mathematics

Article

Solution of the Problem $P = L$

Sergey Goncharov *,† and Andrey Nechesov *,†

Sobolev Institute of Mathematics, Academician Koptyug Ave., 4, 630090 Novosibirsk, Russia
* Correspondence: s.s.goncharov@math.nsc.ru (S.G.); nechesov@math.nsc.ru (A.N.)
† These authors contributed equally to this work.

Abstract: The problems associated with the construction of polynomial complexity computer programs require new techniques and approaches from mathematicians. One of such approaches is representing some class of polynomial algorithms as a certain class of special logical programs. Goncharov and Sviridenko described a logical programming language L_0, where programs inductively are obtained from the set of Δ_0-formulas using special terms. In their work, a new idea has been proposed to look at the term as a program. The computational complexity of such programs is polynomial. In the same years, a number of other logical languages with similar properties were created. However, the following question remained: can all polynomial algorithms be described in these languages? It is a long-standing problem, and the method of describing some polynomial algorithm in a not Turing complete logical programming language was not previously clear. In this paper, special types of terms and formulas have been found and added to solve this problem. One of the main contributions is the construction of p-iterative terms that simulate the work of the Turing machine. Using p-iterative terms, the work showed that class P is equal to class L, which extends the programming language L_0 with p-iterative terms. Thus, it is shown that L is quite expressive and has no halting problem, which occurs in high-level programming languages. For these reasons, the logical language L can be used to create fast and reliable programs. The main limitation of the language L is that the implementation of algorithms of complexity is not higher than polynomial.

Keywords: polynomiality; polynomial function; polynomial algorithm; Turing machine; logical programming language; semantic programming; smart contract; blockchain; AI

Citation: Goncharov, S.; Nechesov, A. Solution of the Problem $P = L$. *Mathematics* **2022**, *10*, 113. https://doi.org/10.3390/math10010113

Academic Editors: Francesco Aldo Costabile, Maria I. Gualtieri, Anna Napoli and Radi Romansky

Received: 29 November 2021
Accepted: 28 December 2021
Published: 31 December 2021

1. Introduction

In the 1980s–1990s, Ershov, Goncharov, and Sviridenko presented the theory of semantic programming [1]. The concepts of Σ-programs and Σ-specifications were introduced in this work. The hereditary finite list superstructure was chosen as a base mathematical model. The universe of this model is the hereditary finite lists generated by elements of the universe of the base model [2], and some LISP-like functions were added. Special logical Σ-formulas with input and output variables were used as Σ-programs [3]. This gave rise to the study of programming language semantics from a mathematical point of view.

Cenzer and Remmel were among the first to study polynomial structures [4]. They investigated the existence of computable isomorphisms between computable and polynomial structures. Then, Lewis and Papadimitriou explored polynomial-time reductions [5]. Then, for a long time, the open problem was to create a logical programming language, which would have polynomial complexity. Mantsivoda developed a logical programming language based on document models. Documents are the main elements of the model universe. Special functions are defined for working with them. Mantsivoda and Ponomaryov formalized this approach in their work [6]. This language is simple and efficient. All operations and relations are polynomial. After that, another type of logical programming language, semantic domain-specific languages, was developed [7]. This language is based on the ideas of semantic programming where a truth-checking formula on the model replaced computability [1]. All programs also have polynomial complexity, but the question of how well it described the class of polynomial algorithms remained. In the work [8], some interesting results were presented. The authors found that the primitive recursive representation of the algorithm with boundaries of the variables often had a polynomial complexity. Then, Alaev investigated the questions of polynomial representability of various structures in his work [9]. This work gave the key to understanding what polynomiality is. In parallel, Goncharov and Sviridenko developed and presented a new logical programming language in which special terms were used as logical programs [10]. All programs in this language have polynomial computational complexity. However, the main question remains: can all polynomial algorithms be described by this language? The result that will be proved in this work answers this question.

After proving the polynomial analogue of Gandy's fixed point theorem [11], it became clear that the language L_0 is wide enough. This language is used to construct quite complex constructions. In particular, this concerned the inductive construction of new types of data and computer programs in programming.

2. Preliminaries

Let $\mathfrak{B}$ be a p-computable model of the signature σ_0. A p-computable hereditary finite list superstructure $HW(\mathfrak{B})$ [11] was chosen as a mathematical model of the signature σ. The universe of the model $HW(\mathfrak{B})$ consists of elements of the model universe B and hereditarily finite lists $HW(B)$. Signature σ extends σ_0 with the next LISP-like list relations $\in$ (to be an element of a list), $\subseteq$ (to be an initial segment of a list) and the list operations $head$, $tail$, $cons$, $conc$, and constant nil [12]. Define new unary list operations $first$, $cons\_l$, $tail\_l$. The first operation $first$ gets the first list element, the second $cons\_l$ adds the element into the beginning of the list and the last operation $tail\_l$ removes the first element from the list correspondingly. Define new unary operations $strList$, $listStr$. The first operation $strList$ based on the input string of the form $l_1 \dots l_k$ builds a list of the form $< l_1, \dots, l_k >$, where $l_i \in \Sigma$, $i \in [1, \dots, k]$, the second operation $listStr$ based on the input list of the form $< l_1, \dots, l_k >$ builds a string $l_1 \dots l_k$, where $l_i \in \Sigma$, $i \in [1, \dots,]$. The signature σ is an extension of σ_0 with these new operation symbols. The main operations $head$, $tail$, $cons$, $conc$, and relations $\in$, $\subseteq$ have polynomial complexity [12]. It is easy to see that other operations $first$, $cons\_l$, $tail\_l$, $strList$, and $listStr$ have polynomial complexity.

Define Δ_0-formulas as first order formulas of the signature σ in which quantification is of the following two types:

- a restriction onto the list elements $\forall x \in t$ and $\exists x \in t$.
- a restriction onto the initial segments of list $\forall x \subseteq t$ and $\exists x \subseteq t$.

The set of Δ_0-formulas of the signature σ has been extended by induction with several types of terms: conditional terms, b-while terms, bounded recursive terms, and etc. [10] These formulas are denoted as $\Delta_0(\mathfrak{I})$-formulas and new terms are denoted as $\Delta_0(\mathfrak{I})$-terms. Denote the resulting set of $\Delta_0(\mathfrak{I})$-terms as a language L_0.

Definition 1. *Any $\Delta_0(\mathfrak{I})$-term from the language L_0 will be referred to as L_0-program.*

Definition 2. *Any $\Delta_0(\mathfrak{I})$-formula will be referred to as L_0-formula.*

L_0-program property:

- any L_0-program has a polynomial computation complexity on any p-computable enrichment $HW(\mathfrak{B})^*$ of the model $HW(\mathfrak{B})$.

L_0-formula property:

- any L_0-formula has a polynomial truth-checking algorithm on any p-computable enrichment $HW(\mathfrak{B})^*$ of the model $HW(\mathfrak{B})$.

In this work, to construct a suitable logical program for a polynomial function, the concept of a conditional term from the work of Goncharov [13] is used. Let $t_0, t_1, \dots, t_{n+1}$

is L_0-programs and $\theta_0, \dots, \theta_n$ is L_0-formulae [10]. Define the concept of a conditional term $t(\overline{v})$ in the next interpretation:

$$t(\overline{v}) = \begin{cases} t_0(\overline{v}), \text{ if } HW(\mathfrak{B}) \models \theta_0(\overline{v}) \\ t_1(\overline{v}), \text{ if } HW(\mathfrak{B}) \models \theta_1(\overline{v}) \& \neg\theta_0(\overline{v}) \\ \dots \\ t_n(\overline{v}), \text{ if } HW(\mathfrak{B}) \models \theta_n(\overline{v}) \& \neg\theta_0(\overline{v}) \& \dots \& \neg\theta_{n-1}(\overline{v}) \\ t_{n+1}(\overline{v}), \text{ otherwise} \end{cases} \tag{1}$$

where $\overline{v}$ have a form $(v_1, \dots, v_k)$ for some $k \in N$.

Conditional terms use a construction similar to the operator "if then else" in high-level programming languages. Leave also the other types of terms from [10] for the expressiveness of a new language. However, in the future, to construct a term describing a polynomial function, only conditional terms from work [10] will be used.

3. Polynomial Functions and Turing Machines

Let T_f be a deterministic Turing machine over the alphabet Σ representing polynomial function f. Let S be the set of symbols $\{1, B\}$ and Q be the set of states of the Turing machine, where q_1 is the initial state and q_0 is the final state. Let P_{T_f} be a Turing machine program that implements the function f. Since any program of the Turing machine is implemented through $\sigma : Q \times S \to Q \times S \times \{R, L\}$, then all elements of the program P_{T_f} will be presented in the form of a list $< q_{i_1}, s_{j_1}, q_{i_2}, s_{j_2}, \beta >$. Sequence of the symbols

$$c_i : \; s_{-m_i} \dots s_{-2}\, s_{-1}\, q_{k_i}\, s_0\, s_1 \dots s_{n_i} \tag{2}$$

is called the configuration of the Turing machine at the ith step.

Let β be the symbol from the set $\{R, L\}$. Let c_i be some configuration of T_f and there is a element $< q_{i_1}, s_{j_1}, q_{i_2}, s_{j_2}, \beta >$ from the program P_{T_f}. Then, with the help of this element, the configuration c_i will switch to another configuration c_j.

Since the Turing machine T_f represents a polynomial function f over the alphabet Σ, the machine will work on any input $x \in \Sigma^*$ no more than

$$p(|x|) = C_p \cdot |x|^{n_p} \tag{3}$$

steps for some fixed $C_p, n_p \in N$. Let $r(f(x))$ be the computational complexity of the function $f(x)$. From (3), it follows:

$$r(f(x)) \le p(|x|) \tag{4}$$

If the Turing machine T_f changes configuration c_i on c_{i+1}, then:

$$|c_i| - 1 \le |c_{i+1}| \le |c_i| + 1 \tag{5}$$

and from (5) the inequality follows for the final configuration c_{final}:

$$|c_{final}| \le |c_0| + p(|x|) \le d(|x|), \text{ for some polynomial } d(|x|) \tag{6}$$

It should be noted that if T_f reached the final configuration $c_{final} = c_j$ for some jth step and $j \le p(|x|)$; then, all the remaining configurations $c_{j+1}, \dots, c_{p(|x|)}$ would be equal to c_{final}.

Using the configuration c_i of the form (2) define a machine word w_{c_i} in the next form:

$$w_{c_i} : << s_{-m_i}, \dots, s_{-1} >, \underline{q_{k_i}}, < s_0, \dots, s_{n_i} >> \tag{7}$$

where $\underline{q_{k_i}}$ is equal to a string $q \dots q$ of the length $k_i + 1$ and $s_k \in \{B, 1\}, k \in [-m_i, n_i]$.

Define $w_{c_i,k}$ as the k-th element of the machine word w_{c_i}, where $k \in \{1, 2, 3\}$.

If configuration c_i has a form $q_{k_i}s_0 \dots s_{n_i}$, then the machine word w_{c_i} has a form:

$$w_{c_i} : < nil, \underline{q_{k_i}}, < s_0, \dots, s_{n_i} >> \tag{8}$$

Remark 1.

(1) *the state $\underline{q_{k_i}}$ is obtained from w_{c_i} and equal $head(tail(w_{c_i}))$*

(2) *monitored symbol s_0 is obtained from w_{c_i} and equal $first(head(w_{c_i}))$.*

Remark 2.

(1) *Equality (5) implies that*

$$|w_{c_i}| - C \leq |w_{c_{i+1}}| \leq |w_{c_i}| + C, \textit{for some fixed } C \in N \tag{9}$$

(2) *Equalities (6) and (9) imply that*

$$|w_{c_{final}}| \leq |w_{c_0}| + p(|x|) \cdot C \leq r'(|x|) \textit{ for some polynomial } r'(|x|).$$

Define a new binary operation $\otimes$ using the machine word w_{c_i} and the element $< \underline{q_1}, s_1, \underline{q_2}, s_2, \beta >$ from the program P_{T_f}:

Case 1: element equal $< \underline{q_1}, s_1, \underline{q_2}, s_2, R >$

$$w_{c_i} \otimes < \underline{q_1}, s_1, \underline{q_2}, s_2, R >= \begin{cases} w_{c_j}, & \text{if } head(tail(w_{c_i})) = \underline{q_1} \text{ and } first(head(w_{c_i})) = s_1 \\ nil, & \text{otherwise} \end{cases} \tag{10}$$

and $w_{c_j} =< w_{c_j,1}, w_{c_j,2}, w_{c_j,3} >$, where

$$w_{c_j,1} = cons(first(w_{c_i}), s_2); w_{c_j,2} = \underline{q_2};$$
$$w_{c_j,3} = tail\_l(head(w_{c_i}));$$

Case 2: element equal $< \underline{q_1}, s_1, \underline{q_2}, s_2, L >$

$$w_{c_i} \otimes < \underline{q_1}, s_1, \underline{q_2}, s_2, L >= \begin{cases} w_{c_j}, & \text{if } head(tail(w_{c_i})) = \underline{q_1} \text{ and } first(head(w_{c_i})) = s_1 \\ nil, & \text{otherwise} \end{cases} \tag{11}$$

and $w_{c_j} =< w_{c_j,1}, w_{c_j,2}, w_{c_j,3} >$, where

$$w_{c_j,1} = tail(first(w_{c_i})); w_{c_j,2} = \underline{q_2};$$
$$w_{c_j,3} = cons\_l(cons\_l(tail\_l(head(w_{c_i})), s_2), head(first(w_{c_i})));$$

Remark 3. *Operation $\otimes$ is polynomial.*

4. *p*-Iterative Terms

From the previous section, the length of the final machine word $w_{c_{final}}$ does not exceed the length of the initial machine word w_{c_0} plus the length of some polynomial $r'(|x|)$. The main goal of this section is to construct a p-iterative term so that the length of the final value should not exceed the length of the input value plus the value of some polynomial $v(|x|)$. Furthermore, it will be shown that such extension using p-iterative terms of the language L_0 does not take us beyond the polynomiality.

Let $HW(\mathfrak{B})$ be a p-computable model of the signature σ, $g(\mathbf{x})$ be a L_0-program, $\varphi(\mathbf{x})$ be a L_0-formula. Require $|g(\mathbf{x})| \leq |\mathbf{x}| + C_g$ for some $C_g \in N$. Let $u(|\mathbf{x}|)$ be a polynomial such that the complexity of checking the truth of the L_0-formula $\varphi(\mathbf{x})$ on model $HW(\mathfrak{B})$ should not exceed $u(|\mathbf{x}|)$. Let computation complexity $r(g(\mathbf{x}))$ of L_0-program $g(\mathbf{x})$ be bounded by some polynomial $s(|\mathbf{x}|) = C_s \cdot |\mathbf{x}|^{n_s}$. Define a p-iterative term $t(\mathbf{x}, n)$ using the following iterative construction:

$$\begin{aligned} g^0(\mathbf{x}) &= g(\mathbf{x}) \\ &\dots \\ g^{i+1}(\mathbf{x}) &= g(g^i(\mathbf{x})) \end{aligned} \tag{12}$$

The p-iterative term has the form:

$$t(\mathbf{x}, n) = \begin{cases} g^i(\mathbf{x}), & \text{if } i \leq n \; HW(\mathfrak{B}) \models \varphi(g^i(\mathbf{x})) \text{ and } \forall j < i \; HW(\mathfrak{B}) \not\models \varphi(g^j(\mathbf{x}))) \\ nil, \text{ otherwise} \end{cases} \tag{13}$$

Remark 4. $|g^{i+1}(x)| \leq |g^i(x)| + C_g$.

Theorem 1. *Let $HW(\mathfrak{B})$ be a p-computable model, $\varphi(x)$ be a L_0-formula, and $g(x)$ be a L_0-program with the condition $|g(\mathbf{x})| \leq |\mathbf{x}| + C_g$, $C_g \in N$. Then, p-iterative term from (13) is a p-computable function.*

Proof. Let $t(\mathbf{x}, n)$ be a p-iterative term. If the value of the term $t(\mathbf{a}, n_0)$ equal $g^{i+1}(\mathbf{a})$; then, $HW(\mathfrak{B}) \models \varphi(g^{i+1}(\mathbf{a}))$ for some $\mathbf{a} \in HW(B)$, $n_0 \in N$, and $i + 1 \leq n_0$. It can be inferred that the length of p-iterative term for $i + 1$ iteration:

$$|g^{i+1}(\mathbf{a})| \leq |g^i(\mathbf{a})| + C_g \leq |g^0(\mathbf{a})| + (i+1) \cdot C_g \leq |\mathbf{a}| + (i+2) \cdot C_g$$

and for any $i \leq n_0$:

$$|g^i(\mathbf{a})| \leq |g^0(\mathbf{a})| + n_0 \cdot C_g \leq |\mathbf{a}| + (n_0 + 1) \cdot C_g \leq (|\mathbf{a}| + n_0) \cdot (C_g + 1) + C_g \leq z(|\mathbf{a}| + n_0)$$

where

$$z(|\mathbf{x}|) = (C_g + 1) \cdot |\mathbf{x}| + C_g. \tag{14}$$

The next step is to calculate the computational complexity $r(t(\mathbf{x}, n))$ of the p-iterative term. The algorithm is the following for some fixed $\mathbf{a}$ and n_0:

step 0: Calculate $g^0(\mathbf{a})$ (it is necessary to calculate $g(\mathbf{a})$) and check the truth of the L_0-formula $\varphi(g^0(\mathbf{a}))$ on the model $HW(\mathfrak{B})$. If L_0-formula is true, then leave the algorithm running and send value $g^0(\mathbf{a})$; otherwise, go to the next step.

step 1: Calculate $g^1(\mathbf{a})$ (it is necessary to calculate $g(g^0(\mathbf{a}))$), where $g^0(\mathbf{a})$ is known on step 0 and check the truth of the L_0-formula $\varphi(g^1(\mathbf{a}))$ on the model $HW(\mathfrak{B})$. If L_0-formula is true, then leave the algorithm running and send value $g^1(\mathbf{a})$; otherwise, go to the next step.

...

step i: Calculate $g^i(\mathbf{a})$ (it is necessary to calculate $g(g^{i-1}(\mathbf{a}))$), where $g^{i-1}(\mathbf{a})$ is known on step $i-$ 1 and check the truth of the L_0-formula $\varphi(g^i(\mathbf{a}))$ on the model $HW(\mathfrak{B})$. If L_0-formula is true, then leave the algorithm running and send value $g^i(\mathbf{a})$, otherwise go to the next step.

...

step n: Calculate $g^{n_0}(\mathbf{a})$ (it is necessary to calculate $g(g^{n_0-1}(\mathbf{a}))$), where $g^{n_0-1}(\mathbf{a})$ is known on step $n-1$ and check the truth of the L_0-formula $\varphi(g^{n_0}(\mathbf{a}))$ on the model $HW(\mathfrak{B})$. If L_0-formula is true, then leave the algorithm running and send value $g^{n_0}(\mathbf{a})$, otherwise send *nil*.

Let $\mathbf{w}$ be $g^i(\mathbf{a})$ and $|g^i(\mathbf{a})| \leq z(|\mathbf{a}| + n_0)$, as $r(g(\mathbf{w})) \leq s(|\mathbf{w}|)$ it can be inferred that:

$$r(t(\mathbf{a}, n_0)) \leq \sum_{i=0}^{n_0} (s(z(|\mathbf{a}| + n_0)) + u(z(|\mathbf{a}| + n_0))) \tag{15}$$

and get inequality:

$$r(t(\mathbf{a}, n_0)) \leq (s(z(|\mathbf{a}| + n_0)) + u(z(|\mathbf{a}| + n_0))) \cdot (n_0 + 1) \leq d(|\mathbf{a}| + n_0)$$

for some polynomial $d(|\mathbf{x}|)$ and polynomial $z(|x|)$ from (14). □

Corollary 1. *Let the conditions of Theorem 1 be satisfied and $f(|x|)$ be some polynomial. Then, any p-iterative term of the form $t(x, f(|x|))$ is a p-computable function relative to variable x.*

Proof. The condition of Theorem 1 implies that there exists a polynomial $z(\mathbf{x})$ from (14)

$$r(t(\mathbf{x}, f(|\mathbf{x}|))) \leq z(|\mathbf{x}| + f(|\mathbf{x}|)) \leq z(w(|\mathbf{x}|)) \leq v(|\mathbf{x}|)$$

where polynomial $w(|\mathbf{x}|)$ has a form $|\mathbf{x}| + f(|\mathbf{x}|)$ and polynomial $v(|\mathbf{x}|)$ has a form $z(w(|\mathbf{x}|))$. □

Definition 3. *Define a new language L. Language L extends L_0 by p-iterative terms. Classes L-formulas and L-programs extend the classes of L_0-formulas and L_0-programs, respectively.*

Theorem 2. *Let $HW(\mathfrak{B})^*$ be a p-computable extension of the p-computable model $HW(\mathfrak{B})$ of the signature σ^*. Then, any L-program has polynomial computational complexity on $HW(\mathfrak{B})^*$.*

Proof. To prove this statement, it is nessesary to use induction on the number of distinct p-iterative terms for some L-program $t(\mathbf{x})$:

Base of induction $n = 0$: $t(\mathbf{x})$ does not contain a p-iterative term. Then computation complexity of the $t(\mathbf{x})$ is polynomial; this follows immediately from the work [10].

Induction step: Let the statement be true for $n = k$; show this for $n = k + 1$. Let $t(\mathbf{x})$ be a L-program with $k + 1$ distinct p-iterative terms. Let $HW(\mathfrak{B})^{**}$ be enrichment of the model $HW(\mathfrak{B})^*$ of the signature $\sigma^{**} = \sigma^* \cup \{t_1\}$, where p-iterative term t_1 is involved in construction $t(\mathbf{x})$. In the new model $HW(\mathfrak{B})^{**}$, L-program $t(\mathbf{x})$ has only k distinct p-iterative terms and by induction step, the L-program $t(\mathbf{x})$ has a polynomial complexity. □

5. Polynomiality via *p*-Iterative Terms

Let $f(\mathbf{x})$ be a p-computable function, and let $h(|\mathbf{x}|)$ be a polynomial, such that $r(f(\mathbf{x})) \leq h(|\mathbf{x}|)$.

Let the universe of the model $HW(\mathfrak{B})$ contain the natural numbers N in the main set. Signature σ contains the constants 0 and 1, Σ alphabet, and R and L, contain multiplication $\times$ and addition $+$ operations on N, operation of string concatenation *concat*, and operation of string length $(|\ |)$.

Remark 5. *The new operations $\times$, $+$, concat, and $|\ |$ are polynomial.*

For any polynomial $h(|\mathbf{x}|)$, there is a suitable L-program. Let $\underline{q_{i_1}}$ be a L-program of the form $concat(concat(\ldots concat(q, q) \ldots), q)$, where the function *concat* is used $i_1 + 1$-times. Then, for each element of the form $< q_{i_1}, s_{j_1}, q_{i_2}, s_{j_2}, \beta >$ from the Turing machine program P_{T_f}, there is a suitable L-program $v(\underline{q_{i_1}}, s_{j_1}, \underline{q_{i_2}}, s_{j_2}, \beta)$ of the form:

$$v(\underline{q_{i_1}}, s_{j_1}, \underline{q_{i_2}}, s_{j_2}, \beta) = cons(cons(cons(cons(cons(nil, \underline{q_{i_1}}), s_{j_1}), \underline{q_{i_2}}), s_{j_2}), \beta) \tag{16}$$

Theorem 3. *For any p-computable function, there is an L-program defining this function.*

Proof. Let f be some p-computable function, $h(|\mathbf{x}|)$ some polynomial such that:

$$r(f(\mathbf{x})) \leq h(|\mathbf{x}|) \tag{17}$$

Consider the Turing machine T_f over alphabet Σ with program P_{T_f} that realizes the function f. Let us construct a list l_f of terms of the form $v(q_{i_1}, s_{j_1}, q_{i_2}, s_{j_2}, \beta)$ use (16) from the program P_{T_f}, where $\beta \in \{R, L\}$. Then, L-formula $\varphi(\mathbf{x})$ has a form:

$$\varphi(\mathbf{x}) : \ Final(\mathbf{x}) \tag{18}$$

where predicate $Final(w)$ is true if w is a machine word of the form

$$<< s_{-m}, \ldots, s_{-1} >, \underline{q_0}, < s_1, \ldots, s_k >>$$

It is apparent that the predicate $Final(\mathbf{x})$ has a polynomial complexity.

Define $tail^k(\mathbf{x})$ as applying *tail* operation k times to $\mathbf{x}$. A conditional term $g(\mathbf{x})$ [10] will be used for constructing a final L-program:

$$g(\mathbf{x}) = \begin{cases} \mathbf{x} \otimes l_1, \text{where } (l_1 = first(l_f))\&(\mathbf{x} \otimes l_1 \neq nil) \\ \ldots \\ \mathbf{x} \otimes l_i, \text{where } (l_i = head(tail^{k-i}(l_f)))\&(\mathbf{x} \otimes l_i \neq nil) \\ \ldots \\ \mathbf{x} \otimes l_k, \text{where } (l_k = head(l_f))\&(\mathbf{x} \otimes l_k \neq nil) \\ \mathbf{x}, \text{otherwise} \end{cases} \tag{19}$$

Define $mw(\mathbf{x})$ as the L-program that transforms a word $\mathbf{w}$ to the machine word of the form $< nil, \underline{q_1}, \mathbf{w} >$. This L-program has a form:

$$mw(\mathbf{x}) = cons(cons\_l(cons(nil, \underline{q_1}), nil), strList(\mathbf{x})) \tag{20}$$

L-program $mw(\mathbf{x})$ transforms the word $\mathbf{x}$ to the machine word for the initial configuration c_0 of the Turing machine T_f.

Define $value(\mathbf{x})$ as the L-program that transforms a machine word w into the word on the tape of the Turing machine T_f. This function is constructed as follows:

$$value(w) = listStr(conc(first(w), head(w))) \tag{21}$$

Define p-iterative term $t(mw(\mathbf{x}), h(|\mathbf{x}|))$ using construction (12) with the L-program $g(x)$ of the form (19), the formula φ from (18), the polynomial $h(|\mathbf{x}|)$ from (17), and L-programs from (20) and (21).

The final L-program representing the function $f(\mathbf{x})$ has the form:

$$value(t(mw(\mathbf{x}), h(|\mathbf{x}|))) \tag{22}$$

Note that the L-program $t(mw(\mathbf{x}), h(|\mathbf{x}|))$ satisfies the conditions of Theorem 1 and, therefore, $value(t(mw(\mathbf{x}), h(|\mathbf{x}|)))$ is a p-computable. □

6. Conclusions

The work shows the equality of classes P and L. The main motivation was to create a not Turing complete logical programming language describing the class of polynomial algorithms. Programs in this language are logical terms and have polynomial complexity. For any polynomial algorithm, there is a program describing it. One of the main contributions of this work is the construction of a new logical language L that is equal to the class P. Another contribution is the construction of a p-iterative term for this. The main limitation is that this language is not Turing complete. Therefore, it is impossible to realize algorithms on it with the complexity being higher than polynomial.

Thus, language L is rich enough to describe any algorithms of polynomial complexity. These results are one more step in the construction of high-level programming languages based on logical language L. Moreover, programs in such languages will remain polynomially computable. It means that programs stop running every time, work quickly, and produce results. It is especially important during the development of blockchain technologies and smart contracts. Since smart contracts are programs in a distributed environment, the correct functioning of the entire blockchain as a whole depends on the result of its execution. Such smart contracts should be executed quickly and should not consume a lot of computing resources.

The work has built a logical language that allows one to create fast and reliable programs. These programs will be used in computer science, robotics, the Internet of things, blockchain technologies, medicine, and artificial intelligence.

High-quality artificial intelligence requires not only just neural networks and machine learning, but also logical rules and their execution. An effective solution is the hybrid technologies of neural networks and logical rules that will make a breakthrough in the future. To construct such logical rules, the semantic programming theory suits perfectly well.

Author Contributions: Conceptualization, S.G. and A.N.; methodology, S.G. and A.N.; formal analysis, S.G.; validation, S.G.; investigation, S.G. and A.N.; writing—original draft preparation, A.N.; writing—review and editing, A.N.; supervision, S.G.; project administration, S.G.; software, A.N. All authors have read and agreed to the published version of the manuscript.

Funding: Goncharov research was funded by the project of Fundamental Research of the Sobolev institute of mathematics SB RAS number 0314-2019-0002.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ershov, Y.L.; Goncharov, S.S.; Sviridenko, D.I. Semantic programming. In Proceedings of the Information Processing 86: Proceedings of the IFIP 10th World Computer Congress, Dublin, Ireland, 1–5 September 1986; Elsevier Sci.: Dublin, Ireland, 1986; Volume 10, pp. 1113–1120.
2. Goncharov, S.S.; Sviridenko, D.I. Σ-programming. *Transl. II Ser. Am. Math. Soc.* **1989**, *142*, 101–121 .
3. Ershov, Y.L. *Definability and Computability*; Springer: Berlin/Heidelberg, Germany, 1996.
4. Cenzer, D.; Remmel, J. Polynomial-time versus recursive models. *Ann. Pure Appl. Log.* **1991**, *54*, 17–58. [CrossRef]
5. Lewis, H.; Papadimitriou, C. *Elements of the Theory of Computation*; Prentice-Hall: Upper Saddle River, NJ, USA, 1998.
6. Mantsivoda, A.; Ponomaryov, D. *A Formalization of Document Models with Semantic Modelling*; Series Mathematics; The Bulletin of Irkutsk State University: Irkutsk, Russia, 2019; Volume 27, pp. 36–54. [CrossRef]
7. Gumirov, V.; Matyukov, P.; Palchunov, D. Semantic Domain-specific Languages. In Proceedings of the International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON), Novosibirsk, Russia, 21–27 October 2019; pp. 955–960. [CrossRef]
8. Kalimullin, I.; Melnikov, A.; Ng, K. Algebraic structures computable without delay. *Theor. Comput. Sci.* **2017**, *674*, 73–98. [CrossRef]
9. Alaev, P.E. Structures Computable in Polynomial Time. I. *Algebra Log.* **2017**, *55*, 421–435. [CrossRef]
10. Goncharov, S.S.; Sviridenko, D.I. Logical language of description of polynomial computing. *Dokl. Math.* **2019**, *99*, 121–124. [CrossRef]
11. Goncharov, S.; Nechesov, A. Polynomial analogue of Gandy's fixed point theorem. *Mathematics* **2021**, *9*, 2102. [CrossRef]
12. Ospichev, S.S.; Ponomaryov, D.K. On the complexity of formulas in semantic programming. *Sib. Electron. Math. Rep.* **2018**, *15*, 987–995. [CrossRef]
13. Goncharov, S.S. Conditional Terms in Semantic Programming. *Sib. Math. J.* **2017**, *58*, 794–800. [CrossRef]