mathematics

MDPI

Article

Parallel Communicating Finite Automata: Productiveness and Succinctness

Jingnan Xie [1,*], Ching-Sheng Lin [2] and Harry B. Hunt III [3]

[1] Computer Science, Millersville University of Pennsylvania, 40 Dilworth Rd, Millersville, PA 17551, USA
[2] Master Program of Digital Innovation, Tunghai University, Taichung City 40704, Taiwan; cslin612@thu.edu.tw
[3] Computer Science, University at Albany, State University of New York, Albany, NY 12222, USA; huntharryiii@gmail.com
* Correspondence: jingnan.xie@millersville.edu

Abstract: Parallel Communicating Finite Automata (PCFA) extend classical finite automata by enabling multiple automata to operate in parallel and communicate upon request, capturing essential aspects of parallel and distributed computation. This model is relevant for studying complex systems such as computer networks and multi-agent environments. In this paper, we explore two key aspects of PCFA: their undecidability and their descriptional complexity. We first show that deterministic PCFA of degree 2 ($DPCFA(2)$) can accept a set of valid computations of a deterministic Turing machine, leading to the undecidability of restricted versions of emptiness and universality problems. Additionally, we employ the concept of productiveness (a stronger form of non-recursive enumerability) to demonstrate that these problems are not only undecidable but also unprovable. Second, we investigate the descriptional complexity of PCFA and establish non-recursive trade-offs between different PCFA models and many classes of language descriptors, such as DFAs and subclasses of regular expressions, offering new insights into their computational and structural properties.

Keywords: parallel communicating finite automata; undecidability; productiveness; 1DFA(k); descriptional complexity

MSC: 03D05; 03D10; 03D15; 03D25

check for updates

Academic Editor: Adolfo Ballester-Bolinches

Received: 4 March 2025
Revised: 5 April 2025
Accepted: 9 April 2025
Published: 11 April 2025

Citation: Xie, J.; Lin, C.-S.; Hunt, H.B., III. Parallel Communicating Finite Automata: Productiveness and Succinctness. *Mathematics* **2025**, *13*, 1265. https://doi.org/10.3390/math13081265

Copyright: © 2025 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

Parallel Communicating Finite Automata (PCFA) were introduced by Martín-Vide, Mateescu, and Mitrana in 2002 [1] as a computational model consisting of multiple finite automata operating in parallel and communicating upon request. This extension of traditional finite automata enables parallel processing and coordinated computation, offering enhanced capabilities for modeling concurrent systems. Unlike classical finite automata, which function as isolated units, PCFA engage in explicit communication, enabling them to capture the complexities of distributed and parallel processes. This characteristic makes PCFA particularly effective for modeling computer networks and multi-agent systems. The study of PCFA is driven by their ability to reflect real-world concurrency and provide insights into the behavior of complex systems. Furthermore, exploring the decidability and undecidability of problems associated with PCFA is essential for understanding the boundaries of computation and the intrinsic challenges of parallel computational models. Further developments in this area are examined in [2–7].

In [5], Bordihn et al. showed that certain types of PCFA can accept a set of valid computations of one-way cellular automata, which is a parallel computational model (for example, see [8–11]). Consequently, the emptiness, universality, inclusion, equivalence, finiteness, and infiniteness problems are undecidable for these PCFA models. They also mentioned that whether these models can accept the valid computations of a Turing machine remains an open problem. In this paper, we show that some fundamental PCFA models, such as deterministic PCFA of degree 2 ($DPCFA(2)$), can accept a set of valid computations of a deterministic Turing machine. As a result, a restricted version of the emptiness problem is undecidable for $DPCFA(2)$, as follows:

Testing equivalence to $\emptyset$ for languages whose cardinalities are at most one (denoted by "$= \emptyset \mid_{|L| \leq 1}$").

Since DPCFA are closed under complementation (proved in [1]), a restricted version of the universality problem is also undecidable for $DPCFA(2)$, as follows:

Testing equivalence to $\{0,1\}^*$ for languages whose complements' cardinalities are at most one (denoted by "$= \{0,1\}^* \mid_{|L^c| \leq 1}$").

The instances of these restricted predicates have very important semantic properties: they are the simplest regular sets. These restrictions make the predicates more widely applicable: for example, they directly apply to promise problems, predicates on regular sets, and descriptional complexity of language descriptors. Furthermore, we employ a stronger form of non-recursive enumerability known as productiveness (as discussed in [12–14]). The undecidability results in this paper are, in fact, productiveness results, demonstrating that these problems are not only undecidable but also unprovable.

The second objective of this paper is to investigate the descriptional complexity of PCFA. In formal language theory, descriptional complexity concerns how succinctly a descriptor can generate a language compared with other descriptors generating the same language. In [15], Hartmanis demonstrated that no recursive trade-off exists between pushdown automata (PDA) and deterministic pushdown automata (DPDA). Further related research can be found in [16–19]. In this paper, we evaluate the trade-offs between various PCFA models and a range of other classes of language descriptors, including DFAs, subclasses of regular expressions, and multiple L systems (as elaborated in [20]).

This paper is organized as follows:

In Section 2, we review the definitions of various types of PCFA and introduce important preliminary definitions and notations. This section also introduces the definition and significance of productiveness to enhance its self-contained nature.

In Section 3, we establish productiveness results for restricted versions of the emptiness and universality problems for $DPCFA(2)$ by demonstrating that the sets of valid and invalid computations of a fixed deterministic Turing machine for a given input can be accepted by some DPCFA. This analysis also yields additional undecidability results.

In Section 4, we examine the descriptional complexity of PCFA, establishing non-recursive trade-offs between several PCFA models and other classes of language descriptors.

2. Definitions and Preliminary Results

In this section, we review the definitions of PCFA and one-way multi-head DFA from [1] and [21,22], respectively. Several preliminary definitions and notations are also explained. The reader is referred to [23] for all unexplained notations and terminologies in language theory.

We use λ to denote the empty string, $\emptyset$ to denote the empty set, and $\mathbb{N}$ to denote the set of natural numbers. For two sets A and B, if A is many-one reducible to B, we write $A \leqslant_m B$. For a finite set A, we denote by $\mathrm{card}(A)$ the cardinality of A.

Let $\mathcal{D}$ be a class of language descriptors that describe languages over Σ. In this paper, we only consider finite Σ. Then, for any $d \in \mathcal{D}$, $\mathcal{L}(d) = \{w \in \Sigma^* \mid w \text{ is described by } d\}$ and $\mathcal{L}(\mathcal{D}) = \{L \subseteq \Sigma^* \mid \exists d \in \mathcal{D} \text{ such that } L = \mathcal{L}(d)\}$. For any $d \in \mathcal{D}$, let $|d|$ denote the size of d. For example, the size of a DFA is the number of states of the DFA. The size of a context-free grammar is the number of symbols of all its productions. For example, consider the context-free grammar $d = (V, \Sigma, R, S)$ that accepts $\{0,1\}^*$ where

- $V = \{s_1\}$ is the set of variables;
- $\Sigma = \{0,1\}$ is the set of terminals;
- $R = \{(s_1, 0s_1), (s_1, 1s_1), (s_1, \lambda)\}$ is the set of production rules; and
- $S = s_1$ denotes the start symbol.

Thus, $d = (\{s_1\}, \{0,1\}, \{(s_1, 0s_1), (s_1, 1s_1), (s_1, \lambda)\}, s_1)$. The size of d is 8 (denoted by $|d| = 8$).

Comparing two classes of language descriptors $\mathcal{D}_1$ and $\mathcal{D}_2$, we assume that $\mathcal{L}(\mathcal{D}_1) \cap \mathcal{L}(\mathcal{D}_2)$ is not finite. We say that a function $f : \mathbb{N} \mapsto \mathbb{N}$ where $f(n) \geq n$ is an *upper bound for the trade-off between $\mathcal{D}_1$ and $\mathcal{D}_2$* when transforming from a minimal descriptor in $\mathcal{D}_1$ for an arbitrary language to an equivalent minimal descriptor in $\mathcal{D}_2$, if, for all $L \in \mathcal{L}(\mathcal{D}_1) \cap \mathcal{L}(\mathcal{D}_2)$, the following holds:

$$Min\{|d| \mid d \in \mathcal{D}_2, \mathcal{L}(d) = L\} \leq f(Min\{|d| \mid d \in \mathcal{D}_1, \mathcal{L}(d) = L\}).$$

If no recursive function is an upper bound for the trade-off between $\mathcal{D}_1$ and $\mathcal{D}_2$, we say that *the trade-off between $\mathcal{D}_1$ and $\mathcal{D}_2$ is non-recursive* .

Definition 1. *A parallel communicating finite automata system of degree n is a construct*

$$A = (V, A_1, A_2, \ldots, A_n, K),$$

where

- *V is the input alphabet;*
- *$A_i = (Q_i, V, f_i, q_i, F_i), 1 \leq i \leq n$ are finite automata with the set of states Q_i, the initial state $q_i \in Q_i$, the set of final states $F_i \subseteq Q_i$, and the transition mapping f_i defined as*

$$f_i : Q_i \times (V \cup \{\lambda\}) \to 2^{Q_i}.$$

Note that Q_i are not necessarily disjoint sets.

- *$K = \{K_1, K_2, \ldots, K_n\} \subseteq \bigcup_{i=1}^{n} Q_i$ is the set of query states.*

The automata $A_1, A_2, \ldots, A_n$ are called the components of the system A. If there exists exactly one $1 \leq i \leq n$ such that $K \subseteq Q_i$, then the system is said to be centralized, with component i as the master. For simplicity, when a system is centralized, the first component is considered the master.

The system is deterministic if the following conditions hold for all $1 \leq i \leq n$, as follows:

1. *$Card(f_i(s, a)) \leq 1$ for all $s \in Q_i$ and $a \in V \cup \{\lambda\}$;*
2. *If $card(f_i(s, \lambda)) \neq 0$ for some $s \in Q_i$, then $card(f_i(s, a)) = 0$ for all $a \in V$.*

Definition 2. *A configuration of a parallel communicating finite automata system A is a $2n$-tuple*

$$(s_1, x_1, s_2, x_2, \ldots, s_n, x_n),$$

where

- *s_i is the current state of component i; and*
- *x_i is the remaining part of the input word not yet read by component i for $1 \leq i \leq n$.*

We define two binary relations on the set of all configurations of A as follows:

$$(s_1, x_1, s_2, x_2, \ldots, s_n, x_n) \vdash (p_1, y_1, p_2, y_2, \ldots, p_n, y_n)$$

if one of the following conditions holds:

1. $K \cap \{s_1, s_2, \ldots, s_n\} = \emptyset$ *and*

$$x_i = a_i y_i, \quad a_i \in V \cup \{\lambda\}, \quad p_i \in f_i(s_i, a_i), \quad 1 \leq i \leq n.$$

2. *For all* $1 \leq i \leq n$ *such that* $s_i = K_{j_i}$ *and* $s_{j_i} \notin K$, *set*

$$p_i = s_{j_i}, \quad p_{j_i} = s_{j_i}, \quad p_r = s_r \text{ for all other } 1 \leq r \leq n, \quad y_t = x_t \text{ for all } 1 \leq t \leq n.$$

$$(s_1, x_1, s_2, x_2, \ldots, s_n, x_n) \vdash_r (p_1, y_1, p_2, y_2, \ldots, p_n, y_n)$$

if one of the following conditions holds:

1. $K \cap \{s_1, s_2, \ldots, s_n\} = \emptyset$ *and*

$$x_i = a_i y_i, \quad a_i \in V \cup \{\lambda\}, \quad p_i \in f_i(s_i, a_i), \quad 1 \leq i \leq n.$$

2. *For all* $1 \leq i \leq n$ *such that* $s_i = K_{j_i}$ *and* $s_{j_i} \notin K$, *set*

$$p_i = s_{j_i}, \quad p_{j_i} = q_{j_i}, \quad p_r = s_r \text{ for all other } 1 \leq r \leq n, \quad y_t = x_t \text{ for all } 1 \leq t \leq n.$$

A parallel communicating automata system in which all moves are based on the relation $\vdash_r$ *is said to be returning.*

The language accepted by a PCFA A consists of all strings $x \in V^*$ *such that the system starts in an initial configuration, as follows:*

$$(q_1, x, q_2, x, \ldots, q_n, x)$$

and reaches a final configuration of the form

$$(s_1, \lambda, s_2, \lambda, \ldots, s_n, \lambda),$$

with $s_i \in F_i$. *Formally,*

$$\mathcal{L}(A) = \{x \in V^* \mid (q_1, x, q_2, x, \ldots, q_n, x) \vdash^* (s_1, \lambda, s_2, \lambda, \ldots, s_n, \lambda), s_i \in F_i, 1 \leq i \leq n\}.$$

Similarly, the language accepted by a returning PCFA A is

$$\mathcal{L}_r(A) = \{x \in V^* \mid (q_1, x, q_2, x, \ldots, q_n, x) \vdash_r^* (s_1, \lambda, s_2, \lambda, \ldots, s_n, \lambda), s_i \in F_i, 1 \leq i \leq n\}.$$

We introduce the following notation:

- rcpcfa(n)—a returning centralized parallel communicating finite automata system of degree n;
- rpcfa(n)—a returning parallel communicating finite automata system of degree n;
- cpcfa(n)—a centralized parallel communicating finite automata system of degree n;
- pcfa(n)—a parallel communicating finite automata system of degree n.

The prefix d is added to denote deterministic variants. If $x(n)$ represents a type of automata system, then $X(n)$ is the class of such systems. For example, RCPCFA(n) is the class of all rcpcfa(n) automata systems.

Example 1 ([1]). *Consider the centralized parallel communicating finite automata system, as follows:*

$$A = (\{a, b, c\}, A_1, A_2, \{K_1, K_2\}),$$

where A_1 and A_2 are two deterministic finite automata with λ-moves, whose transition mappings are given below:

$$f_1(q_1, \lambda) = K_2, \quad f_1(s_1, a) = K_2, \quad f_1(q_2, \lambda) = K_2, \quad f_1(s_2, b) = K_2,$$
$$f_1(s_f, c) = q_f, \quad f_1(q_f, c) = q_f.$$

$$f_2(q_2, a) = q_2, \quad f_2(q_2, b) = s_1, \quad f_2(s_1, b) = u_1, \quad f_2(u_1, c) = s_2,$$
$$f_2(u_2, c) = s_2, \quad f_2(s_2, \lambda) = s_f, \quad f_2(s_f, \lambda) = s_f.$$

By taking the sets of final states as $F_1 = \{q_f\}$ and $F_2 = \{s_f\}$, we obtain the language

$$\mathcal{L}(A) = \{a^n b^n c^n \mid n \geq 1\},$$

which is a non-context-free language.

Lemma 1 ([1]).

1. $\mathcal{L}(RCPCFA(n)) \subseteq \mathcal{L}(RPCFA(n))$ *and* $\mathcal{L}(CPCFA(n)) \subseteq \mathcal{L}(PCFA(n))$, *for all* $n \geq 1$.
2. $\mathcal{L}(X(n)) \subseteq \mathcal{L}(X(n+1))$ *for all* $X \in \{RCPCFA, RPCFA, CPCFA, PCFA\}$.
3. *All of the above relations hold for deterministic variants as well.*

Lemma 2 ([3]). $\mathcal{L}(DRPCFA(n)) = \mathcal{L}(DPCFA(n))$.

Definition 3. *A deterministic one-way k-head finite automaton (1DFA(k)) is a system, as follows:*

$$M = \langle S, A, k, \delta, \mathsf{L}, s_0, F \rangle$$

where

1. *S is the finite set of internal states;*
2. *A is the finite set of input symbols (the input alphabet)*
3. *$k \geq 1$ is the number of heads;*
4. *L is the left endmarker, with $\mathsf{L} \notin A$;*
5. *$s_0 \in S$ is the initial state;*
6. *$F \subseteq S$ is the set of accepting states;*
7. *$\delta : S \times (A \cup \{\mathsf{L}\})^k \to S \times \{0, 1\}^k$ is the partial transition function, where for a state $s \in S$ and symbols $(x_1, x_2, \ldots, x_k) \in (A \cup \{\mathsf{L}\})^k$ scanned by the k heads,*
   - *(a) $\delta(s, (x_1, x_2, \ldots, x_k)) = (s', (d_1, d_2, \ldots, d_k))$;*
   - *(b) $s' \in S$ is the next state;*
   - *(c) $d_i \in \{0, 1\}$ for each i, where 1 moves head i one square to the right, and 0 keeps head i in place.*

A configuration of M at time $t \geq 0$ is a triple, as follows:

$$c_t = (w, s, p)$$

where

1. $w \in A^*$ *is the input word;*
2. $s \in S$ *is the current state; and*
3. $p = (p_1, p_2, \ldots, p_k) \in \{0, 1, \ldots, |w|\}^k$ *represents the positions of the k heads on the tape, where*
   (a) *If* $p_i = 0$, *head i is scanning the left endmarker* L;
   (b) *If* $1 \leq p_i \leq |w|$, *head i is scanning the* p_i*-th letter of the input word w.*

The initial configuration for an input word $w = a_1 a_2 \ldots a_n$ *is*

$$(w, s_0, (1, 1, \ldots, 1))$$

where each head starts at the leftmost position of w (i.e., scanning a_1*).*

The machine transitions from one configuration to the next by applying the transition function. For a configuration $(w, s, (p_1, p_2, \ldots, p_k))$, *it moves to*

$$(w, s', (p_1 + d_1, p_2 + d_2, \ldots, p_k + d_k))$$

if and only if $(s', (d_1, d_2, \ldots, d_k)) = \delta(s, (x_1, x_2, \ldots, x_k))$, *where*

- $x_i = \mathsf{L}$ *if* $p_i = 0$;
- $x_i = a_{p_i}$ *if* $1 \leq p_i \leq |w|$;
- $d_i \in \{0, 1\}$ *for each i, and* $p_i + d_i \leq |w|$ *(heads cannot move beyond the input).*

The computation halts when no further transition is defined or an accepting state is reached.

The language accepted by 1DFA(k) is the set of all input words $w \in A^*$ *such that there exists a computation sequence from the initial configuration to a configuration with an accepting state, as follows:*

$$\mathcal{L}(M) = \{w \in A^* \mid (w, s_0, (1, 1, \ldots, 1)) \xrightarrow{*} (w, s, (p_1, p_2, \ldots, p_k)), s \in F\}.$$

We use $1DFA(k)s$ *to denote the class of one-way k-head deterministic finite automata.*

A key theorem from [1], demonstrating that $\mathcal{L}(\text{DPCFA}(n)) = \mathcal{L}(1\text{DFA}(n)s)$, plays a significant role in our research. In conjunction with Lemma 2, we present a stronger version of this theorem.

Theorem 1. $\mathcal{L}(DRPCFA(n)) = \mathcal{L}(DPCFA(n)) = \mathcal{L}(1DFA(n)s)$.

To make the paper more self-contained, we explain the concept and significance of productiveness as discussed in [14], since most of the undecidability results in this paper are productiveness results.

Productive sets and their properties are a standard topic in mathematical logic/recursion theory textbooks such as [24,25]. Productiveness is a recursion-theoretic abstraction of what causes Gödel's first incompleteness theorem to hold. The next definition recalls the definition of a productive set on $\mathbb{N}$, as developed in [24].

Definition 4. *Let W be an effective Gödel numbering of the recursively enumerable sets. A set A of natural numbers is called* productive *if there exists a total recursive function f so that, for all* $i \in \mathbb{N}$, *if* $W_i \subseteq A$, *then* $f(i) \in A - W_i$. *The function f is called the* productive function *for A.*

From this definition, we can see that no productive set is recursively enumerable. It is well known that the set of all provable sentences in an effective axiomatic system is always a recursively enumerable set. Therefore, for any effective axiomatic system F, if a set A

of Gödel numbers of true sentences in F is productive, then there is at least one element in A that is true but cannot be proven in F. Moreover, there is an effective procedure to produce such an element. Hence, productiveness is particularly significant as it highlights a fundamental distinction between computer science and many other areas of mathematics. In computer science, the focus is often on decidability, which concerns whether a problem can be algorithmically determined to have a solution. In contrast, fields such as formal logic and computability theory emphasize provability, which refers to whether a true statement can be formally derived from a set of axioms.

Let W be an effective Gödel numbering of the recursively enumerable sets. $\mathbf{K}$ denotes the set $\{i \in \mathbb{N} \mid i \in W_i\}$. $\overline{\mathbf{K}}$ denotes the set $\{i \in \mathbb{N} \mid i \notin W_i\}$. Two well-known facts of productive sets that are necessary for the research developed here are as follows:

Proposition 1 ([24])**.**

1. $\overline{\boldsymbol{K}}$ *is productive.*
2. *For all* $A \subseteq \mathbb{N}$, A *is productive if and only if* $\overline{\boldsymbol{K}} \leqslant_m A$.

The following proposition is proven in [12] and is used to prove productiveness results. It also shows in which way productiveness is stronger than non-recursive enumerability; i.e., every productive set A has an infinite recursively enumerable subset, and for any soundproof procedure P, one can effectively construct an element that is in A, but not provable in P.

Proposition 2 ([12])**.** *Let* $A \subseteq \Sigma^*$, $B \subseteq \Delta^*$, *and* $A \leqslant_m B$. *Then, the following hold:*

1. *If A is productive, then so is B.*
2. *If A is productive, then there exists a total recursive function* $\Psi : \Sigma^* \to \Sigma^*$, *called a* productive function for A, *such that for all* $x \in \Sigma^*$,

   $\mathcal{L}(M_x) \subseteq A \Rightarrow \Psi(x) \in A - \mathcal{L}(M_x)$, *where* $\{M_x \mid x \in \Sigma^*\}$ *is some Gödel numbering of Turing machines over alphabet* Σ.
3. *If A is productive, then A is not recursively enumerable (RE). However, A does have an infinite RE subset.*

3. Productiveness of PCFA

To establish the results presented in this paper, we begin by examining the sets of valid and invalid computations of Turing machines. Unlike the definitions provided in [23,26], we adopt the definitions of valid and invalid computations of a fixed deterministic Turing machine on a given input, as outlined in [12,27]. This refined definition enables us to investigate the complexity/undecidability of the following two restricted language predicates:

Testing equivalence to $\{0,1\}^*$ for languages whose complements' cardinalities are less than or equal to one (denoted by "$= \{0,1\}^* \mid_{|L^c| \leq 1}$");

and

Testing equivalence to $\emptyset$ for languages whose cardinalities are less than or equal to one (denoted by "$= \emptyset \mid_{|L| \leq 1}$").

The instances of these restricted predicates have very important semantic properties: they are the simplest regular sets. These restrictions make the predicates more widely applicable; for example, they directly apply to promise problems, predicates on regular sets, and descriptional complexity of language descriptors.

Throughout this section, $M = (Q, \Sigma, T, \delta, q_0, B, F)$ is a single-tape deterministic Turing machine where

1. Q is M's nonempty finite set of states;
2. $q_0 \in Q$ is M's unique start state;

3. $F \subseteq Q$ is M's set of accepting states. Each one in F is final;
4. M's input alphabet is Σ and T is M's tape alphabet where $\Sigma \subseteq T$;
5. $B \in T$ is the blank symbol;
6. $\delta : ((Q - F) \times T) \mapsto (Q \times T \times \{L, R\})$ is the transition function where L is the left shift and R is the right shift; and
7. $\Delta_M = T \cup (Q \times T) \cup \{\#\}$ where the sets T, $(Q \times T)$, and $\{\#\}$ are pairwise disjoint. $\Delta'_M = \Delta_M - \{\#\}$.

Definition 5. *Let M be any fixed deterministic Turing machine. For all $w \in \Sigma^+$, letting $w = w_1w_2w_3...w_k$ where $w_j \in \Sigma$ $(1 \leq j \leq k)$, the set of valid computations of M on w denoted by $VALCM(w)$ is the set of strings of the form $\#id_0\#id_1\#id_2 \cdots \#id_n\#$ such that*

1. *Each id_i $(1 \leq i \leq n)$ is an ID (the definition of an* ID *can be seen in [23]) of M;*
2. *$id_0 = (q_0, w_1)w_2w_3...w_k$ is the initial ID of M on w;*
3. *id_n is a final ID; and*
4. *$id_i \vdash_M id_{i+1}$ ($\vdash_M$ represents a move of M; the definition of $id_i \vdash_M id_{i+1}$ can be seen in [23]) for $0 \leq i < n$.*

The set of invalid computations of M on w denoted by $INVALCM(w)$ is the complement of $VALCM(w)$ with respect to Δ_M^*.

The following theorem demonstrates that $DPCFA(2)$ and $DRPCFA(2)$, two of the simplest PCFA, can accept the sets $VALCM(w)$ and $INVALCM(w)$. Therefore, even for a subset D_1 of $DPCFA(2)$ (or $DRPCFA(2)$), where each element in D_1 generates either $\{0,1\}^*$ or $\{0,1\}^* - \{w\}$ $(w \in \{0,1\}^*)$, the predicate "$= \{0,1\}^*$" is already productive. Similarly, for a subset D_2 of $DPCFA(2)$ (or $DRPCFA(2)$), where each element in D_2 generates either $\emptyset$ or $\{w\}$ $(w \in \{0,1\}^*)$, the predicate "$= \emptyset$" is also productive.

Theorem 2. *There exists a subset D_1 of $DPCFA(2)$ (or $DRPCFA(2)$, $1DFA(2)$s) such that*

1. *$\forall d \in D_1, \mathcal{L}(d) \subseteq \{0,1\}^*$, and $|\{0,1\}^* - \mathcal{L}(d)| \leq 1$; and*
2. *$\overline{\boldsymbol{K}} \leqslant_m \{d \in D_1 \mid \mathcal{L}(d) = \{0,1\}^*\}$.*

Similarly, there exists a subset D_2 of $DPCFA(2)$ (or $DRPCFA(2)$, $1DFA(2)$s) such that

1. *$\forall d \in D_2, |\mathcal{L}(d)| \leq 1$; and*
2. *$\overline{\boldsymbol{K}} \leqslant_m \{d \in D_2 \mid \mathcal{L}(d) = \emptyset\}$.*

Proof. In this proof, we first show that $VALCM(w)$ can be accepted by $1DFA(2)$. Hence, by Theorem 1, the result also holds for $DPCFA(2)$ and $DRPCFA(2)$.

We write $a \vdash_M bc$ where $a, b, c \in \Delta'_M$ if and only if $a \in (Q \times T)$ is the rightmost letter of an ID, $\delta(a) = (q_i, b, R)$ and $c = (q_i, B)$.

We write $ab \vdash_M cd$ where $a, b, c, d \in \Delta'_M$ if and only if a is the leftmost letter of an ID and

1. If $a \in (Q \times T)$ and $\delta(a) = (q_i, t, L)$, then $c = (q_i, B), d = t$;
2. If $a \in (Q \times T)$ and $\delta(a) = (q_i, t, R)$, then $c = t, d = (q_i, b)$;
3. If $a, b \notin (Q \times T)$, then $c = a, d = b$.

Alternatively, b is the rightmost letter of an ID and

1. If $b \in (Q \times T)$ and $\delta(b) = (q_i, t, L)$, then $c = (q_i, a), d = t$;
2. If $a, b \notin (Q \times T)$, then $c = a, d = b$.

We write $abc \vdash_M def$ where $a, b, c, d, e, f \in \Delta'_M$ if and only if abc is an infix of an ID and

1. If $a, b, c \notin (Q \times T)$, then $d = a, e = b, f = c$;
2. If $b \in (Q \times T)$ and $\delta(b) = (q_i, t, L)$, then $d = (q_i, a), e = t, f = c$;
3. If $b \in (Q \times T)$ and $\delta(b) = (q_i, t, R)$, then $d = a, e = t, f = (q_i, c)$.

Intuitively, the notation $abc \vdash_M def$ where $b \in (Q \times T)$ means that three consecutive letters of an ID id_i determine the corresponding three consecutive letters of the next ID id_{i+1}. If $id_i \vdash_M id_{i+1}$, since $b \in (Q \times T)$, the only letters that change between id_i and id_{i+1} are abc and def. Hence, $id_i \# id_{i+1}$ can be represented as

$wabcv\#wdefv$ where $w, v \in \Delta_M'^*$ and $abc \vdash_M def$.

The notations $a \vdash_M bc$ and $ab \vdash_M cd$ are used to handle the boundary cases like a symbol in $(Q \times T)$ is at the beginning or the end of an ID. For a deterministic Turing machine M with a given input, we can design a $1DFA(2)$ that utilizes one head to inspect each triplet of consecutive letters in id_i and a second head to confirm that the corresponding three letters in id_{i+1} consistently match (i.e., $abc \vdash_M def$ holds for all three consecutive letters in id_i). After verifying each triplet, the two heads advance to the next pair of consecutive IDs, ensuring that the process repeats systematically. This enables $1DFA(2)$ to determine whether $id_i \vdash_M id_{i+1}$ holds for all pairs of consecutive IDs.

Since M is a deterministic Turing machine, if M halts, there is only one valid computation in $VALCM(w)$; otherwise, $VALCM(w) = \emptyset$. The other part of the theorem follows from the fact that the class of $1DFA(2)$ is closed under complement (mentioned in [28,29]). Therefore, we can construct $1DFA(2)$ to accept $INVALCM(w)$. The rest of the proof is completed by effectively encoding $VACLM(w)$ and $INVALCM(w)$ into a binary alphabet. □

Theorem 2 resolves an open problem from [5] by establishing that $dpcfa(2)$ can accept the set of valid computations of a deterministic Turing machine. As a result, the predicates "$= \{0,1\}^* \ |_{|L^c| \leq 1}$" and "$= \emptyset \ |_{|L| \leq 1}$" are productive for $DPCFA(2)$. Echoing the logic of [5], this implies that the inclusion, equivalence, finiteness, and infiniteness problems are undecidable for $DPCFA(2)$. Moreover, our findings yield stronger productiveness results and also apply to promise problems. For example, for $DPCFA(2)$, given a promise that every language is finite, the emptiness problem is still productive; given a promise that every language must be co-finite, the universality problem is still productive.

4. Descriptional Complexity of PCFA

Descriptional complexity can be investigated using the properties of the predicates "$= \emptyset|_{|L| \leq 1}$" and "$= \{0,1\}^*|_{|L^c| \leq 1}$". For example, we show that there is no recursive trade-off between $DPCFA(2)$ and DFAs. Intuitively, this is because both $\emptyset$ and $\{w\}$ can be accepted by a DFA, and the size of any DFA accepting $\{w\}$ must be greater than or equal to $|w|$. If the trade-off between $DPCFA(2)$ and DFAs is recursive, then $|w|$ is recursively bounded. Since w from the proof of Theorem 2 is the encoded valid computation of a Turing machine, this implies that the halting problem is recursive, which is a contradiction.

Moreover, it may be practically more relevant to ask the trade-off between $DPCFA(2)$ and a class of language descriptors accepting a particular subset of regular languages. The special properties of the predicates "$= \emptyset|_{|L| \leq 1}$" and "$= \{0,1\}^*|_{|L^c| \leq 1}$" enable us to study such trade-offs since the languages $\{0,1\}^*$ and $\{0,1\}^* - \{w\}$ are both co-finite, the languages $\emptyset$ and $\{w\}$ are both finite, and they are the simplest regular languages.

The following theorem establishes non-recursive trade-offs between $DPCFA(2)$ and many other classes of language descriptors.

Theorem 3.

1. *Let $\mathcal{D}$ be any class of language descriptors over alphabet $\{0,1\}$ such that*
   - *(a) For any $w \in \{0,1\}^+$, $L_w = \{0,1\}^* - \{w\} \in \mathcal{L}(\mathcal{D})$; and*
   - *(b) There exists a strictly increasing recursive function $f : \mathbb{N} \mapsto \mathbb{N}$ such that, for any $d \in \mathcal{D}$ specifying L_w, $f(|d|) > |w|$.*

Then, there is no recursive trade-off between $DPCFA(2)$ *(or* $DRPCFA(2)$, $1DFA(2)s$*) and* $\mathcal{D}$.

2. *Let* $\mathcal{D}$ *be any class of language descriptors over alphabet* $\{0,1\}$ *such that*
   (a) *For any* $w \in \{0,1\}^+$, $L'_w = \{w\} \in \mathcal{L}(\mathcal{D})$; *and*
   (b) *There exists a strictly increasing recursive function* $f : \mathbb{N} \mapsto \mathbb{N}$ *such that, for any* $d \in \mathcal{D}$ *specifying* L'_w, $f(|d|) > |w|$.

   Then, there is no recursive trade-off between $DPCFA(2)$ *(or* $DRPCFA(2)$, $1DFA(2)s$*) and* $\mathcal{D}$.

Proof. Proof of 1: Assume that there exists a recursive function $g : \mathbb{N} \mapsto \mathbb{N}$ such that for any $dpcfa(2)$ s specifying $INVALCM(w)$ (notice that $|INVALCM(w)^c| \leq 1$), for any $d \in \mathcal{D}$ specifying $\mathcal{L}(s)$, $g(|s|) > |d|$. Since f is strictly increasing, we know $f(g(|s|)) > f(|d|) > |t|$ where $t \in VALCM(w)$. Clearly, the function $f \circ g$ remains a recursive function. $|t| > x$, where x is the steps that the Turing machine M takes to halt on w. $\Rightarrow f(g(|s|)) > x$. This implies that x is recursively bounded, so the halting problem would be decidable, leading to a contradiction.

Proof of 2: Assume that there exists an increasing recursive function $g : \mathbb{N} \mapsto \mathbb{N}$ such that for any $dpcfa(2)$ N accepting $VALCM(w)$ (notice that $|VALCM(w)| \leq 1$), for any $d \in \mathcal{D}$ specifying $\mathcal{L}(N)$, $g(|N|) > |d|$. Since f is strictly increasing, we have $f(g(|N|)) > f(|d|) > |t|$, where $t \in VALCM(w)$. We know $|t| > x$, where x is the steps that the Turing machine M takes to halt on w. $\Rightarrow f(g(|N|)) > x \Rightarrow$ contradiction. □

The following lemma is essential to demonstrate the application of Theorem 3. Most of the language descriptors involved are defined in standard textbooks, such as [23]. Others are specified as follows: patterns and multi-patterns in [27], various L systems in [20], extended regular expressions (EXREGs) in [30], and synchronized regular expressions (SREs) in [31].

Lemma 3. *For any* $w \in \{0,1\}^*$, *let* $L_w = \{w\}$. *There exists a strictly increasing recursive function* f *such that, for any pattern (or multi-pattern, DFA, NFA, regular expression, EXREG, SRE, DPDA, linear context-free grammar, context-free grammar, D0L system, 0L system, T0L system, EDT0L system, E0L system, ET0L system)* S *specifying* L_w, $f(|S|) > |w|$.

Proof. Several of the conclusions of the lemma are well-known results in language theory. For patterns, multi-patterns, EXREGs, SREs, and L-systems, the results follow from their definition directly. □

The following corollary is a direct application of Theorem 3 and Lemma 3.

Corollary 1. *The trade-off between* $DPCFA(2)$ *(or* $DRPCFA(2)$, $1DFA(2)s$*) and patterns (or multi-patterns, DFAs, NFAs, regular expressions, EXREGs, SREs, DPDAs, linear context-free grammars, context-free grammars, D0L systems, 0L systems, T0L systems, E0L systems, EDT0L systems, and ET0L systems) is non-recursive.*

The following corollary illustrates the power and applicability of Theorem 3 on particular subsets of regular languages.

Corollary 2. *Any class of language descriptors* $\mathcal{D}$ *for which* $\mathcal{L}(\mathcal{D})$ *satisfies any predicate listed below also satisfies condition 1 of Theorem 3. Consequently, there exists no recursive trade-off between* $DPCFA(2)$ *(or* $DRPCFA(2)$, $1DFA(2)s$*) and* $\mathcal{D}$.

1. L *is a star event, i.e.,* $L = (L)^*$.

2. *L is a code event; i.e., there exist strings* $w_1, ..., w_k \in \{0,1\}^*$ *such that* $L = \{w_1, ..., w_k\}^*$.
3. *For all* $k \geq 1$, *L is a k-parsable event, and L is a locally parsable event.*
4. *L is an ultimate definite event, reverse ultimate definite event, or generalized ultimate definite event.*
5. *L is a comet event, reverse comet event, or generalized comet event.*
6. $L = \gamma(L)$, *where* $\gamma(L) = \{y \mid$ *there exists x in L such that* $\mid y \mid = \mid x \mid\}$.
7. *L is prefix closed, i.e.,* $L = \{x \mid$ *there exists y in* $\{0,1\}^*$ *and* $x \cdot y \in L\}$.
8. *L is suffix closed, i.e.,* $L = \{y \mid$ *there exists x in* $\{0,1\}^*$ *and* $x \cdot y \in L\}$.
9. *L is infix closed, i.e.,* $L = \{y \mid$ *there exist x, z in* $\{0,1\}^*$ *and* $x \cdot y \cdot z \in L\}$.
10. *L is co-finite.*
11. *For all* $k \geq 1$, *L is a k-definite event, k-reverse definite event, or k-generalized definite event.*
12. *L is a definite, reverse definite, or generalized definite event.*
13. *For all* $k \geq 1$, *L is a k-testable event.*
14. *For all* $k \geq 1$, *L is k-testable in the strict sense.*
15. *L is locally testable in the strict sense.*
16. *L is locally testable.*
17. *L is a star-free, non-counting, group-free, permutation-free, or LTO event.*
18. *For all* $k > 2$, *L is a CMk event.*
19. *L is accepted by some strongly connected deterministic finite automaton.*
20. *L is accepted by some permutation automaton.*
21. *L is a pure group event.*
22. $L = L^{rev}$.
23. *L is dot-free; i.e., L is denoted by some* $(\cup, \cdot, *, -)$ *regular expression over {0, 1} with no occurrence of "*$\cdot$*"*

Proof. The definitions of the classes of regular sets of 2, 3, and 11 through 18 can be found in [32]. The definition of 4 can be found in [33], 5 in [34], 19 in [35], 20 in [36], and 21 in [37]. The proof that these classes of language descriptors satisfy condition 1 of Theorem 3 is presented in [12]. □

5. Conclusions

This research has marked a profound advancement in elucidating the computational power of $DPCFA(2)$, $DRPCFA(2)$, and $1DFA(2)$ by demonstrating their capacity to accept sets of valid and invalid computations of a deterministic Turing machine. These findings have settled a persistent question from prior studies and confirmed that even some simplest restricted predicates, $= \{0,1\}^* \mid_{|L^c| \leq 1}$ and $= \emptyset \mid_{|L| \leq 1}$, exhibit productiveness, making key decision problems, such as inclusion, equivalence, finiteness, and infiniteness, undecidable/productive for these models. Furthermore, our analysis established a remarkable lack of recursive trade-offs between $DPCFA(2)$ and an extensive range of language descriptor classes, spanning DFAs, subclasses of regular expressions, context-free grammars, extended regular expressions, and L systems. This work has significantly expanded the theoretical landscape of parallel communicating finite automata and set a robust stage for future investigations into descriptional complexity and promise problems. Notably, our results extend to promise problems: for $DPCFA(2)$, given the promise that every language is finite, the emptiness problem remains productive, and given the promise that every language is co-finite, the universality problem retains its productiveness. These insights not only underscore the enduring challenge of undecidability in formal language theory but also highlight the formidable influence of productiveness, revealing how even the simplest predicates wield substantial computational authority and paving the way for transformative theoretical exploration.

Author Contributions: Conceptualization, J.X. and H.B.H.III; methodology, J.X. and H.B.H.III; validation, J.X., H.B.H.III, and C.-S.L.; formal analysis, J.X. and C.-S.L.; investigation, J.X. and C.-S.L.; resources, J.X., H.B.H.III and C.-S.L.; writing—original draft preparation, J.X.; writing—review and editing, J.X., H.B.H.III and C.-S.L.; supervision, H.B.H.III. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No new data were created or analyzed in this study.

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

References

1. Martín-vide, C.; Mateescu, A.; Mitrana, V. Parallel Finite Automata Systems Communicating by States. *Int. J. Found. Comput. Sci.* **2002**, *13*, 733–749. [CrossRef]
2. Ganguly, D.; Chatterjee, K.; Ray, K.S.; Pal, R.K. Parallel Communicating One-Way Reversible Finite Automata System. In Proceedings of the 4th International Conference on Frontiers in Computing and Systems, Mandi, India, 16–17 October 2024; Kole, D.K., Roy Chowdhury, S., Basu, S., Plewczynski, D., Bhattacharjee, D., Eds.; Springer: Singapore, 2024; pp. 179–189.
3. Bordihn, H.; Kutrib, M.; Malcher, A. On the computational capacity of parallel communicating finite automata. *Int. J. Found. Comput. Sci.* **2012**, *23*, 713–732. [CrossRef]
4. Bordihn, H.; Vaszil, G. Reversible parallel communicating finite automata systems. *Acta Inform.* **2021**, *58*, 263–279. [CrossRef]
5. Bordihn, H.; Kutrib, M.; Malcher, A. Undecidability and hierarchy results for parallel communicating finite automata. *Int. J. Found. Comput. Sci.* **2011**, *22*, 1577–1592. [CrossRef]
6. Vollweiler, M. Asynchronous systems of parallel communicating finite automata. *Fundam. Informaticae* **2015**, *136*, 177–197. [CrossRef]
7. Martín-Vide, C.; Mitrana, V. Some undecidable problems for parallel communicating finite automata systems. *Inf. Process. Lett.* **2001**, *77*, 239–245. [CrossRef]
8. Malcher, A. Descriptional Complexity of Cellular Automata and Decidability Questions. *J. Autom. Lang. Comb.* **2002**, *7*, 549–560. [CrossRef]
9. Kutrib, M. Cellular automata—A computational point of view. In *New Developments in Formal Languages and Applications*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 183–227.
10. Smith, A.R. Cellular Automata and Formal Languages. In Proceedings of the 11th Annual Symposium on Switching and Automata Theory (Swat 1970), SWAT '70, Washington, DC, USA, 28–30 October 1970; pp. 216–224. [CrossRef]
11. Terrier, V. On real time one-way cellular array. *Theor. Comput. Sci.* **1995**, *141*, 331–335. [CrossRef]
12. Xie, J.; Hunt, H.B, III. On the Undecidability and Descriptional Complexity of Synchronized Regular Expressions. *Acta Inform.* **2023**, *60*, 257–278. [CrossRef]
13. Xie, J.; Hunt, H.B., III; Stearns, R.E. Pumping Lemmas can be "Harmful". *Theory Comput. Syst.* **2024**, *68*, 1339–1352. [CrossRef]
14. Xie, J.; Hunt, H.B., III; Stearns, R.E. On Productiveness and Complexity in Computable Analysis Through Rice-Style Theorems for Real Functions. *Mathematics* **2024**, *12*, 3248. . [CrossRef]
15. Hartmanis, J. On the succinctness of different representations of languages. In Proceedings of the Automata, Languages and Programming: Sixth Colloquium, ICALP 1979, Graz, Austria, 16–20 July 1979; Maurer, H.A., Ed.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1979; Volume 71, pp. 282–288. [CrossRef]
16. Holzer, M.; Kutrib, M. Scientific Applications of Language Methods. In *Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory*; World Scientific: Singapore, 2010; Chapter Descriptional Complexity—An Introductory Survey; Volume 2, pp. 1–58. [CrossRef]
17. Gruber, H.; Holzer, M. From finite automata to regular expressions and back—A summary on descriptional complexity. *Int. J. Found. Comput. Sci.* **2015**, *26*, 1009–1040. [CrossRef]
18. Broda, S.; Machiavelo, A.; Moreira, N.; Reis, R. A Hitchhiker's Guide to descriptional complexity through analytic combinatorics. *Theor. Comput. Sci.* **2014**, *528*, 85–100. [CrossRef]
19. Kutrib, M.; Pighizzini, G. Recent trends in descriptional complexity of formal languages. *Bull. EATCS* **2013**, *3*.
20. Xie, J.; Hunt, H.B., III; Stearns, R.E. Decision Problems Concerning L Systems. Available online: https://ssrn.com/abstract=4823208 (accessed on 8 April 2025).
21. Rabin, M.O.; Scott, D. Finite automata and their decision problems. *IBM J. Res. Dev.* **1959**, *3*, 114–125. [CrossRef]
22. Rosenberg, A.L. On multi-head finite automata. *IBM J. Res. Dev.* **1966**, *10*, 388–394. [CrossRef]
23. Hopcroft, J.E.; Ullman, J.D. *Introduction to Automata Theory, Languages, and Computation*; Addison-Wesley: Reading, MA, USA, 1979.
24. Rogers, H., Jr. *Theory of Recursive Functions and Effective Computability*; MIT Press: Cambridge, MA, USA, 1987.

25. Soare, R.I. *Recursively Enumerable Sets and Degrees*; Springer: Berlin/Heidelberg, Germany, 1987.
26. Hartmanis, J. Context-Free Languages and Turing Machine Computations. *J. Symb. Log.* **1972**, *37*, 759–759. [CrossRef]
27. Xie, J.; Hunt, H.B., III; Stearns, R.E. On the computational and descriptional complexity of multi-pattern languages. *Theor. Comput. Sci.* **2025**, *1030*, 115063. [CrossRef]
28. Holzer, M.; Kutrib, M.; Malcher, A. Complexity of multi-head finite automata: Origins and directions. *Theor. Comput. Sci.* **2011**, *412*, 83–96. . [CrossRef]
29. Hromkovič, J. One-way multihead deterministic finite automata. *Acta Inform.* **1983**, *19*, 377–384. [CrossRef]
30. Câmpeanu, C.; Salomaa, K.; Yu, S. A formal study of practical regular expressions. *Int. J. Found. Comput. Sci.* **2003**, *14*, 1007–1018. [CrossRef]
31. Della Penna, G.; Intrigila, B.; Tronci, E.; Venturini Zilli, M. Synchronized regular expressions. *Acta Inform.* **2003**, *39*, 31–70. [CrossRef]
32. McNaughton, R.; Papert, S. *Counter-Free Automata*; MIT Press: Cambridge, MA, USA, 1971.
33. Paz, A.; Peleg, B. Ultimate-Definite and Symmetric-Definite Events and Automata. *J. ACM* **1965**, *12*, 399–410. [CrossRef]
34. Paz, A.; Peleg, B. On Concatenative Decompositions of Regular Events. *IEEE Trans. Comput.* **1968**, *C-17*, 229–237. [CrossRef]
35. Hartmanis, J.; Stearns, R.E. *Algebraic Structure Theory of Sequential Machines*; Prentice-Hall International Series in Applied Mathematics; Prentice-Hall, Inc.: Hoboken, NJ, USA, 1966.
36. Thierrin, G. Permutation Automata. *Math. Syst. Theory* **1968**, *2*, 83–90. [CrossRef]
37. McNaughton, R. The loop complexity of pure-group events. *Inf. Control* **1967**, *11*, 167–176. [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.