



Article

# Functional Variant of Polynomial Analogue of Gandy's Fixed Point Theorem

Andrey Nechesov <sup>1,\*</sup>  and Sergey Goncharov <sup>2,†</sup> <sup>1</sup> The Artificial Intelligence Research Center of Novosibirsk State University, Novosibirsk 630090, Russia<sup>2</sup> Department of Discrete Mathematics and Computer Science, Novosibirsk State University, Novosibirsk 630090, Russia; s.s.goncharov@mail.ru

\* Correspondence: nechesoff@gmail.com; Tel.: +7-913-480-37-63

† These authors contributed equally to this work.

**Abstract:** In this work, a functional variant of the polynomial analogue of Gandy's fixed point theorem is obtained. Sufficient conditions have been found to ensure that the complexity of recursive functions does not exceed polynomial bounds. This opens up opportunities to enhance the expressivity of p-complete languages by incorporating recursively defined constructs. This approach is particularly relevant in the following areas: AI-driven digital twins of smart cities and complex systems, trustworthy AI, blockchains and smart contracts, transportation, logistics, and aerospace. In these domains, ensuring the reliability of inductively definable processes is crucial for maintaining human safety and well-being.

**Keywords:** polynomial computability; Gandy's fixed point theorem; artificial intelligence; smart cities

**MSC:** 03D15; 68Q17



**Citation:** Nechesov, A.; Goncharov, S. Functional Variant of Polynomial Analogue of Gandy's Fixed Point Theorem. *Mathematics* **2024**, *12*, 3429. <https://doi.org/10.3390/math12213429>

Academic Editor: Salvatore A. Marano

Received: 20 September 2024

Revised: 10 October 2024

Accepted: 24 October 2024

Published: 31 October 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The volume of program code generated by programmers continues to increase exponentially each year [1,2]. With large language models now capable of writing code and developing applications, this growth will accelerate even further and is likely to surpass hundreds of trillions of lines of code in the near future [3,4]. As these programs significantly influence our daily lives, there is a potential for them to cause harm if they become stuck in infinite loops [5,6] or operate exponentially in time [7,8] relative to the input data size. Consequently, polynomial computational complexity [9–12] becomes a crucial criterion for developing programs that can be reliably executed in critical areas such as trustworthy AI [13], blockchains and smart contracts [14], digital twins in smart cities [15], autonomous driving [16–18], robotics [19,20], and other advanced technologies. It is essential to carefully select p-complete subsets of Turing-complete languages [21–24] with maximum expressive capabilities to ensure the reliability and efficiency of these systems. To achieve this, it is necessary to define recursive functions [25–31] in such a way that their computational complexity remains polynomial.

In previous works, we established a polynomial analogue of Gandy's fixed point theorem [9,10]. This discovery served as a crucial impetus for the development of p-complete programming languages, one of which was pioneered by us in [12]. A key achievement of this work was the solution of the problem  $P = L$  [12]. Subsequently, our efforts have been focused on advancing the creation of a novel high-level programming language and exploring the concept of a comprehensive programming methodology [32]. However, one aspect of our research remains unaddressed: the characterization of classes of recursive functions with polynomial complexity. We found sufficient conditions for such functions in this work. In many ways, the main ideas of this result are similar to the ideas that we used in the proof of the polynomial analogue of Gandy's fixed point theorem [9,10].

However, there are also striking differences. Functions, as such, differ quite strongly from predicates precisely in the multitude of their values. If a predicate is either true or false, then a function can generally take on a variety of values.

Therefore, the challenge for us over the recent three years has been to identify these constraints on recursive functions, which would be sufficiently soft for the class of functions to remain large, while simultaneously tough enough to prevent them from exceeding polynomial complexity. This task became necessary following the proof of a polynomial analogue of Gandy’s fixed point theorem in the context of predicates extensions. Now we will inductively expand the function itself, not the truth set of the predicate, which is what fundamentally distinguishes these two approaches.

## 2. Preliminaries

In this section, we will delve into the preliminary findings that pave the way for the proof of the FPAG-theorem. We will formulate the classical version of Gandy’s fixed point theorem [33,34], along with its polynomial analogue. The construction of the p-complete language L can be traced back to ref. [12], where L is defined as a collection of formulas and programs, which are constructed inductively through the use of novel term constructions such as conditional terms and p-iterative terms [12]. Therefore, the very concept of a formula changes.

### 2.1. Classical Gandy’s Fixed Point Theorem

Classical Gandy’s fixed point theorem [9,33,34] is one of the most important not only in mathematical logic but also in programming and operator theory. One of the variants of the classical Gandy’s fixed point theorem states that if we have a fixed model  $\Omega = \langle \omega, 0, s, +, \cdot, \leq \rangle$  of the signature  $\sigma_0 = \langle 0, s^1, +^2, \cdot^2, \leq^2 \rangle$ . Denote the extension of the signature  $\sigma_0$  by  $\sigma^*$ , which is obtained by adding symbols for all  $\Sigma$ -functions on  $\Omega$  and constant symbols for all elements of  $\omega$ , and let  $\Omega^*$  denote the corresponding enrichment of  $\Omega$ .

Define the operator  $\Gamma_{\Phi[\bar{x}]}^{\Omega^*}$  as follows:

$$\Gamma_{\Phi[\bar{x}]}^{\Omega^*}(Q) = \{(a_1, \dots, a_{k-1}) \mid \langle \Omega^*, Q \rangle \models \Phi(a_0, \dots, a_{k-1})\}$$

where  $Q \subseteq \omega^k$ .

We associate the following sequence of predicates

$$\Gamma_0, \Gamma_1, \dots, \Gamma_\alpha \text{ where } \alpha \text{ is an ordinal}$$

with the monotone operator  $\Gamma_{\Phi[\bar{x}]}^{\Omega^*}$  as follows:

$$\Gamma_0 = \emptyset, \Gamma_{\alpha+1} = \Gamma_{\Phi[\bar{x}]}^{\Omega^*}(\Gamma_\alpha) \text{ for none limit ordinal, } \dots, \Gamma_\alpha = \bigcup_{\beta < \alpha} \Gamma_\beta \text{ for limit ordinal}$$

Let  $\alpha$  be the smallest ordinal such that  $\Gamma_{\alpha+1} = \Gamma_\alpha$ ; thus,  $\Gamma^* = \Gamma_\alpha$  is a smallest fixed point.

**Theorem 1** (Gandy’s fixed point theorem). *Let  $\Phi(P^+)$  be a  $\Sigma$ -formula of the signature  $\sigma^* \cup \{P^{(k)}\}$  in which the predicate symbol  $P$  enters positively and  $x_0, \dots, x_{k-1}$  be a list of the different free variables of the formula  $\Phi$ . Thus, the smallest fixed point  $\Gamma^*$  of the operator  $\Gamma_{\Phi[\bar{x}]}^{\Omega^*}$  is a  $\Sigma$ -predicate on  $\Omega^*$ .*

The concept of a  $\Sigma$ -predicate inherently implies a level of semi-decidability. Our goal was to achieve decidability and, ideally, to verify whether an element belongs to a smallest fixed point  $\Gamma^*$  within polynomial time.

### 2.2. PAG-Theorem

Now let  $\Sigma_0$  be some finite alphabet, and  $\Sigma = \Sigma_0 \cup \{<, >\} \cup \{, \}$  is a new alphabet obtained by adding new symbols (brackets and comma) to  $\Sigma_0$ . *Word splitting* is the following partial function  $R : \Sigma^* \rightarrow (\Sigma \cup \{\#\})^*$ , such that

$$R(w) = \begin{cases} w_1\#\dots\#w_n, & \text{where } w = \langle w_1, \dots, w_n \rangle \text{ and every } w_i \in \Sigma^* \text{ satisfies 1) or 2)} \\ \uparrow, & \text{otherwise} \end{cases}$$

- (1)  $w_i \in \Sigma_0^*$
- (2)  $w_i$  starts with a left bracket and the number of left brackets in the word is equal to the number of right brackets, while for any initial subword  $\alpha_i$  such that  $w_i = \alpha_i\beta_i$  is not implemented, where the word  $w_i$  can be represented as some concatenation of the words  $\alpha_i, \beta_i \in \Sigma^*$  and  $|\alpha_i| \geq 1$ .

Inductively define the notion *rank of element*  $r(w)$  for  $w \in \Sigma^*$ :

$$r(w) = \begin{cases} \sup\{r(w_1), \dots, r(w_k)\} + 1, & \text{if } R(w) = w_1\#\dots\#w_k \\ 0, & \text{otherwise.} \end{cases}$$

The polynomial analogue of Gandy’s fixed point theorem (PAG-theorem) [9,10] is based on the classical Gandy’s fixed point theorem. The PAG-theorem demonstrates that the smallest fixed point of a special operator is already computable in polynomial time (p-computable).

To prove this, a p-computable hereditary-finite list superstructure  $HW(\mathfrak{M})$  of signature  $\sigma$  was chosen as the basic model. The base set  $HW(M) \subseteq \Sigma^*$  contains hereditary-finite lists, which are inductively constructed from the basic elements of the base set  $M \subseteq \Sigma_0^*$  of the model  $\mathfrak{M}$  of the signature  $\sigma_0 \subset \sigma$ . The basic list operations for  $HW(\mathfrak{M})$  are

- $head(x)$ —operation of taking the last element from a list  $x$ .
- $tail(x)$ —operation of removing the last element from a list  $x$ .
- $cons(x, y)$ —operation of adding to the end of a list  $x$  of a list  $y$ .
- $conc(x, y)$ —concatenation operation of two lists  $x$  and  $y$ , respectively.

There is a constant  $nil$  for the empty list and there are also the following relations:

- $x \in y$ —“ $x$  is an element of  $y$ ”.
- $x \subseteq y$ —“ $x$  is an initial segment of  $y$ ”.

Formulas of the first-order logic are considered only with the bounded quantifiers  $\forall x \in t, \exists x \in t, \forall x \subseteq t, \exists x \subseteq t$ , where  $t$  is a standard term (in addition to standard terms, the language  $L$  also has conditional terms and p-iteration terms [12]).

In order to apply the PAG-theorem generally, it is necessary to construct a p-computable GNF-system [10] based on the following components:

- A finite alphabet  $\Sigma$ .
- An extended alphabet  $\Omega$ .
- A special logical language  $L$  where  $L$ -formulas and  $L$ -programs are defined [12,32].
- p-computable hereditary-finite list superstructure  $HW(M)$  of the signature  $\sigma$ .
- Finite sets of extendable predicates  $P_1, \dots, P_n$ .
- Generative families of formulas  $F_{P_1}, \dots, F_{P_n}$ .
- p-computable functions  $\gamma_1, \dots, \gamma_n$  that, given an element of  $HW(M)$ , construct suitable generative formulas or return false.

**Theorem 2** (Polynomial analogue of Gandy’s fixed point theorem). *Let  $G$  be a p-computable GNF-system; then, the smallest fixed point  $\Gamma^*$  of the operator  $\Gamma_{F_{P_1}, \dots, F_{P_n}}^{<HW(\mathfrak{M}), \sigma>}$  is a p-computable.*

The results obtained in the PAG-theorem allow us to inductively define sets of objects of varying complexity using generative families of  $L$ -formulas and at the same time guarantee their recognizability in polynomial time. Inductivity upwards essentially defines

recursivity downwards, i.e., the algorithm that checks whether an object belongs to a class of objects, working with various list encoded objects, each time splits the list into elements and recursively checks each element for consistency with one of the families of generative formulas.

### 3. Functional Variant of the PAG-Theorem

#### 3.1. *p*-Computable FGNF-Systems

Let  $\langle HW(\mathfrak{M}), \sigma \rangle$  [9,10] be a *p*-computable hereditary-finite list superstructure. Let  $f_1, \dots, f_n$ -*p*-computable functions and all  $f_i$  in the signature  $\sigma$ . Then, we can define a new finite or countable generative families  $T_{f_i}$  of standard terms of the signature  $\sigma$  for each function  $f_i$ :

$$T_{f_i} = \{t_j(x_1, \dots, x_{k_j}) \mid j \in N\}, i \in [1, \dots, n]$$

by default, we assume that any  $f_j$  can be included in any term  $t_k$  of any family  $T_{f_i}$ .

Assume also that for each generative family of terms  $T_{f_i}$ , there exists a *p*-computable function  $\gamma_i$ :

$$\gamma_i : HW(M) \rightarrow T_{f_i} \cup \{false\}$$

By default, we assume that the *false* element belongs to  $M$ . In the future, if the function  $f : HW(M) \cup M \rightarrow HW(M) \cup M$  has the value *false* on some element  $w \in HW(M) \cup M$ , then we denote this as  $f(w) \uparrow$ , otherwise  $f(w) \downarrow$ . The domain  $dom(f)$  of a function  $f$  will be considered as the set of all elements  $w$  on which  $f(w) \downarrow$ .

Define the extension  $f_i^{(n+1)} : HW(M) \cup M \rightarrow HW(M) \cup M$  for each  $f_i^{(n)}$  (where  $f_i^{(0)} = f_i$ ) and

$$f_i^{(n+1)} \upharpoonright_{dom(f_i^{(n)})} = f_i^{(n)}$$

using a *p*-computable function  $\gamma_i : HW(M) \rightarrow T_{f_i} \cup \{false\}$  such that on  $(n + 1)$ -iteration we have

$$f_i^{(n+1)}(w) = \begin{cases} f_i^{(n)}(w), & \text{if } f_i^{(n)}(w) \downarrow, \text{ else} \\ \gamma_i(w) \frac{\bar{x}}{\bar{w}}, & \text{if } \gamma_i(w) \downarrow \text{ and } \gamma_i(w) \frac{\bar{x}}{\bar{w}} \text{ is defined, else} \\ false, & \text{otherwise} \end{cases} \tag{1}$$

Using generative families of the terms  $T_{f_i}$ , *p*-computable functions  $\gamma_i$  and Equation (1), we can define the operator, as follows:

$$\Gamma_{f_1, \dots, f_n}^{\langle HW(\mathfrak{M}), \sigma \rangle} : \{g \mid g : HW(M) \rightarrow HW(M)\}^n \rightarrow \{g \mid g : HW(M) \rightarrow HW(M)\}^n$$

satisfying the following rules:

$$\Gamma_{f_1, \dots, f_n}^{\langle HW(\mathfrak{M}), \sigma \rangle} (g_1^{(k)}, \dots, g_n^{(k)}) = (g_1^{(k+1)}, \dots, g_n^{(k+1)})$$

where  $g_i^{(k)} \subseteq g_i^{(k+1)}$ .

Let us define a partial order:

$$(f_1, \dots, f_n) \subseteq (g_1, \dots, g_n) \Leftrightarrow dom(f_i) \subseteq dom(g_i) \text{ and } f_i = g_i \upharpoonright_{dom(f_i)}$$

The operator  $\Gamma_{f_1, \dots, f_n}^{\langle HW(\mathfrak{M}), \sigma \rangle}$  is monotonic [33] by the partial-order  $\subseteq$  and has the property of a fixed point  $(g_1^*, \dots, g_n^*)$ ; moreover, the fixed point is reached in  $\omega$  steps [9]:

$$\Gamma_{f_1, \dots, f_n}^{\langle HW(\mathfrak{M}), \sigma \rangle} (g_1^*, \dots, g_n^*) = (g_1^*, \dots, g_n^*)$$

where  $g_i^* = g_i^{(\omega)}$ ,  $i \in [1, \dots, n]$

Define a set of free variables for terms and formulas as  $V(t(\bar{x}))$  and  $V(\varphi(\bar{x}))$ , respectively, where  $\bar{x} = (x_1, \dots, x_n)$  for some  $n \in N$ . Denote  $t^{\bar{y}}(\bar{x}, \bar{y})$  as a term where all occur-

rences  $f_j(x_i)$  have been replaced on  $y_i$  in the term  $t(\bar{x}, \bar{y})$ . Denote  $V_{\bar{x}}(t(\bar{x}, \bar{y}))$  and  $V_{\bar{y}}(t(\bar{x}, \bar{y}))$  as the set of free variables  $x_i$  from  $\bar{x}$  and the set of free variables  $y_i$  from  $\bar{y}$ , respectively.

We define the concept of the FGNF-system as a tuple, similarly to the way in which we defined this in the case of the polynomial analogue of Gandy’s fixed point theorem [10]:

$$FGNF = (\Sigma, \sigma, \Omega, L, HW(M), F, T, G)$$

We will say that a FGNF-system is  $p$ -computable if the model  $HW(\mathfrak{M})$  of the signature  $\sigma$  is  $p$ -computable, and all  $\gamma_i \in G$  are  $p$ -computable functions and if the following conditions are met:

- If  $f_j$  is included in some  $t_k$ , then  $f_j$  appears in the term  $t_k$  as  $f_j(x_i)$  for some particular variable  $x_i$ .
- $f_j(x_i)$  and  $f_k(x_i)$  do not belong to any term simultaneously for any  $x_i$  where  $j \neq k$ .
- It is not true that there is  $i$  such that  $x_i \in V_{\bar{x}}(t_j^{\bar{y}}(\bar{x}, \bar{y}))$  and  $y_i \in V_{\bar{y}}(t_j^{\bar{y}}(\bar{x}, \bar{y}))$  for some  $t_j(\bar{x})$
- For any  $T_{f_k}$ , there exist  $C$  and  $p$  such that for any  $t_j \in T_{f_k}$  and for any evaluations  $\bar{w}, \bar{l}$ :

$$t(t_j^{\bar{y}}(\bar{w}, \bar{l})) \leq C \cdot (|conc(list(\bar{w}), list(\bar{l}))|^p, \text{if } t_j^{\bar{y}}(\bar{w}, \bar{l}) \downarrow \tag{2}$$

where  $t(t_j^{\bar{y}}(\bar{w}, \bar{l}))$  is a computational complexity of  $t_j^{\bar{y}}(\bar{w}, \bar{l})$ , and  $list(\bar{w})$  is a conversion function that converts a tuple into a list.

- If  $V_{\bar{y}}(t_j^{\bar{y}}(\bar{x}, \bar{y})) \neq \emptyset$ , then

$$|t_j^{\bar{y}}(\bar{w}, \bar{l})| \leq C \cdot |list(\bar{w})|^p + |list(\bar{l})|, \text{if } t_j^{\bar{y}}(\bar{w}, \bar{l}) \downarrow \tag{3}$$

for any  $t_j \in T_{f_k}$ , and for any evaluations  $\bar{w}, \bar{l}$  of the tuples  $\bar{x}$  and  $\bar{y}$ , respectively.

- For any  $T_{f_k}$ , and for any  $\bar{w}_1, \bar{w}_2 \in HW(M)^n$ , there are no two terms  $t_i$  and  $t_j$  from  $T_{f_k}$  where  $t_i, t_j \in T_{f_k}$  such that  $t_i^{\bar{y}}(\bar{w}_1, \bar{w}_2) \downarrow$  and  $t_j^{\bar{y}}(\bar{w}_1, \bar{w}_2) \downarrow$ .

### 3.2. Functional Variant of Polynomial Analogue of Gandy’s Fixed Point Theorem (FPAG-Theorem)

**Theorem 3** (FPAG-theorem). *Let the  $p$ -computable FGNF-system with initial  $p$ -computable functions  $f_1, \dots, f_n$  be given. Thus, the smallest fixed point  $\Gamma^* = (f_1^*, \dots, f_n^*)$  of the operator  $\Gamma_{f_1, \dots, f_n}^{<HW(\mathfrak{M}), \sigma>}$  is  $p$ -computable.*

**Proof.** By induction on the rank  $r(w)$  of the list element  $w$ , we prove two statements simultaneously for some fixed constants  $C$  and  $p$ . First,

$$\forall i \forall w \in HW(M) |f_i(w)| \leq C \cdot |w|^p$$

and, second, we obtain that for any  $i$ , the computational complexity of the algorithm does not exceed the following:

$$\forall i \forall w t(f_i(w)) \leq 36 \cdot C^{p+1} \cdot (r(w) + 1) \cdot |w|^{2p^2+1}$$

Base of induction:  $n = 0$ . Then, for any  $a \in M$ , we have  $|f_i(a)| \leq C \cdot |a|^p$  and  $t(f_i(a)) \leq C \cdot |a|^p$  because the initial function  $f_i$  is a polynomial computable.

Step of induction: Let the statement be true for every  $k < n$ ; then, show that for  $k = n$ , we have the following:

If the initial polynomial function is  $f_i(w) \downarrow$ , then the answer is obvious.

Else if  $\gamma(w) = false$ , then  $f_i(w) \uparrow$ .

Else if  $\gamma(w) = t_j(\bar{x})$ , we have two cases for  $I = \{i | y_i \in V(t_j^{\bar{y}}(\bar{x}, \bar{y}))\}$ :

(1) If  $I = \emptyset$ , then by conditions for p-computable FGNF-system,

$$|f_i^*(w)| \leq |[\gamma(w)](\bar{w})| = |t_j(w_1, \dots, w_n)| \leq C \cdot |w|^p$$

and it is true by definition of p-computable FGNF-system.

The complexity (see work [9]) is estimated as follows:

$$\begin{aligned} t(f_i^*(w)) &\leq t(f_i(w)) + t(\gamma_i(w)) + t([\gamma_i(w)]_{\bar{w}}^{\bar{x}}) + t([\gamma_i(w)](\bar{w})) \leq \\ &\leq C \cdot |w|^p + C \cdot |w|^p + 12 \cdot C \cdot |w|^{p+1} + C \cdot |w|^p \leq \\ &\leq 36 \cdot C^{p+1} \cdot (r(w) + 1) \cdot |w|^{2p^2+1} \end{aligned}$$

where  $t([\gamma_i(w)]_{\bar{w}}^{\bar{x}})$  is the complexity of replacing variables  $x_i$  on values  $w_i$ .

(2) If  $I \neq \emptyset$ , then

$$|f_i^*(w)| = |[\gamma_i(w)](\bar{w})| = |[\gamma_i(w)]^{\bar{y}}(\bar{w}_x, \bar{v}_y)| \leq C \cdot |list(\bar{w}_x)|^p + |list(\bar{v}_y)| \leq C \cdot |\bar{w}|^p$$

where  $v_i = f_{j_i}(w_i)$  for  $v_i \in \bar{v}_y$  and  $|v_i| = |f_{j_i}(w_i)| \leq C \cdot |w_i|^p$  by induction.

For complexity, we have the following:

$$\begin{aligned} t(f_i^*(w)) &\leq t(f_i(w)) + t(\gamma_i(w)) + t([\gamma_i(w)]_{\bar{w}}^{\bar{x}}) + \\ &+ \sum_{i \in I} t(f_{j_i}^*(w_i)) + t([\gamma_i(w)]_{\bar{w}}^{\bar{x}})_{\{v_i | i \in I\}}^{\{f_{j_i}(w_i) | i \in I\}} + t([\gamma_i(w)]^{\bar{y}}(\bar{w}_x, \bar{v}_y)) \leq \\ &\leq C \cdot |w|^p + C \cdot |w|^p + 12 \cdot C \cdot |w|^p \\ &+ \sum_{i \in I} 36 \cdot C^{p+1} \cdot ((r(w) - 1) + 1) \cdot |w|^{2p^2+1} + 12 \cdot C^2 \cdot |w|^{2p^2+1} + C \cdot (C \cdot |w|^p)^p \leq \\ &\leq 36 \cdot C^{p+1} \cdot (r(w) + 1) \cdot |w|^{2p^2+1} \end{aligned}$$

where  $t([\gamma_i(w)]_{\bar{w}}^{\bar{x}})_{\{v_i | i \in I\}}^{\{f_{j_i}(w_i) | i \in I\}}$  is the complexity of replacing  $f_{j_i}(w_i)$  on  $v_i$  in  $[\gamma_i(w)]_{\bar{w}}^{\bar{x}}$ .

Since the rank  $r(w)$  is less than the length of the list  $|w|$ , our theorem is proven.  $\square$

#### 4. Applications

This section will delve into the main areas in which the FPAG-theorem may be employed, ensuring the preservation of polynomial computational complexity. The fundamental basis of our reality is a complex interaction of inductively defined concepts and recursively defined processes. Consequently, it becomes imperative to master the art of expressing these processes and concepts through mathematical frameworks and methodologies.

##### 4.1. Recursion in p-Complete Languages

One of the principal domains of application for the FPAG-theorem can be identified as p-complete programming languages [12,32]. Previously, we were unable to employ full-fledged recursive calls, as recursion in the general case led us beyond the bounds of polynomiality. Yes, a polynomial analogue of Gandy’s fixed point theorem has been proven [9], but this theorem applies only to predicates where the truth value can be either true or false. In the case of the FPAG-theorem, the value of a function can be arbitrarily large, allowing us to extend our proposed p-complete languages L [12] and L\* [32]. These extensions will be conservative. We can also consider the problem of impoverishing Turing-complete languages where the computational complexity of programs will be polynomial. To do this, we must limit ourselves to the simplest constructions emulating the work of the p-iterative terms and conditional terms proposed by us. In addition, we permit the definition of recursive functions, but only under the same conditions imposed on them for

a p-computable FGNF-system. In order to develop programs of polynomial complexity, we must restrict ourselves of the following syntactical constructs:

- The conditional operator IF THEN ELSE (an analogue of the conditional term).
- The loop operator FOR where the output values of the variables changed in the loop body are limited by some polynomial.
- The loop operator FOR on lists [35].
- Recursive functions that must satisfy the conditions for a p-computable FGNF-system.

To illustrate, let us examine the process of implementing a factorial in Turing-complete languages through our methodology. Firstly, we shall provide a conventional definition of the factorial:

```
function Factorial(n) {
    if (n < 2) then return 1;
    return n*Factorial(n-1);
}
```

The function *Factorial* defined above has exponential computational complexity. This leads to the program running in exponential time. We would like to limit ourselves to polynomial computational complexity. For this purpose, consider the following function *listMul*:

```
function listMul(w) {
    if !list(w) then return w;
    s:= 1;
    for x in w {
        s := s * listMul(x)
    }
    return s;
}
```

where the notation *for x in w* is a loop FOR on the list *w*.

It is worth noting that the *listMul* function can also be employed to determine the factorial of an integer *n*. To accomplish this task, it suffices to enumerate all integers from 1 to *n* in a list *w* in any order. For instance, the enumeration may be as follows:

- $w_1(n) = \langle n, \langle n - 1, \langle \dots \langle 1, \langle 2 \rangle \rangle \dots \rangle \rangle \rangle$
- ...
- $w_k(n) = \langle \langle 1, n \rangle, \dots, \langle n - 1 \rangle, 2 \rangle$

Taking into account the implementation, the value of the function *Factorial*(*n*) coincides with the value of the function *listMul*( $w_j(n)$ ), where  $j \in [1, \dots, k]$ .

**Lemma 1.** *The computational complexity of the function listMul(x) is polynomial.*

**Proof.** The proof follows immediately from the FPAG-theorem where the initial p-computable function *listMul*(*x*) is defined on natural numbers:

$$listMul(n) = n;$$

and where the generative family  $T_{listMul}$  consists of a countable number of terms:

$$\{listMul(x_1) \cdot \dots \cdot listMul(x_n) \mid n \in N\}$$

The p-computable function  $\gamma$  has the following form:

$$\gamma(\langle w_1, \dots, w_k \rangle) = listMul(x_1) \cdot \dots \cdot listMul(x_k)$$

It is necessary to check conditions (2) and (3) for generative families  $T_{listMul}$ .



There exist constants  $C$  and  $p$  for condition (2) such that

$$t_j^{\bar{y}}(l_1, \dots, l_k) = t(l_1 \cdot \dots \cdot l_k) \leq C \cdot | \langle l_1, \dots, l_k \rangle |^p$$

where  $l_1, \dots, l_k \in N$ .

Condition (3) is also true:

$$|t_j^{\bar{y}}(l_1, \dots, l_k)| = |l_1 \cdot \dots \cdot l_k| \leq | \langle l_1, \dots, l_k \rangle |$$

where  $l_1, \dots, l_k \in N$ .

Thus, we can construct the  $p$ -computable FGNF-system, and the FPAG-theorem can be applied.  $\square$

#### 4.2. Blockchains and Smart Contracts

Blockchains and smart contracts represent significant areas of software development. For example, when we examine the smart contracts on the Ethereum blockchain [36], we find that the process of executing a smart contract entails the necessity of assigning a specific gas value for its execution. If there is not enough gas, then the execution of the smart contract is interrupted. This can be critical for both developers and users themselves. This is because smart contracts are essentially programs written in a Turing-complete language, which presents a halting problem that cannot be circumvented. In this context, we envision further progress through the active design of programs with polynomial computational complexity that consistently terminate, and for which the required computing resources can be predetermined. This is accomplished using the methods outlined in the preceding section. However, it is worth noting that the complexity estimates themselves remain polynomial, although there are specific features of the execution of smart contracts on the blockchain.

It is worth noting the phenomenon of mutual recursion in the context of smart contracts, which is of particular interest. Such recursive processes are often challenging to analyze, and it is difficult to forecast their termination. By employing recursion on lists and exercising control over the values of functions, we ensure their eventual termination.

```
function f(w) {
    if !list(w) return 1;
    s:= 1;
    for x in w {
        s:= s * g(x);
    }
    return s;
}
```

```
function g(w){
    if !list(w) return len(w)^2;
    s:= g(first(w));
    for x in w[1:] {
        s:= s + [f(x)/2];
    }
    return s;
}
```

where  $first(w)$  is a first element in the list  $w$ , and  $len(x)$  denotes the length of the string  $x$ .

**Lemma 2.** *Mutually recursive functions  $f$  and  $g$  defined above are  $p$ -computable.*



**Proof.** The value of the initial function  $f$  defined on non-list elements is equal to 1. The value of the initial function  $g$  defined on non-list elements  $w$  is equal to  $|w|^2$ . It follows that the initial functions  $f$  and  $g$  are p-computable.

The generative term family  $T_f$  for the function  $f$  has the following form:

$$\{g(x_1) \cdot \dots \cdot g(x_n) \mid n \in N\}$$

The p-computable function  $\gamma_f$  has the following form:

$$\gamma_f(\langle w_1, \dots, w_k \rangle) = g(x_1) \cdot \dots \cdot g(x_k)$$

The generative term family  $T_g$  for the function  $g$  has the following form:

$$\{g(x_1) + [f(x_2)/2] \dots + [f(x_n)/2] \mid n \in N\}$$

The p-computable function  $\gamma_g$  has the following form:

$$\gamma_g(\langle w_1, \dots, w_k \rangle) = g(x_1) + [f(x_2)/2] \dots + [f(x_k)/2]$$

It is necessary to check conditions (2) and (3) for generative families  $T_f$  and  $T_g$ , respectively. There exist constants  $C$  and  $p$  for condition (2) such that

$$t(t_j^{\bar{y}}(l_1, \dots, l_k)) = t(l_1 \cdot \dots \cdot l_k) \leq C \cdot |\langle l_1, \dots, l_k \rangle|^p$$

where  $t_j(\bar{x}) \in T_f$  and  $l_1, \dots, l_k \in N$ .

$$t(t_m^{\bar{y}}(l_1, \dots, l_h)) = t(l_1 + [l_2/2] \dots + [l_h/2]) \leq C \cdot |\langle l_1, \dots, l_h \rangle|^p$$

where  $t_m(\bar{x}) \in T_g$  and  $l_1, \dots, l_k \in N$ .

Condition (3) is also true:

$$|t_j^{\bar{y}}(l_1, \dots, l_k)| = |l_1 \cdot \dots \cdot l_k| \leq |\langle l_1, \dots, l_k \rangle|$$

where  $t_j(\bar{x}) \in T_f$  and  $l_1, \dots, l_k \in N$ .

$$|t_m^{\bar{y}}(l_1, \dots, l_h)| = t(l_1 + [l_2/2] \dots + [l_h/2]) \leq |\langle l_1, \dots, l_h \rangle|$$

where  $t_m(\bar{x}) \in T_g$  and  $l_1, \dots, l_k \in N$ .

Thus, we can construct a p-computable FGNF-system, and the FPAG-theorem can be applied. It follows that the functions  $f$  and  $g$  are p-computable.  $\square$

### 4.3. Large Language Models

In the recent three years, neural networks [37,38], large language models and generative models [39–41] have elevated the standard of AI solutions to previously unimaginable heights. They have the capability to compose texts, translate languages, create artwork and music, as well as mimic human voice and behavior, which undoubtedly fascinates us as humans. Many claim that ChatGPT [39,42] has already passed the Turing test. However, seemingly simple questions like “How many sisters does Alice’s brother have?” [43] shatter this illusion, revealing a range of underlying issues that are not immediately apparent. Numerous experiments have shown that these systems struggle with mathematical problem solving and the explainability of their outputs. One significant challenge lies in the inductively specified properties and objects that are input to these models. Furthermore, the structure of neural networks and transformer models remains poorly understood concerning explainability and logical reasoning. To date, no major mathematical theorems have emerged that clarify logical inference and the formalization of algorithmic processes in this context. Therefore, to address a broad class of problems involving inductive de-

scriptions, we propose a combined approach where a large language model transforms the inductive descriptions of a class of objects or functions into a mathematical formulation. Subsequently, a program is executed that implements the recursive algorithm derived from the FPAG-theorem. Essentially, this creates hybrid artificial intelligence, where large language models and their derivatives serve as intermediaries between natural and mathematical languages, while the algorithms themselves being executed through dedicated programs. The FPAG-theorem can serve as an effective benchmark for evaluating the quality of outputs generated by large language models.

To illustrate, one can provide ChatGPT with a set of initial polynomial functions, denoted as  $f_1, \dots, f_n$ , along with their corresponding generative term families,  $T_{f_1}, \dots, T_{f_n}$ . Then, one can pose the following question to the model: what is the value of the function at a given element  $w$ ? It is worth noting that even in relatively simple scenarios, large language models have been known to produce erroneous results.

#### 4.4. Trustworthy Artificial Intelligence

Trustworthy artificial intelligence [13,44,45] stands out as one of the most pertinent areas within digital technologies today. First of all, programs in trusted artificial intelligence systems should always stop and have polynomial computational complexity. We close this question by constructing p-complete languages with recursive constructions based on the FPAG-theorem. There are still many other open questions, including questions about logical inferences and explanations, but these questions can be considered partially closed since we have developed a theory of learning for intelligent systems based on logical-probabilistic methods of obtaining and generating knowledge. This learning theory is based on the task approach [13], where a task is defined only when there is a criterion for its solution. The correct formalization of tasks using logical formulas and finding solutions using recursive functions that satisfy the conditions of a p-computable FGNF-system gives a powerful incentive in the direction of explainable AI.

#### 4.5. Smart Cities and Intelligent Digital Twins

We want to consider the processes occurring in a smart city [15,46] through the prism of inductively determined constructs. For example, the number of residents in a city is defined as the sum of the number of residents in each of the city's districts. The number of residents in a district is the sum of the number of residents in each of its quarters and so on, being reduced to houses and then apartments. This process can be described recursively. Therefore, the FPAG-theorem can be applied to count the number of residents in a city, state, country, world.

Let us inductively codify and define the following notions: resident, house, quarter, district, and city:

$$\underline{resident} : resident\_info$$

where *resident\_info* is a JSON file with information about the resident.

$$\underline{house} : < house, house\_info, \underline{resident}_1, \dots, \underline{resident}_n >$$

$$\underline{quarter} : < quarter, quarter\_info, \underline{house}_1, \dots, \underline{house}_k >$$

$$\underline{district} : < district, district\_info, \underline{quarter}_1, \dots, \underline{quarter}_m >$$

$$\underline{city} : < city, city\_info, \underline{district}_1, \dots, \underline{district}_p >$$

Then, the following recursive function *numResidents* will count the number of residents in a house, block, district, and city, respectively:

```

function numResidents(w) {
  if !list(w) then return 1;
  s:= 0;
  for x in (w[2:]) {
    s := s + numResidents(x)
  }
  return s;
}

```

where  $w[2:]$  is a list that is obtained from a list  $w$  by removing the first two elements.

**Lemma 3.** *The computational complexity of the function  $numResidents(x)$  is polynomial.*

**Proof.** The proof follows immediately from the FPAG-theorem where the initial p-computable function  $numResidents$  is defined on non-list elements:

$$numResidents(\underline{resident}_i) = 1,$$

and where the generative family  $T_{numResidents}$  consists of a countable number of terms:

$$\{numResidents(x_3) + \dots + numResidents(x_n) \mid n \in N\}$$

The p-computable function  $\gamma$  has the following form:

$$\gamma(\langle w_1, \dots, w_k \rangle) = numResidents(x_3) + \dots + numResidents(x_k)$$

It is necessary to check conditions (2) and (3) for generative families  $T_{numResidents}$ . There exist constants  $C$  and  $p$  for condition (2) such that

$$t(\bar{t}_j^y(l_3, \dots, l_k)) = t(l_3 + \dots + l_k) \leq C \cdot |\langle l_3, \dots, l_k \rangle|^p$$

where  $l_3, \dots, l_k \in N$ .

Condition (3) is also true:

$$|\bar{t}_j^y(l_3, \dots, l_k)| = |l_3 + \dots + l_k| \leq |\langle l_3, \dots, l_k \rangle|$$

where  $l_3, \dots, l_k \in N$ .

Thus, we can construct the p-computable FGNF-system, and the FPAG-theorem can be applied.  $\square$

The method can also be used to calculate people’s well-being, passenger flows, road congestion; count daily contacts of one person with others; and assess the spread of various diseases such as COVID-19 [47]. It can be used to build various indices [48]: the index of the quality of education, the index of the quality of medicine, and the index of the quality of life. Moreover, it can be utilized for constructing the level of wealth distribution among different segments of the population and for building a digital model of a city and analyzing it to determine the satisfaction of various criteria put forward by society for smart cities, including using urban templates.

As for intelligent digital twins, it is also very important to use inductive concepts and recursive functions. It is necessary to obtain new knowledge based on a set of facts and precedents, which most often generalizes previous experience. The facts and processes themselves can fall under the inductive description, which allows using the PAG-theorem and FPAG-theorem. It is noteworthy to emphasize the rapid evolution of virtual cities, in which digital twins of administrative entities engage in interactions with the digital twins of physical objects. As has been previously demonstrated, it is not feasible to rely solely on language models in this context. In order to tackle logical and recursive problems,

it is imperative to integrate logical-probabilistic approaches and develop algorithms based on the FPAG-theorem.

As for p-complete programming languages, we conclude this series of papers with the FPAG-theorem and the construction of a programming methodology in Turing-complete languages.

## 5. Discussion

This paper left some questions unanswered regarding the reduction in constraints under certain conditions on the p-compatibility of the FGNF-system. Specifically, this concerns the requirements for the complexity  $t(t_j^{\bar{y}}(\bar{w}, \bar{l}))$  (2) and the length  $|t_j^{\bar{y}}(\bar{w}, \bar{l})|$  (3) of terms in generative families. Is it possible to relax these conditions and still maintain polynomial complexity for fixed point functions?

## 6. Conclusions

Recursion is one of the most powerful and expressive concepts in the realm of human thought. Through recursion, we can express a wide range of inductively generated entities and ideas. However, recursion is very dangerous in terms of computational complexity. To mitigate this issue, it becomes imperative to devise strategies that effectively curtail the exponential growth in complexity that is inherent in recursively constructed structures. In this work, we have identified such constraints. Through the use of the FPAG-theorem, we have provided precise conditions for defining recursive functions in such a way that they remain within the bounds of polynomial computational complexity, thereby ensuring the feasibility of their implementation.

The application of the FPAG-theorem is a topic that can gain significant attention in various fields, including trustworthy AI, digital twin technology, smart city development, blockchain systems, and smart contract implementation. Several lemmas were presented in this paper to illustrate the possible applications of the FPAG-theorem. However, there are still numerous areas where the full potential of this theorem remains largely untapped. What are fractal systems worth? Which systems have not yet been studied and which can also be analyzed using the FPAG-theorem?

As for p-complete programming languages, we conclude this series of papers with the FPAG-theorem and the construction of a programming methodology in Turing-complete languages. The FPAG-theorem provides a powerful expressive toolkit while maintaining polynomial computational complexity. Thus, it eliminates the halting problem that arises in Turing-complete languages and allows one to build reliable and trusted solutions. The Applications section contains a series of lemmas that demonstrate methods for implementing the FPAG-theorem in domains such as smart cities, Turing-complete programming languages, and smart contracts.

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

**Funding:** This work was supported by a grant for research centers, provided by the Analytical Center for the Government of the Russian Federation in accordance with the subsidy agreement (agreement identifier 000000D730324P540002) and the agreement with the Novosibirsk State University dated 27 December 2023 No. 70-2023-001318.

**Data Availability Statement:** The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

**Acknowledgments:** The authors would like to thank the entire research team at the Center for Artificial Intelligence at Novosibirsk State University for their discussions and support of our current research project.

**Conflicts of Interest:** The authors declare no conflicts of interest.

### Abbreviations

The following abbreviations are used in this manuscript:

MDPI	Multidisciplinary Digital Publishing Institute
DOAJ	Directory of open access journals
TLA	Three-letter acronym
LD	Linear dichroism

### References

- Metz, C. Google Is 2 Billion Lines of Code—And It’s All in One Place. 2015. Available online: <https://www.wired.com/2015/09/google-2-billion-lines-codeand-one-place/> (accessed on 2 September 2024).
- Visscher, B. Exploring Complexity in Software Systems-From an Irrational Model of Software Evolution to a Theory of Psychological Complexity and Cognitive Limitations Based on Empirical Evidence. Ph.D. Thesis, Department of Computer Science and Software Engineering, University of Portsmouth, Portsmouth, UK, 2005.
- McEnergy, S. How Much Computer Code Has Been Written? 2020. Available online: <https://medium.com/modern-stack/how-much-computer-code-has-been-written-c8c03100f459> (accessed on 2 September 2024).
- Hutson, M. AI Learns to Write Computer Code in ‘Stunning’ Advance. 2022. Available online: <https://www.science.org/content/article/ai-learns-write-computer-code-stunning-advance> (accessed on 15 September 2024).
- Charlesworth, A. Infinite Loops in Computer Programs. *Math. Mag.* **1979**, *52*, 284–291. [[CrossRef](#)]
- Carbin, M.; Misailovic, S.; Kling, M.; Rinard, M.C. Detecting and Escaping Infinite Loops with Jolt. In *Object-Oriented Programming. ECOOP 2011; Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6813. [[CrossRef](#)]
- Schoning, U. Algorithmics in Exponential Time. In *STACS 2005; Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3404. [[CrossRef](#)]
- Dell, H.; Husfeldt, T.; Marx, D.; Taslaman, N.; Vahlen, N. Exponential Time Complexity of the Permanent and the Tutte Polynomial. *ACM Trans. Algorithms (TALG)* **2014**, *10*, 1–32. [[CrossRef](#)]
- Goncharov, S.S.; Nechesov, A.V. Polynomial Analogue of Gandy’s Fixed Point Theorem. *Mathematics* **2021**, *9*, 2102. [[CrossRef](#)]
- Nechesov, A.V. Semantic Programming and Polynomially Computable Representations. *Sib. Adv. Math.* **2023**, *33*, 66–85. [[CrossRef](#)]
- Charlesworth, A. *Nary One Bit O’Magic How Computers Work*; Digital Equipment Corporation: Maynard, MA, USA, 1995.
- Goncharov, S.; Nechesov, A. Solution of the Problem  $P = L$ . *Mathematics* **2022**, *10*, 113. [[CrossRef](#)]
- Vityaev, E.E.; Goncharov, S.S.; Gumirov, V.S.; Mantsivoda, A.V.; Nechesov, A.V.; Sviridenko, D.I. Task approach: On the way to trusting artificial intelligence. In Proceedings of the World Congress: Systems Theory, Algebraic Biology, Artificial Intelligence: Mathematical Foundations and Applications, Novosibirsk, Russia, 26–30 June 2023. [[CrossRef](#)]
- Goncharov, S.; Nechesov, A. Axiomatization of Blockchain Theory. *Mathematics* **2023**, *11*, 2966.. [[CrossRef](#)]
- Haluskova, B. Digital Twin in Smart City. *Transp. Res. Procedia* **2023**, *74*, 1471–1478. [[CrossRef](#)]
- Yang, Z.; Chen, L.; Sun, Y.; Li, H. Visual Point Cloud Forecasting enables Scalable Autonomous Driving. *arXiv* **2023**, arXiv:2312.17655.
- Feng, K.; Li, C.; Ren, D.; Yuan, E.; Wang, G. On the Road to Portability: Compressing End-to-End Motion Planner for Autonomous Driving. *arXiv* **2024**, arXiv:2403.01238.
- Wei, Y.; Wang, Z.; Lu, Y.; Xu, C.; Liu, C.; Zhao, H.; Chen, S.; Wang, Y. Editable Scene Simulation for Autonomous Driving via Collaborative LLM-Agents. *arXiv* **2024**, arXiv:2402.05746.
- Rybczak, M.; Popowniak, N.; Lazarowska, A. A Survey of Machine Learning Approaches for Mobile Robot Control. *Robotics* **2024**, *13*, 12. [[CrossRef](#)]
- Badia Torres, J.; Perez Gracia, A.; Domenech-Mestres, C. Driving Strategies for Omnidirectional Mobile Robots with Offset Differential Wheels. *Robotics* **2024**, *13*, 19. [[CrossRef](#)]
- Andrew, H. *Alan Turing: The Enigma*; Burnett Books/Hutchinson: Cambridgeshire, UK; Simon & Schuster: New York, NY, USA, 1983. ISBN 0-671-49207-1.
- Teller, A. Turing completeness in the language of genetic programming with indexed memory. In Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Orlando, FL, USA, 27–29 June 1994; Volume 1, pp. 136–141. [[CrossRef](#)]
- Boyer, R.; Moore, J. *A Mechanical Proof of the Turing Completeness of Pure LISP*; Institute for Computer Science and Computer Applications, University of Texas at Austin: Austin, TX, USA, 1983. [[CrossRef](#)]
- Dolan, S. Mov Is Turing-Complete. 2013. Available online: <https://drwho.virtadpt.net/files/mov.pdf> (accessed on 5 September 2024).
- Knuth, D.E. Textbook Examples of Recursion. *arXiv* **1991**. [[CrossRef](#)]
- Shilov, N.V.; Danko, D. Teaching Efficient Recursive Programming and Recursion Elimination Using Olympiads and Contests Problems. In *Proceedings of the Workshop on Frontiers in Software Engineering Education (FISEE-2019)*, Lecture Notes in Computer Science; Springer International Publishing: Berlin/Heidelberg, Germany, 2020; Volume 12271, pp. 246–264. [[CrossRef](#)]



27. Shilov, N.V. Etude on Recursion Elimination. *Model. Anal. Inf. Syst.* **2018**, *25*, 549–560. [CrossRef]
28. Shilov, N.V. Study of Recursion Elimination for a Class of Semi-Interpreted Recursive Program Schemata. The 31st Nordic Workshop on Programming Theory, NWPT'19. Available online: <https://cs.ttu.ee/events/nwpt2019/abstracts/paper16.pdf> (accessed on 15 September 2024).
29. McCarthy 91 Function. Available online: [https://en.wikipedia.org/wiki/McCarthy\\_91\\_function](https://en.wikipedia.org/wiki/McCarthy_91_function) (accessed on 15 September 2024).
30. Berry, G. Bottom-up computation of recursive programs. *RAIRO—Inform. Theor. Appl. (Theor. Inform. Appl.)* **1976**, *10*, 47–82. [CrossRef]
31. Bird, R.S. Zippy Tabulations of Recursive Functions. In *Mathematics of Program Construction*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2008; Volume 5133. [CrossRef]
32. Goncharov, S.S.; Nechesov, A.V. Semantic programming for AI and Robotics. In Proceedings of the 2022 IEEE International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON), Yekaterinburg, Russia, 11–13 November 2022; pp. 810–815. [CrossRef]
33. Ershov, Y.L. *Definability and Computability*; Springer: New York, NY, USA, 1996. ISBN 978-0-306-11039-9.
34. Barwise, J. *Admissible Sets, and Structures: An Approaches to Definability Theory*; Springer: Berlin/Heidelberg, Germany, 1975; ISBN 978-0-387-074511.
35. Python-Loop Lists. Available online: [https://www.w3schools.com/python/python\\_lists\\_loop.asp](https://www.w3schools.com/python/python_lists_loop.asp) (accessed on 15 September 2024).
36. Introduction to Smart Contracts. Available online: <https://ethereum.org/en/developers/docs/smart-contracts/> (accessed on 15 September 2024).
37. Paul, S.; Kumar, V.; Jha, P. Chapter 9-Artificial neural network and its applications: Unraveling the efficiency for hydrogen production. *Applications of Artificial Intelligence in Process Systems Engineering*; Elsevier: Amsterdam, The Netherlands, 2021; pp. 187–206. [CrossRef]
38. Russel, S.; Norving, P. *Artificial Intelligence: A Modern Approach (Pearson Series in Artificial Intelligence)*, 4th ed.; Pearson: London, UK, 2020; p. 1115, ISBN 978-0134610993.
39. Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F.L.; Almeida, D.; Altschmidt, J.; Altman, S.; Anadkat, S.; et al. GPT-4 Technical Report. *arXiv* **2023**, arXiv:2303.08774.
40. Liu, Y.; Han, T.; Ma, S.; Zhang, J.; Yang, Y.; Tian, J.; He, H.; Li, A.; He, M.; Liu, Z.; et al. Summary of ChatGPT-Related Research and Perspective Towards the Future of Large Language Models. *arXiv* **2023**, arXiv:2304.01852. [CrossRef]
41. Awesome-LLM. Available online: <https://github.com/Hannibal046/Awesome-LLM> (accessed on 15 September 2024).
42. Pang, S.; Nol, E.; Heng, K. ChatGPT-4o for English language teaching and learning: Features, applications, and future prospects. *Cambodian J. Educ. Res.* **2024**, *4*, 35–56. [CrossRef]
43. Nezhurina, M.; Cipolina-Kun, L.; Cherti, M.; Jitsev, J. Alice in Wonderland: Simple Tasks Showing Complete Reasoning Breakdown in State-Of-the-Art Large Language Models. *arXiv* **2024**, arXiv:2406.02061.
44. Gillis, R.; Laux, J.; Mittelstadt, B. Trust and Trustworthiness in Artificial Intelligence. In *Handbook on Artificial Intelligence and Public Policy*; Paul, R., Carmel, E., Cobbe, J., Eds.; Edward Elgar: Cheltenham, UK, 2024; ISBN 978-1803922164.
45. Li, B.; Qi, P.; Liu, B.; Di, S.; Liu, J.; Pei, J.; Yi, J.; Zhou, B. Trustworthy AI: From Principles to Practices. *arXiv* **2021**, arXiv:2110.01167. [CrossRef]
46. Mohanty, S.P.; Choppali, U.; Kougiannos, E. Everything you wanted to know about smart cities: The Internet of things is the backbone. *IEEE Consum. Electron. Mag.* **2016**, *5*, 60–70. [CrossRef]
47. Megahed, N.; Abdel-Kader, R. Smart Cities after COVID-19: Building a conceptual framework through a multidisciplinary perspective. *Sci. Afr.* **2022**, *17*, e01374. [CrossRef] [PubMed]
48. Bosch, P.; Jongeneel, S.; Neumann, H.-M.; Iglar, B.; Huovila, A.; Airaksinen, M.; Seppa, I. *Recommendations for a Smart City Index*; CITYkeys; European Commission: Brussels, Belgium, 2016. [CrossRef]

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.