# Studying Mixed Normalization Strategies of Lambda Terms

Vladyslav Shramenko, Victoriya Kuznietcova, Grygoriy Zholtkevych

*V.N. Karazin Kharkiv National University, Svobody sq., 4, Kharkiv, 61022, Ukraine*

**Abstract**
In this work, we introduce a new approach how to normalize lambda terms. We have defined and researched two randomized reduction strategies and compared their efficiency with the normal and applicative strategies for uniformly randomly generated lambda terms. In addition, we have developed a framework that allows us to find a randomized reduction strategy that will almost surely be normalized and the efficiency will be no worse than the efficiency of the existing reduction strategies for normalizing the given set of lambda terms.

**Keywords 1**
lambda-calculus, reduction strategies, uniformly random reduction strategy, mixed reduction strategy, functional programming

## 1. Introduction

Functional programming has its roots in academia, evolving from the lambda calculus, a formal system of computation based only on functions [1]. Functional programming languages use lambda calculus as a low- level programming language.

Functional programming is widely used to solve applied problems of artificial intelligence, such as parallel processing of large data streams [2], modeling of large data arrays [3], big data analysis [4], working with machine learning models [5], decentralized systems and blockchain [6], advanced applications with interactive interfaces [7] and more. Therefore, with the development of the sphere of high-tech information technologies, functional programming becomes relevant not only for solving specialized problems but also for classical projects.

Compilers of functional programming languages perform a reduction process to obtain normal forms of lambda terms. Various reduction strategies differ in the efficiency of the normalizing process. For large programs, the compilation process can take a long time. Thus, there is a desire to optimize the process of normalizing lambda terms.

The **object** of the study is the process of reducing lambda terms to obtain the normal form of a lambda term.

The **subject** of the study is lambda term reduction strategies.

The **aims** of the study:

- to develop more sophisticated lambda term reduction strategies for normalizing lambda terms;
- to study the impact of randomness in the process of normalizing randomly generated lambda terms;
- to develop a framework that searches for a reduction strategy that will find the normal forms of a given class of lambda terms as quickly as possible.

To achieve the **aim** the following tasks were formulated:

- to analyze the existing reduction strategies for lambda terms, their advantages and disadvantages;
- to define a randomized reduction strategy;

2nd International Workshop of IT-professionals on Artificial Intelligence (ProfIT AI 2022), December 2-4, 2022, Łódź, Poland
EMAIL: shramenko.vlad@gmail.com (V. Shramenko); vkuznietcova@karazin.ua (V. Kuznietcova); g.zholtkevych@gmail.com (G. Zholkevych)
ORCID: 0000-0002-3551-6942 (V. Shramenko); 0000-0003-3882-1333 (V. Kuznietcova); 0000-0002-7515-2143 (G. Zholkevych)

- to define a uniformly random reduction strategy;
- to define a mixed reduction strategy;
- to implement reduction strategies as program code;
- to find the best reduction strategy for normalizing randomly generated lambda terms;
- to conduct simulation experiments to compare reduction strategies.

## 2. Background

## 2.1. Reduction Process in Lambda Calculus

Functional languages are closely related to the lambda calculus. Functional languages may be divided into two classes: strict and non-strict. Strict languages typically employ a call-by-value left-to-right evaluation strategy for arguments to functions. Non-strict languages employ a graphical form of call-by-name evaluation strategy, in which once the value of an argument is calculated this value replaces the expression representing the argument. This ensures that an argument is reduced at most once in the evaluation of function call. Each of these methods of evaluation has an associated cost [8].

Functional languages perform no reduction under abstractions and thus reduce terms to weak normal forms only. In particular, call-by-value reduces to weak normal form, and call-by-name reduces to weak head normal form. These normal forms are defined in [9]. This section contains basic definitions of some well-known concepts related to the lambda calculus [10].

*Definition 1.* Let $V = \{v_1, v_2, \ldots\}$ is set of variables. lambda calculus is the language of lambda terms defined by the following grammar:

$$M, N \to var | (MN) | (\lambda\, var\; M), \tag{1}$$

where $var$ is a metavariable used for referring to elements of $V$ and $M$, $N$ are lambda terms.

We denote the set of lambda terms as $\Lambda$.

*Definition 2.* The lambda term $K$ is called $\beta$-reducible term or $\beta$-redex if $K$ include the term

$$(\lambda\, var\; M)N, \tag{2}$$

where $M, N \in \Lambda$, $var \in V$.

We denote the set of $\beta$-reducible lambda terms as $\Lambda_\beta \subseteq \Lambda$.

*Definition 3.* We denote the $K\{M\}$ as lambda term $K \in \Lambda$ that include $M \in \Lambda$. The relation of $\beta$-reduction $\to_\beta \subseteq \Lambda \to \Lambda$ is defined as

$$\to_\beta = \{(K\{(\lambda x, M)N\}, K\{M(N \coloneqq x)\}) | K \in \Lambda_\beta, M, N \in \Lambda\}. \tag{3}$$

We denote by $\twoheadrightarrow_\beta$ the reflexive and transitive closure of $\to_\beta$.

*Definition 4.* The lambda term $K$ is in normal form if $K$ does not contain any $\beta$-redex, i.e. $K \notin \Lambda_\beta$. We denote by $\Lambda_N \subseteq \Lambda$ the set of lambda terms that have normal form.

*Definition 5.* Reduction strategy $R$ is mapping $\Lambda \to \Lambda$, that

$$\forall M \in \Lambda : M \twoheadrightarrow_\beta R(M). \tag{4}$$

We denote the mapping $R$ as $\to_R$ and by $\twoheadrightarrow_R$ the reflexive and transitive closure of $\to_R$.

## 2.2. Classification of Reduction Strategies

A given term can contain several redexes, hence several different β-reductions could be applied. The reduction strategy chooses between different methods of term reduction. Each strategy has its advantages and disadvantages and different computational efficiency. In addition, depending on the lambda term, the efficiency of reduction strategies is different. Also, for some terms, one reduction strategy turns out to be more efficient, and for others, another reduction strategy.

The normalizing process can consist of a different number of steps, depending on the chosen strategy. The main types of reduction strategies are defined below [11].

*Definition 6.* Reduction strategy $R$ is called one-step strategy, if

$$\forall M \in \Lambda: M \to_\beta R(M). \tag{5}$$

Otherwise, it is a many-step strategy.

*Definition 7.* Reduction strategy $R$ is called deterministic strategy, if

$$\forall M \in \Lambda \exists N \in \Lambda: M \twoheadrightarrow_\beta N. \tag{6}$$

Otherwise, it is a nondeterministic strategy.

*Definition 8.* Reduction strategy $R$ is called normalized strategy, if a lambda term has a normal form, then reduction strategy $R$ will eventually reach it.

$$\forall M \in \Lambda, M =_\beta N, N \notin \Lambda_\beta: M \twoheadrightarrow_R N. \tag{7}$$

*Definition 9.* Reduction strategy $R$ is called efficient if $\forall M \in \Lambda$: $M \to_R N$ and $N$ can be computed in polynomial time.

One-step strategies for term reducing include [12]:

- leftmost-innermost (LI): in each step the leftmost of the innermost redexes is reduced, where an innermost redex is a redex not containing any redexes;
- leftmost-outermost (LO): in each step, the leftmost of the outermost redexes is reduced, where an outermost redex is a redex not contained in any redex;
- rightmost-innermost (RI): in each step, the rightmost of the innermost redexes is reduced, where an innermost redex is a redex not contained in any redexes;
- rightmost-outermost (RO): in each step the rightmost of the outermost redexes is reduced, where an outermost redex is a redex not contained in any redex.

Many-step reduction strategies include [12]:

- parallel-innermost: reduces all innermost redexes simultaneously. This is well-defined because the redexes are pairwise disjoint;
- parallel-outermost: reduces all outermost redexes simultaneously. This is well-defined because the redexes are pairwise disjoint;
- Gross-Knuth reduction, also called full substitution or Kleene reduction: all redexes in the term are simultaneously reduced.

The most commonly used reduction strategies in compilers for functional programming languages are LO (Normal order) and RI (Applicative order).

The normal order reduction strategy is normalized which ensures that the normal form can be obtained in a finite number of reductions if it exists. In addition, the LO strategy avoids doing useless work and reduces needed redexes only. Although this strategy reduces the same redexes multiple times, doing duplicating work. For compilers of functional programming languages, the LO strategy turns out to be not optimal in terms of the number of reduction steps.

The applicative order reduction strategy is generally considered more efficient than the LO strategy for compiling functional programs because programs often need to copy their arguments.[13] So, the RI strategy avoids duplicating work and does not reduce the same redexes. However, the RI strategy reduces redexes that will not be included in the resulting reduced lambda term. Also, in contrast to normal order, applicative order reduction may not terminate, even when the term has a normal form.

## 2.3. The Problem of an Efficient Optimal Normalized Strategy

*Definition 10*. Let mapping $F$ is the reduction strategy $R$. Then we denote the function $L_F(M)$ as the number of reduction steps to obtain the normal form of the term $M$ using the reduction strategy $R$.

*Definition 11*. Let $F$, $G$ are reduction strategies. $F \leq_L G$ if

$$\forall M \in \Lambda : L_F(M) \leq_L L_G(M). \tag{8}$$

*Definition 12*. Reduction strategy $F$ is called $L$-optimal if $F$ is normalized strategy and $\nexists$ strategy $R$: $F \leq_L R$.

There are many various reduction strategies. The question arises as to which reduction strategy to choose in order to get the normal form of the lambda term as quickly as possible. The problem of choosing a redex leading to a minimal-length reduction sequence is undecidable for the lambda calculus. It has been proven that there is no efficient $L$-optimal strategy and there is an $L$-optimal strategy [11].

## 2.4. Efficiency Criterion of Reduction Strategies

We will measure the efficiency of reduction strategy $R$ as the expected value of a random variable whose values are the outcome of the $L_R$ function of uniformly random generated lambda terms of the same size. We will use the Boltzmann model to uniform generate random lambda terms.

A Boltzmann generator for a class $\Lambda$ is built according to a recursive specification of the class $\Lambda$. The recurrence relation for $S_{m,n}$ (the number of lambda terms of size $n$ with at most $m$ distinct free variables) allows us to define the function generating lambda terms. More precisely, we construct bijections, called unranking functions, between all non-negative integers not greater than $S_{m,n}$ and binary lambda terms of size n with at most m distinct free variables. This approach is known as the recursive method, originating with Nijenhuis and Wil [14].

Using the algorithm for the uniform generation of random lambda terms, described in [14], we generated lambda terms with the number of vertices in the tree representation of lambda terms in the range from 50 to 60. From the generated terms, we select 100 terms that have a normal form. We determine the existence of a normal form using the normal reduction strategy. To do this, we reduce the lambda terms and count the number of reduction steps. If the reduction process is not completed in a limited number of steps, then we will assume that this term does not have a normal form.

Let's denote the set of the generated lambda terms as S.

## 3. Studying Randomized Normalization Strategies

## 3.1. Concept of Random Reduction Strategy

It is known that there is no strategy that will be optimal for all lambda terms at the same time. However, there is a strategy that is optimal for a certain class of terms. Therefore, there is a desire to look for optimal strategies for a certain class of lambda terms. Thus, the found strategy should quickly normalize the group of terms to the normal form.

A similar situation arises in game theory. Optimal strategies do not always exist among pure strategies. This problem is solved by introducing mixed strategies, we use each pure strategy with some probability. So, there is an element of randomness in the mixed strategy. As a result, it is guaranteed that the player's average payoff will be maximized [15].

Randomized algorithms are algorithms that make random choices during their execution. In practice, a randomized program would use values generated by a random number generator to decide the next step at several branches. Randomized algorithms are typically used to reduce the running time, time complexity; or the memory used, or space complexity, in a standard deterministic algorithm. For many problems, a randomized algorithm is the simplest, the fastest, or both [16].

There are two main types of randomized algorithms: Las Vegas algorithms and Monte-Carlo algorithms. In Las Vegas algorithms, the algorithm may use randomness to speed up the computation, but the algorithm must always return the correct answer to the input. Monte-Carlo algorithms do not have the former restriction, that is, they are allowed to give wrong return values. However, returning a wrong return value must have a small probability, otherwise, the Monte-Carlo algorithm would not be of any use [17].

Adding randomness may improve the efficiency of the lambda-term reduction process. We can also use machine learning or genetic algorithms to match the probabilities of using different reduction strategies for a given class of lambda terms. This will combine the advantages of various reduction strategies in one strategy in the best way.

*Definition 13*. Random reduction strategy $R$ is mapping $(\Lambda, \mathbb{N}) \rightarrow \Lambda$, that

$$\forall M \in \Lambda, n \in \mathbb{N}: M \twoheadrightarrow_\beta R(M,n). \tag{9}$$

*Definition 14*. Let $R$ is the random reduction strategy. Then we denote the function $L_{R,n}(M)$ as the average of $n$ times calculated the number of reduction steps to obtain the normal form of the term $M$ using the reduction strategy $R$ $n$ times.

## 3.2. Uniformly Random Reduction Strategy

The basic QuickSort algorithm has $O(n^2)$ running time for an array of n elements when we always choose the leftmost element in the array as the pivot. In the randomized sorting algorithm QuickSort, when we choose a random element in an array as a pivot, we achieve that the expected time of this algorithm is $O(n \log n)$ [18]. So, we can use randomness in a reduction strategy by reducing an arbitrary redex. The easiest way is to use uniform distribution and select uniformly random redex for reduction from all redexes in the given term.

*Definition 15*. Uniformly random reduction strategy $UR$ is mapping $(\Lambda, \mathbb{N}) \rightarrow \Lambda$, that $R$ reduces a redex that is uniformly randomly selected from all redexes in the term.

$$\forall M \in \Lambda, n \in \mathbb{N}: M \rightarrow_\beta UR(M,n). \tag{10}$$

The implementation of the uniformly random reduction strategy consists in:

1. Count the number of $\beta$-redexes in given lambda term. Let's denote the number of redexes by $N$;
2. Generate a random natural number $R$ in the range $[1, N]$;
3. Reduce the $R$-th redex.

The asymptotic of this algorithm is $O(V)$, where $V$ is the number of vertices of the lambda term tree representation. Hence the uniformly random reduction strategy is efficient.

## 3.3. Mixed Reduction Strategy

Various reduction strategies have their advantages and disadvantages. For different lambda terms, different strategies turn out to be optimal. As in game theory, we can introduce mixed strategies, which means that we use each pure strategy with some probability. This can significantly increase the average rate of reduction of lambda terms from a given set.

*Definition 16*. Let $R = \{R_1, R_2, \dots, R_n\}$ is set of reduction strategies and $p = (p_1, p_2, \dots, p_n)$ and $\sum_{i=1}^{n} p_i = 1$. Mixed reduction strategy $MR_{R,p}$ is mapping $(\Lambda, \mathbb{N}) \rightarrow \Lambda$, that $MR_{R,p}$ is applying one of the reduction strategies from $R - R_i$ to reduce a redex where $p_i$ is the probability to choose $R_i$ from $R$.

*Definition 17*. For given reduction strategy $R$ define a function $V: \Lambda_N \rightarrow \mathbb{R}$ as Lyapunov if:

$$\exists n \in \mathbb{R}\ \forall \mathrm{M} \in \Lambda_N: V(M) \geq n. \tag{11}$$

$$\exists \epsilon > 0 \forall M \in \Lambda_N \cap \Lambda_\beta: V(M) > V\big(R(M)\big) + \epsilon. \tag{12}$$

where V is extended to partial distribution as follows

$$V\big(R(M)\big) = \sum\nolimits_{m \in \Lambda} V(m)\, \mathbb{P}(M \rightarrow m). \tag{13}$$

Foster's theorem [19]. If we can define for reduction strategy $R$ a Lyapunov function $V$, then $R$ is normalized and the average derivation length

$$\exists \epsilon > 0 \forall M \in \Lambda_N : L_R(M) \le V(M)/\epsilon. \tag{14}$$

*Theorem 1.* Let $R = \{R_1, R_2, \dots, R_n\}$ is set of reduction strategies and $p = (p_1, p_2, \dots, p_n)$ and $\sum_{i=1}^{n} p_i = 1$. Let $\exists\ i \in [1, n]$: $R_i = LO$ is the normal order reduction strategy and $p_i > 0$. Then, mixed reduction strategy $MR_{R,p}$ is almost-surely normalized.

*Proof.* To prove this theorem, we use Foster's Theorem. Consider function $V = L_{LO}$. Obviously, the first condition of the Lyapunov function definition is satisfied:

$$\forall M \in \Lambda_N : L_{LO}(M) \ge 0.$$

Consider $M \in \Lambda_N \cap \Lambda_\beta$.

Let $m = MR_{R,p}(M)$ and $M \to_{Rj} N_j$ , $j \in [1, n]$.

Obviously, $L_{LO}(M) = L_{LO}(N_i) + 1$.

The LO reduction strategy satisfies the inequality $L_{LO}(K) \le L_{LO}(M)$, where $M \to_\beta K$.

Therefore $L_{LO}(N_j) \le L_{LO}(M)$ , $j \in [1, n]$.

$$\begin{aligned}
L_{LO}(m) &= \sum_{j=1}^{n} p_j L_{LO}(N_j) = p_i(L_{LO}(M) - 1) + \sum_{j=1}^{i} p_j L_{LO}(N_j) + \sum_{j=i+1}^{n} p_j L_{LO}(N_j) = \\
&= p_i L_{LO}(M) - p_i + \sum_{j=1}^{i} p_j L_{LO}(M) + \sum_{j=i+1}^{n} p_j L_{LO}(M) = \\
&= p_i L_{LO}(M) - p_i + L_{LO}(M)\left(\sum_{j=1}^{i} p_j + \sum_{j=i+1}^{n} p_j\right) = \\
&= p_i L_{LO}(M) - p_i + L_{LO}(M)\left(\sum_{j=1}^{n} p_j - p_i\right) = \\
&= p_i L_{LO}(M) - p_i + L_{LO}(M)(1 - p_i) = L_{LO}(M) - p_i.
\end{aligned}$$

$$L_{LO}(m) \le L_{LO}(M) - p_i \Rightarrow \exists \epsilon = p_i > 0 : L_{LO}(M) \ge L_{LO}\left(MR_{R,p}(M)\right) + \epsilon.$$

So $V = L_{LO}$ is Lyapunov function for the given reduction strategy $MR_{R,p}$.

Hence, by the Foster's theorem the mixed reduction strategy $MR_{R,p}$ is normalized.

The implementation of the mixed reduction strategy $MR_{R,p}$ consists in:

1. Generate a random float number $r$ in the range [0, 1];
2. Find the smallest index $i$ for which $r \le \sum_{j=1}^{i} p_j$;
3. Apply $R_i$ reduction strategy.

If all reduction strategies from $R = \{R_1, R_2, \dots, R_n\}$ are efficient then the mixed reduction strategy $MR_{R,p}$ is also efficient.

The mixed reduction strategy $MR_{R,p}$ involves choosing a probability vector $p$, which is the hyperparameter of the $MR_{R,p}$ reduction strategy. In this regard, the question arises of how to choose a probability vector for the maximum efficiency of reducing lambda terms from $S \subseteq \Lambda_N$ to normal forms. In this case we need to solve the optimization problem: minimize the sum of the number of required reduction steps for the lambda terms from $S$ by finding the as best as possible probability vector $p$ for the reduction strategy $MR_{R,p}$:

$$e(p) = \sum_{M \in S} L_{MR \underset{R,p}{\rightarrow}}(M) \to min. \tag{15}$$

A common way to find good combinations of hyperparameters is a grid search. The main disadvantage of searching on a grid is the need for a complete enumeration of all combinations, which can take a very long time. Any search on a grid is limited by a subset of hyperparameters that we have defined. The genetic algorithm allows us to find the best combination in less time than in the case of an exhaustive search on the grid. Genetic algorithms can be used not only as an efficient way to search on a grid but also for direct search in the entire parameter space [20].

The basis of genetic algorithms is based on the hypothesis that the optimal solution to a problem can be assembled from small structural elements. Individuals that contain some of the desirable structural elements are assigned a higher score. Repeated operations of selection and crossing lead to the emergence of ever better individuals, passing these structural elements to the next generation, perhaps combined with other successful structural elements. This creates a genetic pressure that directs the population towards the emergence of an increasing number of individuals with structural elements that form the optimal solution. As a result, each generation is better than the previous one and contains more individuals close to the optimal solution.

Consider mixed reduction strategy $MR_{\{LO,\text{RI},\},p}$, $p = (p_{LO}, p_{RI})$ is probability vector, where $p_{RI} = 1 - p_{LO}$ . For various values $p_{LO} \in [0, 1]$, we calculate the efficiency of the obtained reduction strategies $MR_{\{LO,\text{RI},\},p}$. The results are shown in Fig. 1, which shows the dependence of the efficiency of the mixed reduction strategy $MR_{\{LO,\text{RI},\},p}$ on the probability of using the normal strategy $p_{LO}$.

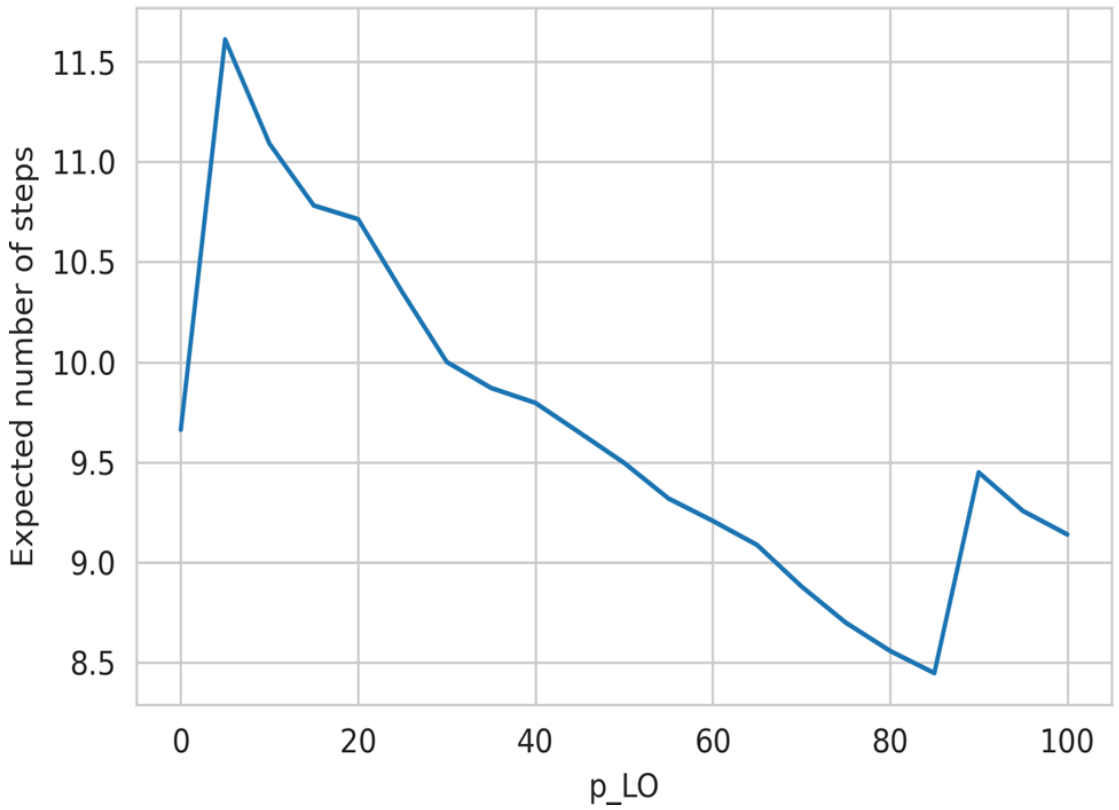

**Figure 1**: Dependence of the efficiency of the mixed reduction strategy $MR_{\{LO,\text{RI}\},p}$ on the probability of using the normal strategy $p_{LO}$

As seen from the figure 1, the most optimal values of the probabilities of using normal and applicative strategies of the mixed reduction strategy $MR_{\{LO,\text{RI}\},p}$ are

$$\begin{cases} p_{LO} = 0.85 \\ p_{RI} = 0.15 \end{cases}$$

for normalizing uniformly random generated lambda terms.

The expected value of reduction steps number to normalize the uniformly random generated lambda term:

$$M_{MR_{\{LO,RI\},(0.85,0.15)}} = 8.45. \tag{16}$$

In addition, the mixed reduction strategy $MR_{\{LO,\mathrm{RI}\},(0.85,0.15)}$ is almost surely normalized by Theorem 1. Thus, we have found a mixed reduction strategy that is more efficient than applicative and normal reduction strategies for normalizing uniformly random generated lambada terms, and it is almost surely normalized.

Consider mixed reduction strategy $MR_{\{LO,\mathrm{RI},\mathrm{LI},\mathrm{RO},UR\},p}$, where $p = (p_1, p_2, p_3, p_4, p_5)$ and $\sum_{i=1}^{5} p_i = 1$.

We used a genetic algorithm to find the best as possible probability vector $p$. We defined an individual as a list of decimal numbers that will be the desired probability vector. The fitness function for an individual p was the efficiency criterion of the mixed strategy which was described in section 2.4. As a genetic selection, crossover, and mutation operator, we used tournament selection of size 2 in combination with elitism [21], Simulated Binary Crossover [22], and Polynomial Bounded Mutation [23], respectively.

As the result, we found the probability vector

$$p = (0.72, 0.02, 0.02, 0.17, 0.7). \tag{17}$$

Let's denote the obtained mixed strategy $MR_{\{LO,\mathrm{RI},\mathrm{LI},\mathrm{RO},UR\},p}$ as $MR_{\mathrm{opt}}$.

The mixed reduction strategy $MR_{\mathrm{opt}}$ is almost surely normalized by Theorem 1.

We can generalize this algorithm to an arbitrary set $R$ of reduction strategies. Thus, we have developed a framework that allows us to find the mixed reduction strategy that will normalize a given class of lambda terms with almost the greatest efficiency. In addition, if our mixed strategy is using the normal strategy with non-zero probability then it is almost surely normalized for lambda terms having a normal form.

## 3.4. Comparing the Efficiency of the Uniformly Random, Mixed, Normal, and Applicative Reduction Strategies

We want to compare the efficiency of the most common reduction strategies LO (normal order) and RI (applicative order) with UR (uniformly random) and $MR_{\mathrm{opt}}$ (mixed strategy obtained in the previous section) reduction strategies for uniformly random generated lambda terms.

Based on the generated lambda terms S in section 1.4, we construct histograms of the distributions of the required number of reduction steps to obtain the normal form by using LO, RI, UR, and $MR_{\mathrm{opt}}$ reduction strategies respectively (Fig. 2-5).

Hypotheses about the laws of distribution of the values of the functions $L_{LO}$, $L_{RI}$, $L_{UR,100}$ and $L_{MR\{LO,\mathrm{RI},\mathrm{LI},\mathrm{RO},UR\},p,100}$ for uniformly random generated lambda terms from $S$ are checked. The following distribution laws were taken for comparison: normal, Cauchy, $\chi^2$, exponential, generalized normal (exponential power), gamma, log-normal and Rayleigh. Graphs of distribution laws that best describe the values of the functions $L_{LO}$, $L_{RI}$, $L_{UR,100}$ and $L_{MR\mathrm{opt},100}$ for lambda terms from $S$ are shown on Fig. 2-5.

It is determined that the values of the functions $L_{LO}$ of the uniform generated lambda terms is distributed according to the log-normal distribution law with the $\mu = 1.65$, $\sigma = 1.06$ parameters:

$$f(x,\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\ln(x)-\mu}{\sigma}\right)^2}. \tag{18}$$

The expected value of this distribution is

$$M_{LO} = e^{\mu+\frac{\sigma^2}{2}} = 9.14. \tag{19}$$

It is determined that the values of the functions $L_{RI}$ of the uniformly random generated lambda terms is distributed according to the log-normal distribution law with the $\mu$ = 2.04, $\sigma$ = 0.68 parameters of (18). The expected value of this distribution is

$$M_{RI} = e^{\mu+\frac{\sigma^2}{2}} = 9.66. \tag{20}$$

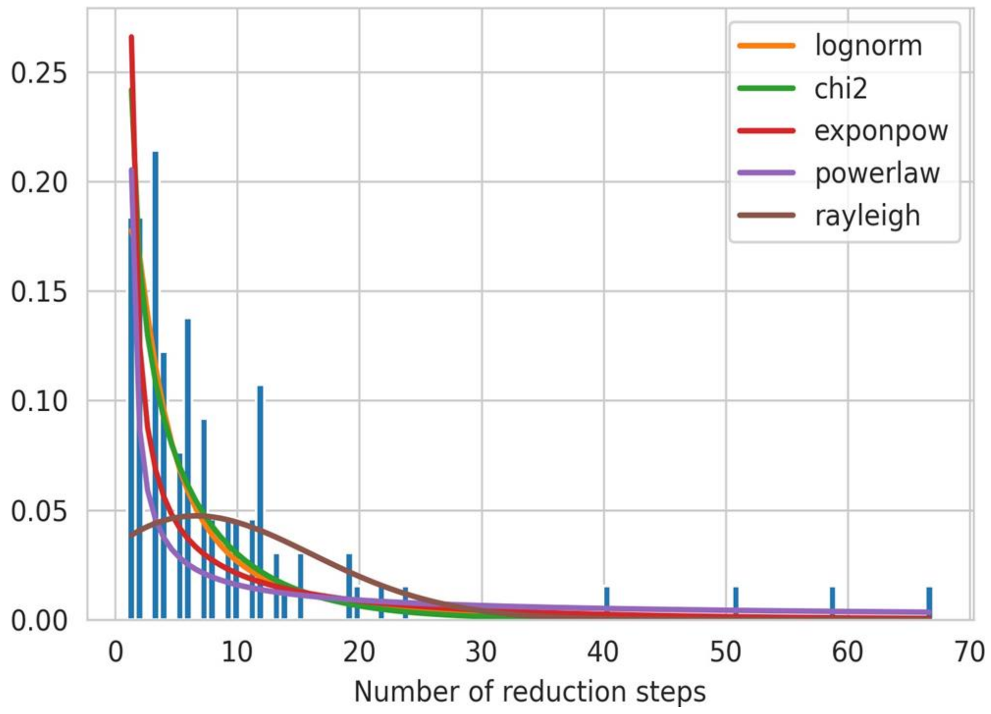

**Figure 2**: Histogram of the distributions of the reduction steps numbers $L_{LO}(M)$ of the uniformly random generated lambda terms $M \in$ S

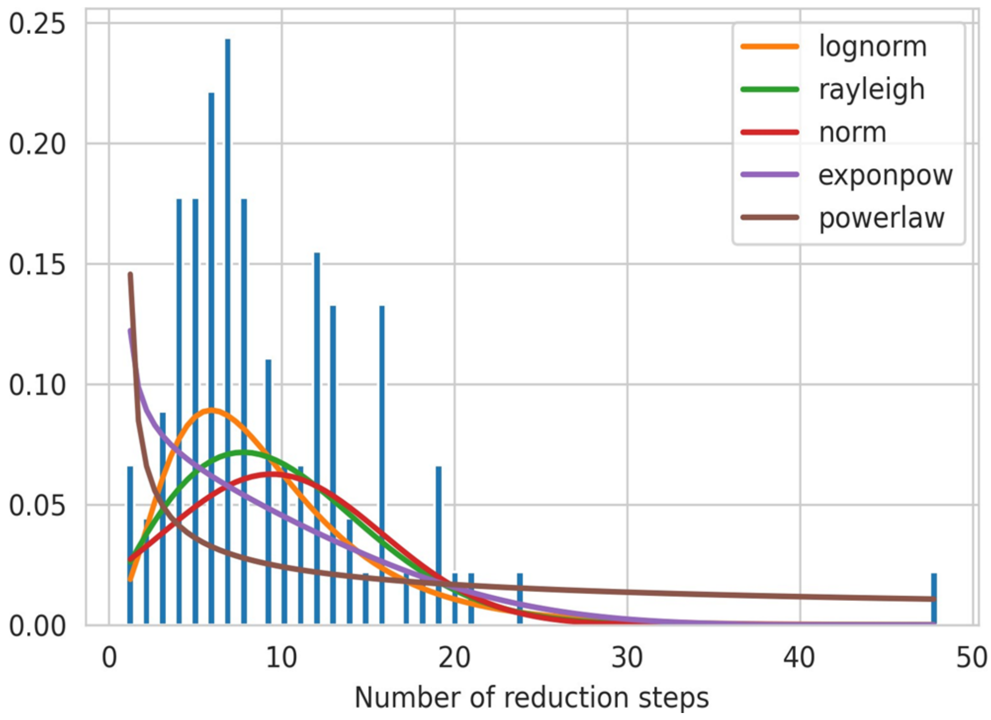

**Figure 3**: Histogram of the distributions of the reduction steps numbers $L_{\mathrm{RI}}(M)$ of the uniformly random generated lambda terms $M \in$ S

In addition, the 4 lambda terms were not normalized, although they have a normal form. The use of the applicative strategy for these terms turned out to be incessant. It is determined that the values of the functions $L_{UR,100}$ of the uniformly random generated lambda terms is distributed according to the log-normal distribution law with the $\mu$ = 1.92, $\sigma$ = 0.8 parameters of (18). The expected value of this distribution is

$$M_{RI} = 9.42. \tag{21}$$

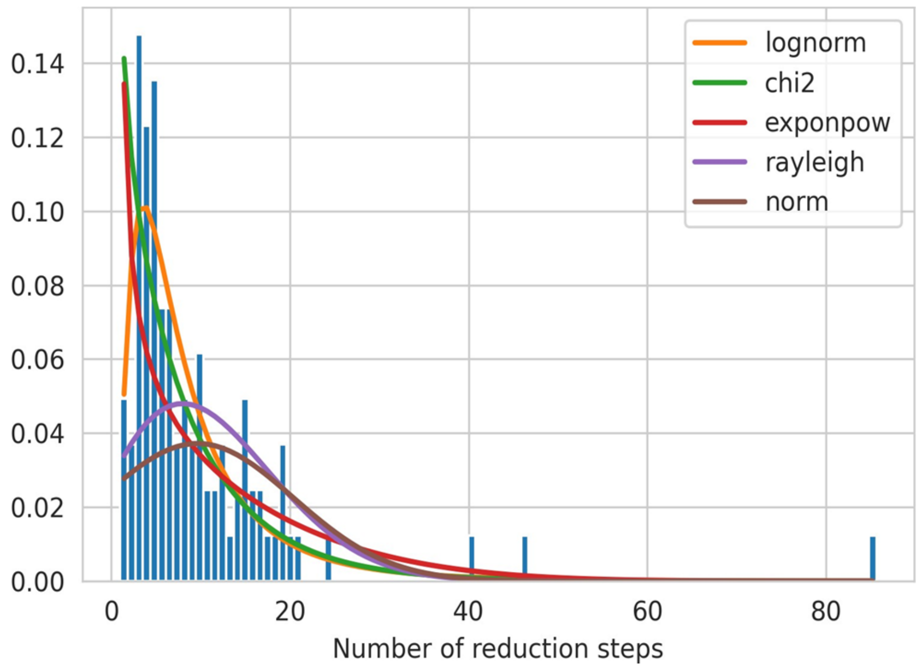

**Figure 4**: Histogram of the distributions of the reduction steps numbers $L_{\mathrm{UR},100}(M)$ of the uniformly random generated lambda terms $M \in \mathrm{S}$

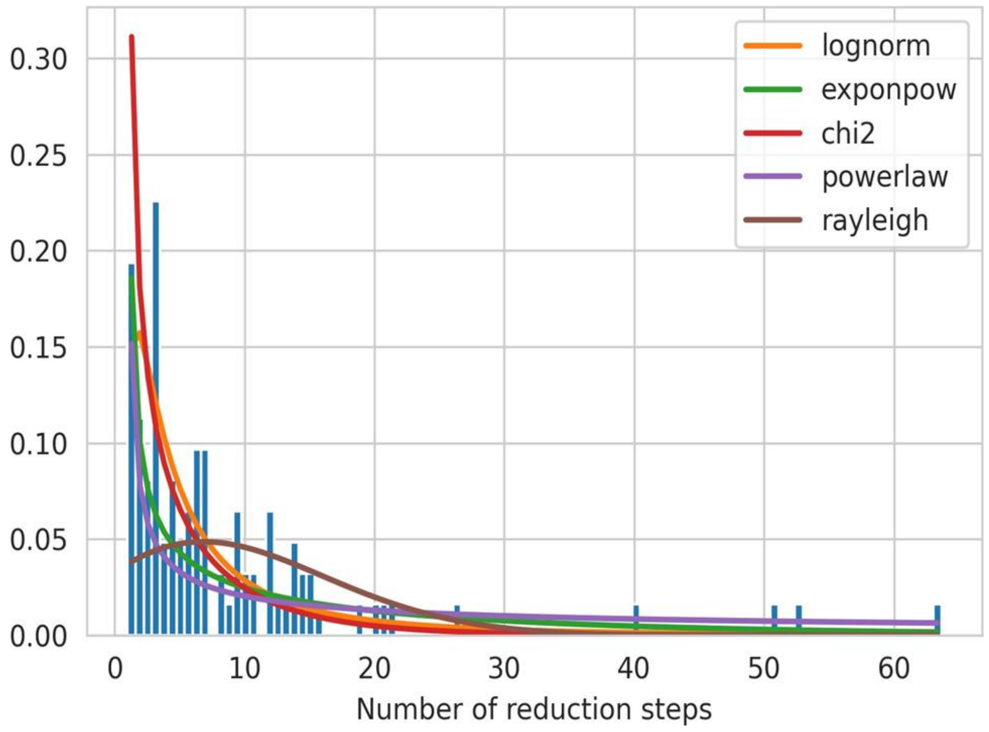

**Figure 5**: Histogram of the distributions of the reduction steps numbers $L_{\mathrm{MRopt},100}(M)$ of the uniformly random generated lambda terms $M \in \mathrm{S}$

Also note that all terms have been normalized, as opposed to using the applicative strategy.

It is determined that the values of the functions $L_{MR\text{opt},100}$ of the uniformly random generated lambda terms is distributed according to the log-normal distribution law with the $\mu = 1.67$, $\sigma = 0.96$ parameters of (18). The expected value of this distribution is

$$M_{MR_{opt}} = 8.39. \tag{22}$$

Thus, from the formulas 19 – 22, we got that

$$M_{RI} > M_{UR} > M_{LO} > M_{MR_{opt}}.$$

for uniform generated lambda terms of the same size.

We can see that the efficiency of the uniformly random reduction strategy is commensurate with the efficiency of normal and applicative strategies. Thus, the uniformly random strategy can be used for the lambda term reduction process the same as the normal or applicative strategies.

The mixed strategy turned out to be the worst among the considered strategies. In addition, the $MR_{\text{opt}}$ strategy is almost surely normalized and all lambda terms were normalized by using this strategy.

## 4. Conclusions

There is no universal strategy that has been the most efficient for reducing any lambda term. All existing reduction strategies have various advantages and disadvantages. The efficiency of reduction strategies essentially depends on the lambda terms for which it is used.

We defined a randomized reduction strategy for lambda terms. Two randomized reduction strategies have been defined and implemented as a program code: a uniformly random and a mixed reduction strategy.

A mixed reduction strategy is investigated. It is proved that a mixed reduction strategy for which the normal strategy is used with a probability greater than 0 is almost surely normalized.

A framework has been developed that allows us to find a mixed reduction strategy that will almost surely be normalized and the efficiency will be no worse than the efficiency of the existing reduction strategies for normalizing the existing class of lambda terms.

Simulation experiments to normalize uniformly random generated lambda terms using the mixed, uniformly random, normal, and applicative reduction strategies have been conducted.

The uniformly random reduction strategy is proposed for use since it is comparable in efficiency with the normal and applicative strategies. Also, the uniformly random reduction strategy turned out to be better than the applicative one in terms of the number of normalized lambda terms.

The efficiency of the mixed reduction strategy is significantly better than the efficiency of the other considered reduction strategies for normalizing randomly generated lambda terms. The mixed reduction strategy combines the advantages of several reduction strategies in the best way.

To sum up, randomized reduction strategies may be more efficient than standard non-randomized reduction strategies.

## 5. References

[1] Z. Hu, J. Hughes, and M. Wang, "How functional programming mattered," *National Science Review*, vol. 2, no. 3, pp. 349–370, Sep. 2015, doi: 10.1093/nsr/nwv042.

[2] J. Arora, S. Westrick, and U. A. Acar, "Provably space-efficient parallel functional programming," *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–33, Jan. 2021, doi: 10.1145/3434299.

[3] K. Bazilevych, et. al., "stochastic modelling of cash flow for personal insurance fund using the cloud data storage," *International Journal of Computing*, vol. 17, no. 3, pp. 153–162, Sep. 2018, doi: 10.47839/ijc.17.3.1035.

[4] L. Belcastro, R. Cantini, F. Marozzo, A. Orsino, D. Talia, and P. Trunfio, “Programming big data analysis: principles and solutions,” *Journal of Big Data*, vol. 9, no. 1, Jan. 2022, doi: 10.1186/s40537-021-00555-2.

[5] D. Chumachenko, et al., “Investigation of Statistical Machine Learning Models for COVID-19 Epidemic Process Simulation: Random Forest, K-Nearest Neighbors, Gradient Boosting,” *Computation*, vol. 10, no. 6, 86, 2022, doi: 10.3390/computation10060086.

[6] Q. Pan and X. Koutsoukos, “Building a Blockchain Simulation using the Idris Programming Language,” *Proceedings of the 2019 ACM Southeast Conference*, Apr. 2019, doi: 10.1145/3299815.3314456.

[7] S. Yakovlev *et al.*, “The Concept of Developing a Decision Support System for the Epidemic Morbidity Control.” *CEUR Workshop Proceedings*, vol. 2753, pp. 265-274, 2022.

[8] S. L. Peyton Jones, “Parallel Implementations of Functional Programming Languages,” *The Computer Journal*, vol. 32, no. 2, pp. 175–186, 1989, doi: 10.1093/comjnl/32.2.175.

[9] P. Sestoft, “Demonstrating Lambda Calculus Reduction,” *Electronic Notes in Theoretical Computer Science*, vol. 45, pp. 424–432, 2001, doi: 10.1016/s1571-0661(04)80973-3.

[10] D. Wright, “Reduction Types and Intensionality in the Lambda-Calculus,” 1992.

[11] E. Engeler, “H. P. Barendregt. The lambda calculus. Its syntax and semantics. Studies in logic and foundations of mathematics” *Journal of Symbolic Logic*, vol. 49, no. 1, pp. 301–303, 1984, doi: 10.2307/2274112.

[12] J. Klop, “Term Rewriting Systems.” 2021, 77 p.

[13] G. Mazzola, G. Milmeister, and J. Weissmann, Comprehensive Mathematics for Computer Scientists 2: Calculus and ODEs, Splines, Probability, Fourier and Wavelet Theory, Fractals and Neural Networks, Categories and Lambda Calculus. Springer Science & Business Media, 2004.

[14] K. Grygiel, P. Lescanne, “Counting and Generating Terms in the Binary Lambda Calculus (Extended version) Counting and Generating Terms in the Binary Lambda Calculus (Extended version),” 2015.

[15] S. Tadelis, *Game Theory*. 2013.

[16] M. Mitzenmacher and E. Upfal, “Probability and Computing: Randomized Algorithms and Probabilistic Analysis,” *undefined*, 2005.

[17] D. Fucci *et al.*, “A Longitudinal Cohort Study on the Retainment of Test-Driven Development”

[18] H. J. Karloff and P. Raghavan, “Randomized algorithms and pseudorandom numbers,” *Journal of the ACM*, vol. 40, no. 3, pp. 454–476, 1993, doi: 10.1145/174130.174132.

[19] P. Bremaud, “Markov chains”, 1999.

[20] Eyal Wirsansky, Hands-on genetic algorithms with Python : applying genetic algorithms to solve real-world deep learning and artificial intelligence problems / Eyal Wirsansky. Birmingham, Uk: Packt Publishing Ltd, 2020.

[21] M. Yusoff and A. A. Othman, “Genetic Algorithm with Elitist-Tournament for Clashes-Free Slots of Lecturer Timetabling Problem,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 12, no. 1, p. 303, 2018, doi: 10.11591/ijeecs.v12.i1.pp303-309.

[22] K. Deb and R. B. Agrawal, “Simulated Binary Crossover for Continuous Search Space Kalya nmoy D eb’ Ram B hushan A gr awal,” 2014.

[23] K. Deb and ayan Deb, “Analysing mutation schemes for real-parameter genetic algorithms,” *International Journal of Artificial Intelligence and Soft Computing*, vol. 4, no. 1, p. 1, 2014, doi: 10.1504/ijaisc.2014.059280.