilon$. Let now Q and R be such that $M \overset{(n)}{\twoheadrightarrow} Q \overset{\rho}{\twoheadrightarrow} N$ and $M \overset{(m)}{\twoheadrightarrow} R \overset{\delta}{\twoheadrightarrow} L$. From Proposition 3, we can distinguish two cases:

- $Q \equiv R$ (and $m = n$). By induction hypothesis, we know there is T such that $N \overset{\gamma}{\twoheadrightarrow} T$ and $L \overset{\delta}{\twoheadrightarrow} T$, where $||\rho\gamma|| = ||\sigma\delta||$, which yields $||\alpha\gamma|| = ||\beta\delta||$.
- There is T with $Q \overset{(m)}{\twoheadrightarrow} T$ and $R \overset{(n)}{\twoheadrightarrow} T$. By induction hypothesis, there are two terms U, W such that $T \overset{\xi}{\twoheadrightarrow} U$, $T \overset{\eta}{\twoheadrightarrow} W$, $N \overset{\theta}{\twoheadrightarrow} U$, $L \overset{\mu}{\twoheadrightarrow} W$, where $||\rho\theta|| = ||(m)\xi||$ and $||\sigma\mu|| = ||(n)\eta||$. By induction hypothesis, there is P such that $U \overset{\nu}{\twoheadrightarrow} P$ and $W \overset{\tau}{\twoheadrightarrow} P$, where $||\xi\nu|| = ||\eta\tau||$. But, summing up, this implies

$$\begin{aligned} N &\overset{\theta\nu}{\twoheadrightarrow} P \\ L &\overset{\eta\tau}{\twoheadrightarrow} P \\ ||\alpha\theta\nu|| &= ||(n)\rho\theta\nu|| = ||(n)(m)\xi\nu|| = \\ &= ||(m)(n)\xi\nu|| = ||(m)(n)\eta\tau|| = ||(m)\sigma\mu\tau|| = \\ &= ||\beta\mu\tau||. \end{aligned}$$

This concludes the proof. □

As a consequence of the previous two lemmas, for every term M, any sequence α such that $M \overset{\alpha}{\twoheadrightarrow} N$ (where N is a normal form) has the same length $||\alpha||$.

Proposition 4 *For every term M, there are at most one normal form N and one integer n such that $M \overset{\alpha}{\twoheadrightarrow} N$ and $||\alpha|| = n$.*

Proof. Suppose $M \stackrel{\alpha}{\to} N$ and $M \stackrel{\beta}{\to} L$, with N and L normal forms. Then, by Lemma 2, there are P, γ, δ such that $N \stackrel{\gamma}{\to} P$ and $L \stackrel{\delta}{\to} P$ and $||\alpha\gamma|| = ||\beta\delta||$. But since N and L are normal forms, $P \equiv N$, $P \equiv L$ and $\gamma = \delta = \varepsilon$, which yields $N \equiv L$ and $||\alpha|| = ||\beta||$. □

We are now ready to define the abstract time measure which is the core of the paper.

Definition 3 (Difference cost model) *If $M \stackrel{\alpha}{\to} N$, where N is a normal form, then $Time(M)$ is $||\alpha|| + |M|$. If M diverges, then $Time(M)$ is infinite.*

Observe that this is a good definition, in view of Proposition 4. In other words, showing $M \stackrel{\alpha}{\to} N$ suffices to prove $Time(M) = ||\alpha|| + |M|$. This will be particularly useful in the following section.

As an example, consider again the term $\underline{n}\ \underline{2}$ we discussed in the introduction. It reduces to normal form in one step, because we do not reduce under the abstraction. To force reduction, consider $E \equiv \underline{n}\ \underline{2}\ x$, where x is a (free) variable; E reduces to

$$F \equiv \lambda y_n.(\lambda y_{n-1} \ldots (\lambda y_2.(\lambda y_1.x^2 y_1)^2 y_2)^2 \ldots)^2 y_n$$

in $\Theta(n)$ beta steps. However, $Time(E) = \Theta(2^n)$, since at any step the size of the term is duplicated. Indeed, the size of F is exponential in n.

4 Simulating Turing Machines

In this and the following sections we will show that the difference cost model satisfies the polynomial invariance thesis. The present section shows how to encode Turing machines into the lambda calculus.

The first thing we need to encode is a form of recursion. We denote by H the term MM, where $M \equiv \lambda x.\lambda f.f(\lambda z.xxfz)$. H is a call-by-value fixed-point operator: for every value N, there is α such that

$$HN \stackrel{\alpha}{\to} N(\lambda z.HNz),$$
$$||\alpha|| = O(|N|).$$

The lambda term H provides the necessary computational expressive power to encode the whole class of computable functions.

The simplest objects we need to encode in the lambda-calculus are finite sets.

Elements of any finite set $A = \{a_1, \ldots, a_n\}$ can be encoded as follows:

$$\ulcorner a_i \urcorner^A \equiv \lambda x_1. \ldots . \lambda x_n . x_i \ .$$

Notice that the above encoding induces a total order on A such that $a_i \leq a_j$ iff $i \leq j$.

Other useful objects are finite strings over an arbitrary alphabet, which will be encoded using a scheme attributed to Scott [16]. Let $\Sigma = \{a_1, \ldots, a_n\}$ be a finite alphabet. A string in $s \in \Sigma^*$ can be represented by a value $\ulcorner s \urcorner^{\Sigma^*}$ as follows, by induction on s:

$$\begin{aligned} \ulcorner \varepsilon \urcorner^{\Sigma^*} &\equiv \lambda x_1. \ldots . \lambda x_n . \lambda y . y \ , \\ \ulcorner a_i u \urcorner^{\Sigma^*} &\equiv \lambda x_1. \ldots . \lambda x_n \lambda y . x_i \ulcorner u \urcorner^{\Sigma^*} . \end{aligned}$$

Observe that representations of symbols in Σ and strings in Σ^* depend on the cardinality of Σ. In other words, if $u \in \Sigma^*$ and $\Sigma \subset \Delta$, $\ulcorner u \urcorner^{\Sigma^*} \neq \ulcorner u \urcorner^{\Delta^*}$. Besides data, we want to be able to encode functions between them. In particular, the way we have defined numerals lets us concatenate two strings in linear time in the underlying lambda calculus. In some situations, it is even desirable to concatenate a string with the reversal of another string.

Lemma 4 *Given a finite alphabet Σ, there are terms $AC(\Sigma)$, $AS(\Sigma)$ and $AR(\Sigma)$ such that for every $a \in \Sigma$ and $u, v \in \Sigma^*$ there are α, β, γ such that*

$$\begin{aligned} AC(\Sigma) \ulcorner a \urcorner^{\Sigma} \ulcorner u \urcorner^{\Sigma^*} &\overset{\alpha}{\twoheadrightarrow} \ulcorner au \urcorner^{\Sigma^*}, \\ AS(\Sigma) \ulcorner u \urcorner^{\Sigma^*} \ulcorner v \urcorner^{\Sigma^*} &\overset{\beta}{\twoheadrightarrow} \ulcorner uv \urcorner^{\Sigma^*}, \\ AR(\Sigma) \ulcorner u \urcorner^{\Sigma^*} \ulcorner v \urcorner^{\Sigma^*} &\overset{\gamma}{\twoheadrightarrow} \ulcorner u^r v \urcorner^{\Sigma^*}, \end{aligned}$$

and

$$\begin{aligned} ||\alpha|| &= O(1), \\ ||\beta|| &= O(|u|), \\ ||\gamma|| &= O(|u|). \end{aligned}$$

Proof. The three terms are defined as follows:

$$
\begin{aligned}
AC(\Sigma) &\equiv \lambda a.\lambda u.aM_1 \dots M_{|\Sigma|}u \\
&\quad \text{where for any } i,\ M_i \equiv \lambda u.\lambda x_1.\dots.\lambda x_{|\Sigma|}.\lambda w.x_i u; \\
AS(\Sigma) &\equiv H(\lambda x.\lambda u.\lambda v.uN_1 \dots N_{|\Sigma|}(\lambda w.w)v) \\
&\quad \text{where for any } i,\ N_i \equiv \lambda u.\lambda v.(\lambda h.\lambda x_1.\dots.\lambda x_{|\Sigma|}.\lambda g.x_i h)(xuv); \\
AR(\Sigma) &\equiv H(\lambda x.\lambda u.\lambda v.uP_1 \dots P_{|\Sigma|}(\lambda w.w)v) \\
&\quad \text{where for any } i,\ P_i \equiv \lambda u.\lambda v.xu(\lambda x_1.\dots.\lambda x_{|\Sigma|}.\lambda h.x_i v).
\end{aligned}
$$

Observe that

$$
\begin{aligned}
AC(\Sigma)\ulcorner a_i \urcorner^{\Sigma}\ulcorner u \urcorner^{\Sigma^*} &\overset{(1,1)}{\twoheadrightarrow} \ulcorner a_i \urcorner^{\Sigma} M_1 \dots M_{|\Sigma|}\ulcorner u \urcorner^{\Sigma^*} \\
&\overset{\alpha}{\twoheadrightarrow} M_i\ulcorner u \urcorner^{\Sigma^*} \overset{1}{\twoheadrightarrow} \ulcorner a_i u \urcorner^{\Sigma^*},
\end{aligned}
$$

where α does not depend on u. Now, let R_i be $N_i\{\lambda z.AS(\Sigma)z/x\}$. Then, we can proceed by induction:

$$
\begin{aligned}
AS(\Sigma)\ulcorner \varepsilon \urcorner^{\Sigma^*}\ulcorner v \urcorner^{\Sigma^*} &\overset{\alpha}{\twoheadrightarrow} (\lambda u.\lambda v.uR_1 \dots R_{|\Sigma|}(\lambda w.w)v)\ulcorner \varepsilon \urcorner^{\Sigma^*}\ulcorner v \urcorner^{\Sigma^*} \\
&\overset{(1,1)}{\twoheadrightarrow} \ulcorner \varepsilon \urcorner^{\Sigma^*} R_1 \dots R_{|\Sigma|}(\lambda w.w)\ulcorner v \urcorner^{\Sigma^*} \\
&\overset{\beta}{\twoheadrightarrow} (\lambda w.w)\ulcorner v \urcorner^{\Sigma^*} \overset{(1)}{\twoheadrightarrow} \ulcorner v \urcorner^{\Sigma^*}; \\
AS(\Sigma)\ulcorner a_i u \urcorner^{\Sigma^*}\ulcorner v \urcorner^{\Sigma^*} &\overset{\alpha}{\twoheadrightarrow} (\lambda u.\lambda v.uR_1 \dots R_{|\Sigma|}(\lambda w.w)v)\ulcorner a_i u \urcorner^{\Sigma^*}\ulcorner v \urcorner^{\Sigma^*} \\
&\overset{(1,1)}{\twoheadrightarrow} \ulcorner a_i u \urcorner^{\Sigma^*} R_1 \dots R_{|\Sigma|}(\lambda w.w)\ulcorner v \urcorner^{\Sigma^*} \\
&\overset{\gamma}{\twoheadrightarrow} R_i\ulcorner u \urcorner^{\Sigma}\ulcorner v \urcorner^{\Sigma^*} \\
&\overset{(1,1,1)}{\twoheadrightarrow} (\lambda h.\lambda x_1.\dots.\lambda x_{|\Sigma|}.\lambda g.x_i h)(AS(\Sigma)\ulcorner u \urcorner^{\Sigma}\ulcorner v \urcorner^{\Sigma^*}) \\
&\overset{\delta}{\twoheadrightarrow} (\lambda h.\lambda x_1.\dots.\lambda x_{|\Sigma|}.\lambda g.x_i h)\ulcorner uv \urcorner^{\Sigma^*} \\
&\overset{(1)}{\twoheadrightarrow} \lambda x_1.\dots.\lambda x_{|\Sigma|}.\lambda g.x_i\ulcorner uv \urcorner^{\Sigma^*} \\
&\equiv \ulcorner a_i uv \urcorner^{\Sigma^*},
\end{aligned}
$$

where α, β, γ do not depend on u and v. Finally, let Q_i be $P_i\{\lambda z.AR(\Sigma)z/x\}$. Then, we can proceed by induction:

$$
\begin{aligned}
AR(\Sigma)\ulcorner\varepsilon^{\urcorner\Sigma^*}\ulcorner v^{\urcorner\Sigma^*} &\overset{\alpha}{\twoheadrightarrow} (\lambda u.\lambda v.uQ_1\dots Q_{|\Sigma|}(\lambda w.w)v)\ulcorner\varepsilon^{\urcorner\Sigma^*}\ulcorner v^{\urcorner\Sigma^*}\\
&\overset{(1,1)}{\twoheadrightarrow} \ulcorner\varepsilon^{\urcorner\Sigma^*}Q_1\dots Q_{|\Sigma|}(\lambda w.w)\ulcorner v^{\urcorner\Sigma^*}\\
&\overset{\beta}{\twoheadrightarrow} (\lambda w.w)\ulcorner v^{\urcorner\Sigma^*} \overset{(1)}{\twoheadrightarrow} \ulcorner v^{\urcorner\Sigma^*};\\
AR(\Sigma)\ulcorner a_i u^{\urcorner\Sigma^*}\ulcorner v^{\urcorner\Sigma^*} &\overset{\alpha}{\twoheadrightarrow} (\lambda u.\lambda v.uQ_1\dots Q_{|\Sigma|}(\lambda w.w)v)\ulcorner a_i u^{\urcorner\Sigma^*}\ulcorner v^{\urcorner\Sigma^*}\\
&\overset{(1,1)}{\twoheadrightarrow} \ulcorner a_i u^{\urcorner\Sigma^*}Q_1\dots Q_{|\Sigma|}(\lambda w.w)\ulcorner v^{\urcorner\Sigma^*}\\
&\overset{\gamma}{\twoheadrightarrow} Q_i\ulcorner u^{\urcorner\Sigma\ulcorner}v^{\urcorner\Sigma^*}\\
&\overset{(1,1,1)}{\twoheadrightarrow} AR(\Sigma)\ulcorner u^{\urcorner\Sigma\ulcorner}a_i v^{\urcorner\Sigma^*}\\
&\overset{\delta}{\twoheadrightarrow} \ulcorner u^r a_i v^{\urcorner\Sigma^*} \equiv \ulcorner(a_i u)^r v^{\urcorner\Sigma^*},
\end{aligned}
$$

where α, β, γ do not depend on u, v. □

The encoding of a string depends on the underlying alphabet. As a consequence, we also need to be able to convert representations for strings in one alphabet to corresponding representations in a bigger alphabet. This can be done efficiently in the lambda-calculus.

Lemma 5 *Given two finite alphabets Σ and Δ, there are terms $CC(\Sigma,\Delta)$ and $CS(\Sigma,\Delta)$ such that for every $a_0, a_1, \dots, a_n \in \Sigma$ there are α and β with*

$$
\begin{aligned}
CC(\Sigma,\Delta)\ulcorner a_0{}^{\urcorner\Sigma} &\overset{\alpha}{\twoheadrightarrow} \ulcorner u_0{}^{\urcorner\Delta^*};\\
CS(\Sigma,\Delta)\ulcorner a_1\dots a_n{}^{\urcorner\Sigma^*} &\overset{\beta}{\twoheadrightarrow} \ulcorner u_1\dots u_n{}^{\urcorner\Delta^*};\\
\textit{where for any } i,\ u_i &= \begin{cases} a_i \ \textit{if } a_i \in \Delta \\ \varepsilon \quad \textit{otherwise,} \end{cases}
\end{aligned}
$$

and

$$
\begin{aligned}
||\alpha|| &= O(1),\\
||\beta|| &= O(n).
\end{aligned}
$$

Proof. The two terms are defined as follows:

$$CC(\Sigma,\Delta) \equiv \lambda a.aM_1 \dots M_{|\Sigma|}\ ,$$
$$\text{where for any } i,\ M_i \equiv \begin{cases} \ulcorner a_i \urcorner^{\Delta^*} & \text{if } a_i \in \Delta \\ \ulcorner \varepsilon \urcorner^{\Delta^*} & \text{otherwise.} \end{cases}$$
$$CS(\Sigma,\Delta) \equiv H(\lambda x.\lambda u.uN_1 \dots N_{|\Sigma|}(\ulcorner \varepsilon \urcorner^{\Delta^*})),$$
$$\text{where for any } i,\ N_i \equiv \begin{cases} \lambda u.(\lambda v.\lambda x_1.\dots.\lambda x_{|\Delta|}.\lambda h.x_i v)(xu) & \text{if } a_i \in \Delta \\ \lambda u.xu & \text{otherwise.} \end{cases}$$

Observe that

$$\begin{aligned} CC(\Sigma,\Delta)\ulcorner a_i \urcorner^{\Sigma} &\overset{(1)}{\twoheadrightarrow} \ulcorner a_i \urcorner^{\Sigma} M_1 \dots M_{|\Sigma|} \\ &\overset{\alpha}{\twoheadrightarrow} \begin{cases} \ulcorner a_i \urcorner^{\Delta^*} & \text{if } a_i \in \Delta \\ \ulcorner \varepsilon \urcorner^{\Delta^*} & \text{otherwise.} \end{cases} \end{aligned}$$

Let P_i be $N_i\{\lambda z.CS(\Sigma,\Delta)z/x\}$. Then:

$$\begin{aligned} CS(\Sigma,\Delta)\ulcorner \varepsilon \urcorner^{\Sigma^*} &\overset{\alpha}{\twoheadrightarrow} (\lambda u.uP_1 \dots P_{|\Sigma|}\ulcorner \varepsilon \urcorner^{\Delta^*})\ulcorner \varepsilon \urcorner^{\Sigma^*} \\ &\overset{(1)}{\twoheadrightarrow} \ulcorner \varepsilon \urcorner^{\Sigma^*} P_1 \dots P_{|\Sigma|}\ulcorner \varepsilon \urcorner^{\Delta^*} \\ &\overset{\beta}{\twoheadrightarrow} \ulcorner \varepsilon \urcorner^{\Delta^*}; \\ CS(\Sigma,\Delta)\ulcorner a_i u \urcorner^{\Sigma^*} &\overset{\gamma}{\twoheadrightarrow} (\lambda u.uP_1 \dots P_{|\Sigma|}\ulcorner \varepsilon \urcorner^{\Delta^*})\ulcorner a_i u \urcorner^{\Sigma^*} \\ &\overset{(1)}{\twoheadrightarrow} \ulcorner a_i u \urcorner^{\Sigma^*} P_1 \dots P_{|\Sigma|}\ulcorner \varepsilon \urcorner^{\Delta^*} \\ &\overset{\delta}{\twoheadrightarrow} P_i \ulcorner u \urcorner^{\Sigma^*} \\ &\overset{(1,1)}{\twoheadrightarrow} \begin{cases} (\lambda v.\lambda x_1.\dots.\lambda x_{|\Delta|}.\lambda h.x_i v)(CS(\Sigma,\Delta)\ulcorner u \urcorner^{\Sigma^*}) & \text{if } a_i \in \Delta \\ CS(\Sigma,\Delta)\ulcorner u \urcorner^{\Sigma^*} & \text{otherwise,} \end{cases} \end{aligned}$$

where $\alpha, \beta, \gamma, \delta$ do not depend on u. □

A deterministic Turing machine $\mathcal{M}$ is a tuple $(\Sigma, a_{blank}, Q, q_{initial}, q_{final}, \delta)$ consisting of:

- A finite alphabet $\Sigma = \{a_1, \dots, a_n\}$;
- A distinguished symbol $a_{blank} \in \Sigma$, called the *blank symbol*;
- A finite set $Q = \{q_1, \dots, q_m\}$ of *states*;
- A distinguished state $q_{initial} \in Q$, called the *initial state*;
- A distinguished state $q_{final} \in Q$, called the *final state*;
- A partial *transition function* $\delta : Q \times \Sigma \rightharpoonup Q \times \Sigma \times \{\leftarrow, \rightarrow, \downarrow\}$ such that $\delta(q_i, a_j)$ is defined iff $q_i \neq q_{final}$.

A configuration for $\mathcal{M}$ is a quadruple in $\Sigma^* \times \Sigma \times \Sigma^* \times Q$. For example, if $\delta(q_i, a_j) = (q_l, a_k, \leftarrow)$, then $\mathcal{M}$ evolves from (ua_p, a_j, v, q_i) to $(u, a_p, a_k v, q_l)$ (and from $(\varepsilon, a_j, v, q_i)$ to $(\varepsilon, a_{blank}, a_k v, q_l)$). A configuration like (u, a_i, v, q_{final}) is *final* and cannot evolve. Given a string $u \in \Sigma^*$, the *initial configuration* for u is $(\varepsilon, a, v, q_{initial})$ if $u = av$ and $(\varepsilon, a_{blank}, \varepsilon, q_{initial})$ if $u = \varepsilon$. The string corresponding to the final configuration (u, a_i, v, q_{final}) is ua_iv.

A Turing machine $(\Sigma, a_{blank}, Q, q_{initial}, q_{final}, \delta)$ computes the function $f : \Delta^* \to \Delta^*$ (where $\Delta \subseteq \Sigma$) in time $g : \mathbb{N} \to \mathbb{N}$ iff for every $u \in \Delta^*$, the initial configuration for u evolves to a final configuration for $f(u)$ in $g(|u|)$ steps.

A configuration (s, a, v, q) of a machine $\mathcal{M} = (\Sigma, a_{blank}, Q, q_{initial}, q_{final}, \delta)$ is represented by the term

$$\ulcorner(u, a, v, q)\urcorner^{\mathcal{M}} \equiv \lambda x.x \ulcorner u^r \urcorner^{\Sigma^*} \ulcorner a \urcorner^{\Sigma} \ulcorner v \urcorner^{\Sigma^*} \ulcorner q \urcorner^{Q}.$$

We now encode a Turing machine $\mathcal{M} = (\Sigma, a_{blank}, Q, q_{initial}, q_{final}, \delta)$ in the lambda-calculus. Suppose $\Sigma = \{a_1, \ldots, a_{|\Sigma|}\}$ and $Q = \{q_1, \ldots, q_{|Q|}\}$ We proceed by building up three lambda terms:

- First of all, we need to be able to build the initial configuration for u from u itself. This can be done in linear time.
- Then, we need to extract a string from a final configuration for the string. This can be done in linear time, too.
- Most importantly, we need to be able to simulate the transition function of $\mathcal{M}$, i.e. compute a final configuration from an initial configuration (if it exists). This can be done with cost proportional to the number of steps $\mathcal{M}$ takes on the input.

The following three lemmas formalize the above intuitive argument:

Lemma 6 *Given a Turing machine $\mathcal{M} = (\Sigma, a_{blank}, Q, q_{initial}, q_{final}, \delta)$ and an alphabet $\Delta \subseteq \Sigma$ there is a term $I(\mathcal{M}, \Delta)$ such that for every $u \in \Delta^*$, $I(\mathcal{M}, \Delta) \ulcorner u \urcorner^{\Delta^*} \overset{\alpha}{\twoheadrightarrow} \ulcorner C \urcorner^{\mathcal{M}}$ where C is the initial configuration for u and $||\alpha|| = O(|u|)$.*

Proof. $I(\mathcal{M}, \Delta)$ is defined as

$$H(\lambda x.\lambda u.uM_1 \ldots M_{|\Delta|}N),$$

where

$$N \equiv \ulcorner(\varepsilon, a_{blank}, \varepsilon, q_{initial})\urcorner^{\mathcal{M}};$$
$$M_i \equiv \lambda u.(xu)(\lambda u.\lambda a.\lambda v.\lambda q.(\lambda w.(\lambda x.xu \ulcorner a_i \urcorner^{\Sigma} wq))(AC(\Sigma)av)).$$

Let P_i be $M_i\{\lambda z.I(\mathcal{M}, \Delta)z/x\}$. Then,

$$
\begin{aligned}
I(\mathcal{M},\Delta)^{\ulcorner}\varepsilon^{\urcorner\Delta^*} &\overset{\alpha}{\twoheadrightarrow} (\lambda u.uP_1\dots P_{|\Delta|}N)^{\ulcorner}\varepsilon^{\urcorner\Delta^*}\\
&\overset{(1)}{\twoheadrightarrow} {}^{\ulcorner}\varepsilon^{\urcorner\Delta^*}P_1\dots P_{|\Delta|}N\\
&\overset{\beta}{\twoheadrightarrow} N \equiv {}^{\ulcorner}(\varepsilon, a_{blank}, \varepsilon, q_{initial})^{\urcorner\mathcal{M}};\\
I(\mathcal{M},\Delta)^{\ulcorner}a_iu^{\urcorner\Delta^*} &\overset{\alpha}{\twoheadrightarrow} (\lambda u.uP_1\dots P_{|\Delta|}N)^{\ulcorner}a_iu^{\urcorner\Delta^*}\\
&\overset{(1)}{\twoheadrightarrow} {}^{\ulcorner}a_iu^{\urcorner\Delta^*}P_1\dots P_{|\Delta|}N\\
&\overset{\beta}{\twoheadrightarrow} P_i{}^{\ulcorner}u^{\urcorner\Delta^*}\\
&\overset{(1)}{\twoheadrightarrow} (I(\mathcal{M},\Delta)^{\ulcorner}u^{\urcorner\Delta^*})(\lambda u.\lambda a.\lambda v.\lambda q.(\lambda w.(\lambda x.xu^{\ulcorner}a_i{}^{\urcorner\Sigma}wq))(AC(\Sigma)av))\\
&\overset{\gamma}{\twoheadrightarrow} {}^{\ulcorner}D^{\urcorner\mathcal{M}}(\lambda u.\lambda a.\lambda v.\lambda q.(\lambda w.(\lambda x.xu^{\ulcorner}a_i{}^{\urcorner\Sigma}wq))(AC(\Sigma)av)),
\end{aligned}
$$

where α, β do not depend on u and D is and initial configuration for u. Clearly

$$
{}^{\ulcorner}D^{\urcorner\mathcal{M}}(\lambda u.\lambda a.\lambda v.\lambda q.(\lambda w.(\lambda x.xu^{\ulcorner}a_i{}^{\urcorner\Sigma}wq))(AC(\Sigma)av)) \overset{\delta}{\twoheadrightarrow} {}^{\ulcorner}E^{\urcorner\mathcal{M}},
$$

where E is an initial configuration for a_iu and δ does not depend on u. □

Lemma 7 *Given a Turing machine $\mathcal{M} = (\Sigma, a_{blank}, Q, q_{initial}, q_{final}, \delta)$ and for every alphabet Δ, there is a term $F(\mathcal{M},\Delta)$ such that for every final configuration C for $u_1\dots u_n$ there is α such that $F(\mathcal{M},\Delta)^{\ulcorner}C^{\urcorner\mathcal{M}} \overset{\alpha}{\twoheadrightarrow} {}^{\ulcorner}v_1\dots v_n{}^{\urcorner\Delta^*}$, $||\alpha|| = O(n)$ and, for any i,*

$$
v_i = \begin{cases} u_i & \text{if } u_i \in \Delta \\ \varepsilon & \text{otherwise.} \end{cases}
$$

Proof. $F(\mathcal{M},\Delta)$ is defined as

$$
\lambda x.x(\lambda u.\lambda a.\lambda v.\lambda q.AR(\Delta)(CS(\Sigma,\Delta)u)(AS(\Sigma)(CC(\Sigma,\Delta)a)(CS(\Sigma,\Delta)v)).
$$

Consider an arbitrary final configuration ${}^{\ulcorner}(u,a,v,q_{final})^{\urcorner\mathcal{M}}$. Then:

$$
\begin{aligned}
& F(\mathcal{M},\Delta)^{\ulcorner}(u,a,v,q_{final})^{\urcorner\mathcal{M}}\\
\overset{(1,1,1,1,1)}{\twoheadrightarrow}\ & AR(\Delta)(CS(\Sigma,\Delta)^{\ulcorner}u^{\urcorner\Sigma^*})(AS(\Sigma)(CC(\Sigma,\Delta)^{\ulcorner}a^{\urcorner\Sigma})(CS(\Sigma,\Delta)^{\ulcorner}v^{\urcorner\Sigma^*}))\\
\overset{\alpha}{\twoheadrightarrow}\ & AR(\Delta)({}^{\ulcorner}u^{\urcorner\Delta^*})(AS(\Delta)({}^{\ulcorner}a^{\urcorner\Delta^*})({}^{\ulcorner}v^{\urcorner\Delta^*}))\\
\overset{\beta}{\twoheadrightarrow}\ & AR(\Delta)^{\ulcorner}u^{\urcorner\Delta^*}{}^{\ulcorner}av^{\urcorner\Delta^*}\\
\overset{\gamma}{\twoheadrightarrow}\ & {}^{\ulcorner}u^rav^{\urcorner\Delta^*},
\end{aligned}
$$

where $\alpha = O(|u|+|v|)$, β does not depend on u, v and $\gamma = O(|u|)$. □

Lemma 8 *Given a Turing machine $\mathcal{M} = (\Sigma, a_{blank}, Q, q_{initial}, q_{final}, \delta)$, there is a term $T(\mathcal{M})$ such that for every configuration C,*

- *if D is a final configuration reachable from C in n steps, then there exists α such that $T(\mathcal{M})^\ulcorner C^{\urcorner\mathcal{M}} \stackrel{\alpha}{\twoheadrightarrow} {}^\ulcorner D^{\urcorner\mathcal{M}}$; moreover $||\alpha|| = O(n)$;*
- *the term $T(\mathcal{M})^\ulcorner C^{\urcorner\mathcal{M}}$ diverges if there is no final configuration reachable from C.*

Proof. $T(\mathcal{M})$ is defined as

$$H(\lambda x.\lambda y.y(\lambda u.\lambda a.\lambda v.\lambda q.q(M_1 \dots M_{|Q|})uav)),$$

where, for any i and j:

$$M_i \equiv \lambda u.\lambda a.\lambda v.a(N_i^1 \dots N_i^{|\Sigma|})uv;$$

$$N_i^j \equiv \begin{cases} \lambda u.\lambda v.\lambda x.xu^\ulcorner a_j{}^{\urcorner\Sigma} v^\ulcorner q_i{}^{\urcorner Q} & \text{if } q_i = q_{final} \\ \lambda u.\lambda v.x(\lambda z.zu^\ulcorner a_k{}^{\urcorner\Sigma} v^\ulcorner q_l{}^{\urcorner Q}) & \text{if } \delta(q_i,a_j) = (q_l,a_k,\downarrow) \\ \lambda u.\lambda v.x(uP_1 \dots P_{|\Sigma|} P(AC(\Sigma)^\ulcorner a_k{}^{\urcorner\Sigma} v)^\ulcorner q_l{}^{\urcorner Q}) & \text{if } \delta(q_i,a_j) = (q_l,a_k,\leftarrow) \\ \lambda u.\lambda v.x(vR_1 \dots R_{|\Sigma|} R(AC(\Sigma)^\ulcorner a_k{}^{\urcorner\Sigma} u)^\ulcorner q_l{}^{\urcorner Q}) & \text{if } \delta(q_i,a_j) = (q_l,a_k,\rightarrow); \end{cases}$$

$$P_i \equiv \lambda u.\lambda v.\lambda q.\lambda x.xu^\ulcorner a_i{}^{\urcorner\Sigma} vq;$$
$$P \equiv \lambda v.\lambda q.\lambda x.x^\ulcorner \varepsilon^{\urcorner\Sigma^*} {}^\ulcorner a_{blank}{}^{\urcorner\Sigma} vq;$$
$$R_i \equiv \lambda v.\lambda u.\lambda q.\lambda x.xu^\ulcorner a_i{}^{\urcorner\Sigma} vq;$$
$$R \equiv \lambda u.\lambda q.\lambda x.xu^\ulcorner a_{blank}{}^{\urcorner\Sigma} {}^\ulcorner \varepsilon^{\urcorner\Sigma^*} q.$$

To prove the thesis, it suffices to show that

$$T(\mathcal{M})^\ulcorner C^{\urcorner\mathcal{M}} \stackrel{\beta}{\twoheadrightarrow} T(\mathcal{M})^\ulcorner E^{\urcorner\mathcal{M}},$$

where E is the next configuration reachable from C and β is bounded by a constant independent of C. We need some abbreviations:

$$\forall i.Q_i \equiv M_i\{\lambda z.T(\mathcal{M})z/x\};$$
$$\forall i,j.T_i^j \equiv N_i^j\{\lambda z.T(\mathcal{M})z/x\}.$$

Suppose $C = (u, a_j, v, q_i)$. Then

$$\begin{aligned} T(\mathcal{M})^\ulcorner C^{\urcorner\mathcal{M}} &\stackrel{\gamma}{\twoheadrightarrow} {}^\ulcorner q_i{}^{\urcorner Q} Q_1 \dots Q_{|Q|} {}^\ulcorner u^{\urcorner\Sigma^*} {}^\ulcorner a_j{}^{\urcorner\Sigma} {}^\ulcorner v^{\urcorner\Sigma^*} \\ &\stackrel{\delta}{\twoheadrightarrow} Q_i{}^\ulcorner u^{\urcorner\Sigma^*} {}^\ulcorner a_j{}^{\urcorner\Sigma} {}^\ulcorner v^{\urcorner\Sigma^*} \\ &\stackrel{(1,1,1)}{\twoheadrightarrow} {}^\ulcorner a_j{}^{\urcorner\Sigma} T_i^1 \dots T_i^{|\Sigma|} {}^\ulcorner u^{\urcorner\Sigma^*} {}^\ulcorner v^{\urcorner\Sigma^*} \\ &\stackrel{\rho}{\twoheadrightarrow} T_i^j {}^\ulcorner u^{\urcorner\Sigma^*} {}^\ulcorner v^{\urcorner\Sigma^*}, \end{aligned}$$

where γ, δ, ρ do not depend on C. Now, consider the following four cases, depending on the value of $\delta(q_i, a_j)$:

- If $\delta(q_i, a_j)$ is undefined, then $q_i = q_{final}$ and, by definition $T_i^j \equiv \lambda u.\lambda v.\lambda x.xu^{\ulcorner}a_j{}^{\urcorner\Sigma}v^{\ulcorner}q_i{}^{\urcorner Q}$. As a consequence,

$$\begin{aligned} T_i^{j\ulcorner}u^{\urcorner\Sigma^*\ulcorner}v^{\urcorner\Sigma^*} &\overset{(1,1)}{\twoheadrightarrow} \lambda x.x^{\ulcorner}u^{\urcorner\Sigma^*\ulcorner}a_j{}^{\urcorner\Sigma\ulcorner}v^{\urcorner\Sigma^*\ulcorner}q_i{}^{\urcorner Q} \\ &\equiv {}^{\ulcorner}(u, a_j, v, q_i)^{\urcorner\mathcal{M}}. \end{aligned}$$

- If $\delta(q_i, a_j) = (q_l, a_k, \downarrow)$, then $T_i^j \equiv \lambda u.\lambda v.(\lambda z.T(\mathcal{M})z)(\lambda z.zu^{\ulcorner}a_k{}^{\urcorner\Sigma}v^{\ulcorner}q_l{}^{\urcorner Q})$. As a consequence,

$$\begin{aligned} T_i^{j\ulcorner}u^{\urcorner\Sigma^*\ulcorner}v^{\urcorner\Sigma^*} &\overset{(1,1)}{\twoheadrightarrow} (\lambda z.T(\mathcal{M})z)(\lambda z.z^{\ulcorner}u^{\urcorner\Sigma^*\ulcorner}a_k{}^{\urcorner\Sigma\ulcorner}v^{\urcorner\Sigma^*\ulcorner}q_l{}^{\urcorner Q}) \\ &\overset{(1)}{\twoheadrightarrow} T(\mathcal{M})(\lambda z.z^{\ulcorner}u^{\urcorner\Sigma^*\ulcorner}a_k{}^{\urcorner\Sigma\ulcorner}v^{\urcorner\Sigma^*\ulcorner}q_l{}^{\urcorner Q}) \\ &\equiv T(\mathcal{M})^{\ulcorner}E^{\urcorner\mathcal{M}}. \end{aligned}$$

- If $\delta(q_i, a_j) = (q_l, a_k, \leftarrow)$, then

$$\lambda u.\lambda v.x(uP_1 \dots P_{|\Sigma|}P(AC(\Sigma)^{\ulcorner}a_j{}^{\urcorner\Sigma}v)^{\ulcorner}q_l{}^{\urcorner Q}).$$

  As a consequence,

$$T_i^{j\ulcorner}u^{\urcorner\Sigma^*\ulcorner}v^{\urcorner\Sigma^*} \overset{(1,1)}{\twoheadrightarrow} (\lambda z.T(\mathcal{M})z)({}^{\ulcorner}u^{\urcorner\Sigma^*}P_1 \dots P_{|\Sigma|}P(AC(\Sigma)^{\ulcorner}a_k{}^{\urcorner\Sigma\ulcorner}v^{\urcorner\Sigma^*})^{\ulcorner}q_l{}^{\urcorner Q}).$$

  Now, if u is ε, then

$$\begin{aligned} & (\lambda z.T(\mathcal{M})z)({}^{\ulcorner}u^{\urcorner\Sigma^*}P_1 \dots P_{|\Sigma|}P(AC(\Sigma)^{\ulcorner}a_k{}^{\urcorner\Sigma\ulcorner}v^{\urcorner\Sigma^*})^{\ulcorner}q_l{}^{\urcorner Q}) \\ \overset{\eta}{\twoheadrightarrow}\ & (\lambda z.T(\mathcal{M})z)P(AC(\Sigma)^{\ulcorner}a_k{}^{\urcorner\Sigma\ulcorner}v^{\urcorner\Sigma^*})^{\ulcorner}q_l{}^{\urcorner Q} \\ \overset{\xi}{\twoheadrightarrow}\ & (\lambda z.T(\mathcal{M})z)P({}^{\ulcorner}a_kv^{\urcorner\Sigma^*})^{\ulcorner}q_l{}^{\urcorner Q}) \\ \overset{(1,1)}{\twoheadrightarrow}\ & (\lambda z.T(\mathcal{M})z)^{\ulcorner}(\varepsilon, a_{blank}, a_kv, q_l)^{\urcorner\mathcal{M}} \\ \overset{(1)}{\twoheadrightarrow}\ & T(\mathcal{M})^{\ulcorner}(\varepsilon, a_{blank}, a_kv, q_l)^{\urcorner\mathcal{M}}, \end{aligned}$$

  where η, ξ do not depend on C. If u is ta_p, then

$$\begin{aligned} & (\lambda z.T(\mathcal{M})z)({}^{\ulcorner}u^{r\urcorner\Sigma^*}U_1 \dots U_{|\Sigma|}U(AC(\Sigma)^{\ulcorner}a_k{}^{\urcorner\Sigma\ulcorner}v^{\urcorner\Sigma^*})^{\ulcorner}q_l{}^{\urcorner Q}) \\ \overset{\pi}{\twoheadrightarrow}\ & (\lambda z.T(\mathcal{M})z)U_p{}^{\ulcorner}t^{r\urcorner\Sigma^*}(AC(\Sigma)^{\ulcorner}a_k{}^{\urcorner\Sigma\ulcorner}v^{\urcorner\Sigma^*})^{\ulcorner}q_l{}^{\urcorner Q} \\ \overset{\theta}{\twoheadrightarrow}\ & (\lambda z.T(\mathcal{M})z)U_p{}^{\ulcorner}t^{r\urcorner\Sigma^*}({}^{\ulcorner}a_kv^{\urcorner\Sigma^*})^{\ulcorner}q_l{}^{\urcorner Q}) \\ \overset{(1,1)}{\twoheadrightarrow}\ & (\lambda z.T(\mathcal{M})z)^{\ulcorner}(t, a_p, a_kv, q_l)^{\urcorner\mathcal{M}} \\ \overset{(1,1)}{\twoheadrightarrow}\ & T(\mathcal{M})^{\ulcorner}(t, a_p, a_kv, q_l)^{\urcorner\mathcal{M}}, \end{aligned}$$

  where π, θ do not depend on C.
- The case $\delta(q_i, a_j) = (q_l, a_k, \rightarrow)$ can be treated similarly.

This concludes the proof. □

At this point, we can give the main simulation result:

Theorem 1 *If $f : \Delta^* \to \Delta^*$ is computed by a Turing machine $\mathcal{M}$ in time g, then there is a term $U(\mathcal{M}, \Delta)$ such that for every $u \in \Delta^*$ there is α with $U(\mathcal{M}, \Delta)\ulcorner u \urcorner^{\Delta^*} \stackrel{\alpha}{\twoheadrightarrow} \ulcorner f(u) \urcorner^{\Delta^*}$ and $||\alpha|| = O(g(|u|))$*

Proof. Simply define $U(\mathcal{M}, \Delta) \equiv \lambda x.F(\mathcal{M}, \Delta)(T(\mathcal{M})(I(\mathcal{M}, \Delta)x))$. □

Noticeably, the just described simulation induces a linear overhead: every step of $\mathcal{M}$ corresponds to a constant cost in the simulation, the constant cost not depending on the input but only on $\mathcal{M}$ itself.

5 Evaluating Terms with Turing Machines

We informally describe a Turing machine $\mathcal{R}$ computing the normal form of a given input term, if it exists, and diverging otherwise. If M is the input term, $\mathcal{R}$ takes time $O((Time(M))^4)$.

First of all, let us observe that the usual notation for terms does not take into account the complexity of handling variables, and substitutions. We introduce a notation in the style of deBruijn [7], with binary strings representing occurrences of variables. In this way, terms can be denoted by finite strings in a finite alphabet.

Definition 4 • *The alphabet Θ is $\{\lambda, @, 0, 1, \blacktriangleright\}$.*

- *To each lambda term M we can associate a string $M^\# \in \Theta^+$ in the standard deBruijn way, writing @ for (prefix) application. For example, if $M \equiv (\lambda x.xy)(\lambda x.\lambda y.\lambda z.x)$, then $M^\#$ is $@\lambda@\blacktriangleright 0 \blacktriangleright \lambda\lambda\lambda \blacktriangleright 10$. In other words, free occurrences of variables are translated into $\blacktriangleright$, while bounded occurrences of variables are translated into $\blacktriangleright s$, where s is the binary representation of the deBruijn index for that occurrence.*
- *The* true length *$||M||$ of a term M is the length of $M^\#$.*

Observe that $||M||$ grows more than linearly on $|M|$:

Lemma 9 *For every term M, $||M|| = O(|M| \log |M|)$. There is a sequence $\{M_n\}_{n \in \mathbb{N}}$ such that $|M_n| = \Theta(n)$, while $||M_n|| = \Theta(|M_n| \log |M_n|)$.*

Proof. Consider the following statement: for every M, the string $M^\#$ contains at most $2|M| - 1$ characters from $\{\lambda, @\}$ and at most $|M|$ blocks of characters from $\{0, 1, \blacktriangleright\}$, the length of each of them being at most $1 + \lceil \log_2 |M| \rceil$. Proving that would imply the thesis. We proceed by induction on M:

- If M is a variable x, then $M^\#$ is $\blacktriangleright$. The thesis is satisfied, because $|M| = 1$.
- If M is $\lambda x.N$, then $M^\#$ is λu, where u is obtained from $N^\#$ by replacing

Fig. 1. The status of some tapes after step 1

Preredex	@@
Functional	$\lambda\lambda@@\blacktriangleright 1\blacktriangleright 0\blacktriangleright 0$
Argument	$\lambda\blacktriangleright 0$
Postredex	$\lambda\blacktriangleright 0$

some blocks in the form $\blacktriangleright$ with $\blacktriangleright s$, where $|s|$ is at most $\lceil \log_2 |M| \rceil$. Moreover, any block $\blacktriangleright\ s$ (where s represents n) must be replaced by $\blacktriangleright\ t$ (where t represents $n+1$). As a consequence, the thesis remains satisfied.

- If M is NL, then $M^{\#}$ is $@N^{\#}L^{\#}$ and the thesis remains satisfied.

This proves $||M|| = O(|M| \log |M|)$. For the second part, define

$$M_n \equiv \lambda x.\overbrace{\lambda y.\ldots.\lambda y}^{n \text{ times}}.\ \overbrace{x\ldots x}^{n+1 \text{ times}}.$$

Clearly,

$$M_n^{\#} \equiv \overbrace{\lambda\ldots\lambda}^{n+1 \text{ times}}\ \overbrace{@\blacktriangleright u\ldots@\blacktriangleright u}^{n \text{ times}}\blacktriangleright u,$$

where u is the binary coding of n (so $|u| = \Theta(\log n)$). As a consequence:

$$\begin{aligned} |M| &= 3n+3 = \Theta(n);\\ ||M|| &= |M^{\#}| = 3n+2+(n+1)|u| = \Theta(n\log n). \end{aligned}$$

This concludes the proof. □

The Turing machine $\mathcal{R}$ has nine tapes, expects its input to be in the first tape and writes the output on the same tape. The tapes will be referred to as *Current* (the first one), *Preredex*, *Functional*, *Argument*, *Postredex*, *Reduct*, *StackTerm*, *StackRedex*, *Counter*. $\mathcal{R}$ operates by iteratively performing the following four steps:

1. First of all, $\mathcal{R}$ looks for redexes in the term stored in *Current* (call it M), by scanning it. The functional part of the redex will be put in *Functional* while its argument is copied into *Argument*. Everything appearing before (respectively, after) the redex is copied into *Preredex* (respectively, in *Postredex*). If there is no redex in M, then $\mathcal{R}$ halts. For example, consider the term $(\lambda x.\lambda y.xyy)(\lambda z.z)(\lambda w.w)$ which becomes $@@\lambda\lambda@@\blacktriangleright 1\blacktriangleright 0\blacktriangleright 0\lambda\blacktriangleright 0\lambda\blacktriangleright 0$ in deBruijn notation. Figure 1 summarizes the status of some tapes after this initial step.
2. Then, $\mathcal{R}$ copies the content of *Functional* into *Reduct*, erasing the first occurrence of λ and replacing every occurrence of the bounded variable by the content of *Argument*. In the example, *Reduct* becomes $\lambda@@\lambda\blacktriangleright 0\blacktriangleright 0\blacktriangleright 0$.

Fig. 2. How the stack evolves while processing $@\lambda \blacktriangleright 0\lambda \blacktriangleright 0$

$\underline{@}\lambda \blacktriangleright \lambda \blacktriangleright 0$	$F_@$
$@\underline{\lambda} \blacktriangleright 0\lambda \blacktriangleright 0$	$F_@ A_\lambda$
$@\lambda\underline{\blacktriangleright} 0\lambda \blacktriangleright 0$	$S_@$
$@\lambda \blacktriangleright \underline{0}\lambda \blacktriangleright 0$	$S_@$
$@\lambda \blacktriangleright 0\underline{\lambda} \blacktriangleright 0$	$S_@ A_\lambda$
$@\lambda \blacktriangleright 0\lambda\underline{\blacktriangleright} 0$	ε
$@\lambda \blacktriangleright 0\lambda \blacktriangleright \underline{0}$	ε

3. $\mathcal{R}$ replaces the content of *Current* with the concatenation of *Preredex*, *Reduct* and *Postredex* in this particular order. In the example, *Current* becomes $@\lambda@@\lambda \blacktriangleright 0 \blacktriangleright 0 \blacktriangleright 0\lambda \blacktriangleright 0$, which correctly correspond to $(\lambda y.(\lambda z.z)yy)(\lambda w.w)$.
4. Finally, the content of every tape except *Current* is erased.

Every time the sequence of steps from 1 to 4 is performed, the term M in *Current* is replaced by another term which is obtained from M by performing a normalization step. So, $\mathcal{R}$ halts on M if and only if M is normalizing and the output will be the normal form of M.

Tapes *StackTerm* and *StackRedex* are managed in the same way. They help keeping track of the structure of a term as it is scanned. The two tapes can only contain symbols A_λ, $F_@$ and $S_@$. In particular:
- The symbol A_λ stands for the argument of an abstraction;
- the symbol $F_@$ stands for the first argument of an application;
- the symbol $S_@$ stands for the second argument of an application;

StackTerm and *StackRedex* can only be modified by the usual stack operations, i.e. by pushing and popping symbols from the top of the stack. Anytime a new symbol is scanned, the underlying stack can possibly be modified:
- If @ is read, then $F_@$ must be pushed on the top of the stack.
- If λ is read, then A_λ must be pushed on the top of the stack.
- If $\blacktriangleright$ is read, then symbols $S_@$ and A_λ must be popped from the stack, until we find an occurrence of $F_@$ (which must be popped and replaced by $S_@$) or the stack is empty.

For example, when scanning the term $@\lambda \blacktriangleright 0\lambda \blacktriangleright 0$, the underlying stack evolves as in Figure 2 (the symbol currently being read is underlined).

Now, consider an arbitrary iteration step, where M is reduced to N. We claim that the steps 1 to 4 can all be performed in $O((||M|| + ||N||)^2)$. The following is an informal argument.

- Step 1 can be performed with the help of auxiliary tapes *StackTerm* and *StackRedex*. *Current* is scanned with the help of *StackTerm*. As soon as $\mathcal{R}$ encounter a λ symbol in *Current*, it treats the subterm in a different way, copying it into *Functional* with the help of *StackRedex*. When the subterm has been completely processed (i.e. when *StackRedex* becomes empty), the machine can verify whether or not it is the functional part of a redex. It suffices to check the topmost symbol of *StackTerm* and the next symbol in *Current*. We are in presence of a redex only if the topmost symbol of *StackTerm* is $F_@$ and the next symbol in *Current* is either λ or $\blacktriangleright$. Then, $\mathcal{R}$ proceeds as follows:
  - If we are in presence of a redex, then the subterm corresponding to the argument is copied into *Argument*, with the help of *StackRedex*;
  - Otherwise, the content of *Functional* is moved to *Preredex* and *Functional* is completely erased.
- Step 2 can be performed with the help of *StackRedex* and *Counter*. Initially, $\mathcal{R}$ simply writes 0 into *Counter*, which keeps track of λ-nesting depth of the current symbol (in binary notation) while scanning *Functional*. *StackRedex* is used in the usual way. Whenever we push A_λ into *StackRedex*, *Counter* is incremented by 1, while it is decremented by 1 whenever A_λ is popped from *StackRedex*. While scanning *Functional*, $\mathcal{R}$ copies everything into *Reduct*. If $\mathcal{R}$ encounters a $\blacktriangleright$, it compares the binary string following it with the actual content of *Counter*. Then it proceeds as follows:
  - If they are equal, $\mathcal{R}$ copies to *Reduct* the entire content of *Argument*.
  - Otherwise, $\mathcal{R}$ copies to *Reduct* the representation of the variable occurrences, without altering it.

Lemma 10 *If $M \rightarrow^n N$, then $n \leq Time(M)$ and $|N| \leq Time(M)$.*

Proof. Clear from the definition of $Time(M)$. □

Theorem 2 *$\mathcal{R}$ computes the normal form of the term M in $O((Time(M))^4)$ steps.*

6 Closed Values as a Partial Combinatory Algebra

If U and V are closed values and UV has a normal form W (which must be a closed value), then we will denote W by $\{U\}(V)$. In this way, we can give Ξ the status of a partial applicative structure, which turns out to be a partial combinatory algebra. The abstract time measure induces a finer structure on Ξ, which we are going to illustrate in this section. In particular, we will be able to show the existence of certain elements of Ξ having both usual combinatorial properties as well as bounded behaviour. These properties are exploited in [4], where elements of Ξ serves as (bounded) realizers in a semantic framework.

In the following, $Time(\{U\}(V))$ is simply $Time(UV)$ (if it exists). Moreover, couples of terms can be encoded in the usual way: $\langle V, U\rangle$ will denote the term $\lambda x.xVU$.

First of all, we observe the identity and basic operations on couples take constant time. For example, there is a term M_{swap} such that $\{M_{swap}\}(\langle V, U\rangle) = \langle U, V\rangle$ and $Time(\{M_{swap}\}(\langle V, U\rangle)) = 5$. Formally:

Proposition 5 (Basic Operators) *There are terms $M_{id}, M_{swap}, M_{assl}, M_{tens} \in \Xi$ and constants c_{id}, c_{swap}, c_{assl}, c^1_{tens} and c^2_{tens} such that, for every $V, U, W \in \Xi$, there is $Y \in \Xi$ such that*

$$\begin{aligned}
\{M_{id}\}(V) &= V;\\
\{M_{swap}\}(\langle V, U\rangle) &= \langle U, V\rangle;\\
\{M_{assl}\}(\langle V, \langle U, W\rangle\rangle) &= \langle\langle V, U\rangle, W\rangle;\\
\{M_{tens}\}(V) &= Y;\\
\{Y\}(\langle U, W\rangle) &= \langle\{V\}(U), W\rangle;\\
Time(\{M_{id}\}(V)) &\leq c_{id};\\
Time(\{M_{swap}\}(\langle V, U\rangle)) &\leq c_{swap};\\
Time(\{M_{assl}\}(\langle V, \langle U, W\rangle\rangle)) &\leq c_{assl};\\
Time(\{M_{tens}\}(V)) &\leq c^1_{tens};\\
Time(\{Y\}(\langle U, W\rangle)) &\leq c^2_{tens} + Time(\{V\}(U)).
\end{aligned}$$

Proof. First of all, let us define terms:

$$\begin{aligned}
M_{id} &\equiv \lambda x.x;\\
M_{swap} &\equiv \lambda x.x(\lambda y.\lambda w.\lambda z.zwy);\\
M_{assl} &\equiv \lambda x.x(\lambda y.\lambda w.w(\lambda z.\lambda q.\lambda r.r(\lambda s.syz)q));\\
M_{tens} &\equiv \lambda s.\lambda x.x(\lambda y.\lambda w.(\lambda x.\lambda z.zxw)(sy)).
\end{aligned}$$

Now, let us observe that

$$
\begin{aligned}
M_{id}V &\overset{(1)}{\twoheadrightarrow} V;\\
M_{swap}\langle V,U\rangle &\overset{(1)}{\twoheadrightarrow} \langle V,U\rangle(\lambda y.\lambda w.\lambda z.zwy)\\
&\overset{(1)}{\twoheadrightarrow} (\lambda y.\lambda w.\lambda z.zwy)VU\\
&\overset{(1)}{\twoheadrightarrow} (\lambda w.\lambda z.zwV)U\\
&\overset{(1)}{\twoheadrightarrow} (\lambda w.\lambda z.zwV)U\\
&\overset{(1)}{\twoheadrightarrow} \langle U,V\rangle;\\
M_{assl}\langle V,\langle U,W\rangle\rangle &\overset{(1)}{\twoheadrightarrow} \langle V,\langle U,W\rangle\rangle(\lambda y.\lambda w.w(\lambda z.\lambda q.\lambda r.r(\lambda s.syz)q))\\
&\overset{(1)}{\twoheadrightarrow} (\lambda y.\lambda w.w(\lambda z.\lambda q.\lambda r.r(\lambda s.syz)q))V\langle U,W\rangle\\
&\overset{(1)}{\twoheadrightarrow} (\lambda w.w(\lambda z.\lambda q.\lambda r.r(\lambda s.sVz)q))\langle U,W\rangle\\
&\overset{(1)}{\twoheadrightarrow} \langle U,W\rangle(\lambda z.\lambda q.\lambda r.r(\lambda s.sVz)q)\\
&\overset{(1)}{\twoheadrightarrow} (\lambda z.\lambda q.\lambda r.r(\lambda s.sVz)q)UW\\
&\overset{(1)}{\twoheadrightarrow} \lambda r.r(\lambda s.sVU)W) \equiv \langle\langle V,U\rangle,W\rangle;\\
M_{tens}V &\overset{(1)}{\twoheadrightarrow} \lambda x.x(\lambda y.\lambda w.(\lambda x.\lambda z.zxw)(Vy)) \equiv Y,\\
Y\langle U,W\rangle &\overset{(1)}{\twoheadrightarrow} \langle U,W\rangle(\lambda y.\lambda w.(\lambda x.\lambda z.zxw)(Vy))\\
&\overset{(1)}{\twoheadrightarrow} (\lambda y.\lambda w.(\lambda x.\lambda z.zxw)(Vy))UW\\
&\overset{(1)}{\twoheadrightarrow} (\lambda w.(\lambda x.\lambda z.zxw)(VU))W\\
&\overset{(1)}{\twoheadrightarrow} (\lambda x.\lambda z.zxW)(VU).
\end{aligned}
$$

□

There is a term in Ξ which takes as input a pair of terms $\langle V,U\rangle$ and computes the composition of the functions computed by V and U. The overhead is constant, i.e. do not depend on the intermediate result.

Proposition 6 (Composition) *There are a term $M_{comp} \in \Xi$ and two constants c^1_{comp}, c^2_{comp} such that, for every $V,U,W,Z \in \Xi$, there is $X \in \Xi$ such that:*

$$
\begin{aligned}
\{M_{comp}\}(\langle V,U\rangle) &= X;\\
\{X\}(W) &= \{V\}(\{U\}(W));\\
Time(\{M_{comp}\}(\langle V,U\rangle)) &\leq c^1_{comp};\\
Time(\{X\}(W)) &\leq c^2_{comp} + Time(\{U\}(W)) + Time(\{V\}(\{U\}(W))).
\end{aligned}
$$

Proof. First of all, let us define the term:

$$M_{comp} \equiv \lambda x.x(\lambda x.\lambda y.\lambda z.x(yz)).$$

Now, let us observe that

$$\begin{aligned} M_{comp}\langle V, U\rangle &\overset{(1)}{\twoheadrightarrow} \langle V, U\rangle(\lambda x.\lambda y.\lambda z.x(yz)) \\ &\overset{(1)}{\twoheadrightarrow} (\lambda x.\lambda y.\lambda z.x(yz))VU \\ &\overset{(1)}{\twoheadrightarrow} (\lambda y.\lambda z.V(yz))U \overset{(1)}{\twoheadrightarrow} \lambda z.V(Uz) \equiv X, \\ XW &\overset{(1)}{\twoheadrightarrow} V(UW) \end{aligned}$$

This concludes the proof. □

We need to represent functions which go beyond the realm of linear logic. In particular, terms can be duplicated, but linear time is needed to do it.

Proposition 7 (Contraction) *There are a term $M_{cont} \in \Xi$ and a constant c_{cont} such that, for every $V \in \Xi$:*

$$\begin{aligned} \{M_{cont}\}(V) &= \langle V, V\rangle; \\ Time(\{M_{cont}\}(V)) &\leq c_{cont} + |V|. \end{aligned}$$

Proof. First of all, let us define the term:

$$M_{cont} \equiv \lambda x.\lambda y.yxx.$$

Now, let us observe that

$$M_{cont}V \overset{(n)}{\twoheadrightarrow} \langle V, V\rangle,$$

where $n \leq |V|$. □

From a complexity viewpoint, what is most interesting is the possibility to perform higher-order computation with constant overhead. In particular, the universal function is realized by a term M_{eval} such that $\{M_{eval}\}(\langle V, U\rangle) = \{V\}(U)$ and $Time(\{M_{eval}\}(\langle V, U\rangle)) = 4 + Time(\{U\}(V))$.

Proposition 8 (Higher-Order) *There are terms $M_{eval}, M_{curry} \in \Xi$ and constants c_{eval}, c^1_{curry}, c^2_{curry}, c^3_{curry} such that, for every $V, U \in \Xi$, there are $W, X, Y, Z \in \Xi$ such that:*

$$
\begin{aligned}
\{M_{eval}\}(\langle V, U\rangle) &= \{V\}(U);\\
\{M_{curry}\}(V) &= W;\\
\{W\}(X) &= Y;\\
\{Y\}(Z) &= \{V\}(\langle X, Z\rangle);\\
Time(\{M_{eval}\}(\langle V, U\rangle)) &\leq c_{eval} + Time(\{U\}(V));\\
Time(\{M_{curry}\}(V)) &\leq c^1_{curry};\\
Time(\{W\}(X)) &\leq c^2_{curry};\\
Time(\{Y\}(Z)) &\leq c^3_{curry} + Time(\{V\}(\langle X, Z\rangle)).
\end{aligned}
$$

Proof. Define:

$$
\begin{aligned}
M_{eval} &\equiv \lambda x.x(\lambda y.\lambda w.yw);\\
M_{curry} &\equiv \lambda x.\lambda y.\lambda w.x(\lambda z.zyw).
\end{aligned}
$$

Now, observe that

$$
\begin{aligned}
M_{eval}\langle V, U\rangle &\overset{(1)}{\twoheadrightarrow} \langle V, U\rangle(\lambda y.\lambda w.yw)\\
&\overset{(1)}{\twoheadrightarrow} (\lambda y.\lambda w.yw)VU\\
&\overset{(1)}{\twoheadrightarrow} (\lambda w.Vw)U \overset{(1)}{\twoheadrightarrow} VU;\\
M_{curry}V &\overset{(1)}{\twoheadrightarrow} \lambda y.\lambda w.V(\lambda z.zyw) \equiv W,\\
WX &\overset{(1)}{\twoheadrightarrow} \lambda w.V(\lambda z.zXw) \equiv W \equiv Y,\\
YZ &\overset{(1)}{\twoheadrightarrow} V(\lambda z.zXZ) \equiv V\langle X, Z\rangle.
\end{aligned}
$$

This concludes the proof. □

The fact that a "universal" combinator with a constant cost can be defined is quite remarkable. It is a consequence of the inherent higher-order of the lambda-calculus. Indeed, this property does not hold in the context of Turing machines.

7 Conclusions

We have introduced and studied the difference cost model for the pure, untyped, call-by-value lambda-calculus. The difference cost model satisfies the invariance thesis, at least in its weak version [15]. We have given sharp complexity bounds on the simulations establishing the invariance and giving evidence that the difference cost model is a parsimonious one. We do not claim

this model is the definite word on the subject. More work should be done, especially on lambda-calculi based on other evaluation models.

The availability of this cost model allows to reason on the complexity of call-by-value reduction by arguing on the structure of lambda-terms, instead of using complicated arguments on the details of some implementation mechanism. In this way, we could obtain results for eager functional programs without having to resort to, e.g., a SECD machine implementation.

We have not treated space. Indeed, the very definition of space complexity for lambda-calculus—at least in a less crude way than just "the maximum ink used" [11] —is an elusive subject which deserves better and deeper study.

References

[1] Andrea Asperti. On the complexity of beta-reduction. In *Proc. 23rd ACM SIGPLAN Symposium on Principles of Programming Languages*, pages 110–118, 1996.

[2] Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages*, volume 45 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.

[3] Tomasz Blanc, Jean-Jacques Lévy, and Luc Maranget. Sharing in the weak lambda-calculus. In *Processes, Terms and Cycles*, volume 3838 of *LNCS*, pages 70–87. Springer, 2005.

[4] Ugo Dal Lago and Martin Hofmann. Quantitative models and implicit complexity. In *Proc. Foundations of Software Technology and Theoretical Computer Science*, volume 3821 of *LNCS*, pages 189–200. Springer, 2005.

[5] Ugo Dal Lago and Simone Martini. An invariant cost model for the lambda calculus. In A. Beckmann *et al.*, editor, *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006*, volume 3988 of *Lecture Notes in Computer Science*, pages 105–114. Springer, 2006.

[6] Vincent Danos and Laurent Regnier. Head linear reduction. Manuscript, 2004.

[7] Nicolaas Govert de Bruijn. Lambda calculus with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae*, 34(5):381–392, 1972.

[8] Mariangiola Dezani-Ciancaglini, Simona Ronchi della Rocca, and Lorenza Saitta. Complexity of lambda-terms reductions. *RAIRO Informatique Theorique*, 13(3):257–287, 1979.

[9] Gudmund Skovbjerg Frandsen and Carl Sturtivant. What is an efficient implementation of the lambda-calculus? In *Proc. 5th ACM Conference on*

Functional Programming Languages and Computer Architecture, pages 289–312, 1991.

[10] John Lamping. An algorithm for optimal lambda calculus reduction. In *Proc. 17th ACM SIGPLAN Symposium on Principles of Programming Languages*, pages 16–30, 1990.

[11] Julia L. Lawall and Harry G. Mairson. Optimality and inefficiency: What isn't a cost model of the lambda calculus? In *Proc. 1996 ACM SIGPLAN International Conference on Functional Programming*, pages 92–101, 1996.

[12] Julia L. Lawall and Harry G. Mairson. On global dynamics of optimal graph reduction. In *Proc. 1997 ACM SIGPLAN International Conference on Functional Programming*, pages 188–195, 1997.

[13] Jean-Jacques Lévy. Réductions correctes et optimales dans le lambda-calcul. Université Paris 7, Thèses d'Etat, 1978.

[14] Simona Ronchi Della Rocca and Luca Paolini. *The parametric lambda-calculus.* Texts in Theoretical Computer Science: An EATCS Series. Springer-Verlag, 2004.

[15] Peter van Emde Boas. Machine models and simulation. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 1–66. MIT Press, 1990.

[16] Christopher Wadsworth. Some unusual λ-calculus numeral systems. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 215–230. Academic Press, 1980.