Not All Multi-Valued Partial CFL Functions Are Refined by Single-Valued Functions (Extended Abstract)

Tomoyuki Yamakami

Not All Multi-Valued Partial CFL Functions Are Refined by Single-Valued Functions (Extended Abstract)

Tomoyuki Yamakami

Department of Information Science, University of Fukui
3-9-1 Bunkyo, Fukui 910-8507, Japan

Abstract. We give an answer to a fundamental question, raised by Konstantinidis, Santean, and Yu [Act. Inform. 43 (2007) 395–417], of whether all multi-valued partial CFL functions can be refined by single-valued partial CFL functions. We negatively solve this question by presenting a special multi-valued partial CFL function as an example function and by proving that no refinement of this particular function becomes a single-valued partial CFL function. This contrasts an early result of Kobayashi [Inform. Control 15 (1969) 95–109] that multi-valued partial NFA functions are always refined by single-valued NFA functions. Our example function turns out to be unambiguously 2-valued, and thus we obtain a stronger separation result, in which no refinement of unambiguously 2-valued partial CFL functions can be single-valued. Our proof consists of manipulations and close analyses of underlying one-way one-head nondeterministic pushdown automata equipped with write-only output tapes.

Keywords: multi-valued partial function, CFL function, NFA function, refinement, pushdown automaton, context-free language, stack history

1 Resolving a Fundamental Question

Since early days of automata and formal language theory, multi-valued partial functions,[⋆] which are also known as transductions, computed by various types of automata equipped with supplemental write-only output tapes have been investigated extensively. Among them, we intend to spotlight *CFL functions* (also known as algebraic transductions), which are computed by *one-way one-head nondeterministic pushdown automata* (succinctly abbreviated as npda's) *with write-only output tapes*. These functions naturally inherit certain distinctive traits from context-free languages; however, their behaviors are in essence quite different from the behaviors of the language counterpart. Intriguing properties of those functions have been addressed in the past literature (e.g., [1–3, 6, 14]).

When the number of output values is restricted to at most one, we obtain *single-valued* functions. Concerning a relationship between multi-valued and single-valued partial functions, multi-valued partial functions in general cannot

[⋆] We often call those multi-valued partial functions just "functions."

be single-valued; therefore, it is more appropriate to ask a question of whether multi-valued partial functions can be *refined* by single-valued partial functions, where "refinement" refers to a certain natural restriction on the outcomes of multi-valued functions. To be more precise, we say that a function g is a *refinement* of another function f [7] (which was also called "uniformization" [6]) if and only if (i) f and g have the same domain and (ii) for every input x in the domain of f, all output values of g on x are also output values of f on the same input x. When g is particularly single-valued, g acts as a "selecting" function that picks exactly one value from a set of output values of f on x. This refinement notion is known to play a significant role in language recognition. In a polynomial-time setting, for instance, if we can effectively find an accepting computation path of any polynomial-time nondeterministic Turing machine, then every multi-valued partial NP function (computed by a certain polynomial-time nondeterministic Turing machine) has a refinement in the form of single-valued NP function. The "no-refinement" claim therefore leads to a negative answer to the long-standing P =?NP question.

We intend to discuss the same refinement question regarding CFL functions. In this line of research, the first important step was taken by Kobayashi [5] in 1969. He gave an affirmative answer to the refinement question for multi-valued partial NFA functions, which are computed by one-way one-head nondeterministic finite automata (or nfa's, in short) with write-only output tapes; namely, multi-valued partial NFA functions can be refined by certain single-valued partial NFA functions. Konstantinidis, Santean, and Yu [6] discussed the same question for CFL functions. They managed to obtain a partial affirmative answer but left the whole question open, probably due to a technical limitation of their algebraic treatments of CFL functions.

This paper is focused on CFL functions whose output values are particularly produced by npda's running in linear time** (that is, all computation paths terminate in time $O(n)$, where n is the size of input) with write-only output tapes. By adopting succinct notations from [12], we express as CFLMV a collection of all such CFL functions and we also write CFLSV for a collection of all single-valued functions in CFLMV. As a concrete example of our CFL function, let us consider f defined by setting $f(1^n\#x)$ to be a set of all substrings of x of length between 1 and n exactly when $1 \leq n \leq |x|$. This function f is a multi-valued partial CFL function and the following function g is an obvious refinement of f: let $g(1^n\#x)$ consist only of the first symbol of x whenever $1 \leq n \leq |x|$. Notice that g belongs to CFLSV.

Given two classes $\mathcal{F}$ and $\mathcal{G}$ of multi-valued partial functions, the notation $\mathcal{F} \sqsubseteq_{ref} \mathcal{G}$ means that every function in $\mathcal{F}$ can be refined by a certain function in $\mathcal{G}$. Using these notations, the aforementioned refinement question regarding CFL functions can be neatly rephrased as follows.

** This linear time-bound ensures that every CFL function produces only at most an exponential number of output values and it therefore becomes an NP function. This fact naturally extends a well-known containment of CFL $\subseteq$ NP.

Question 1 *Is it true that* CFLMV $\sqsubseteq_{ref}$ CFLSV*?*

Various expansions of CFLMV are possible. Yamakami [14], for instance, introduced a hierarchy $\{\Sigma_k^{\mathrm{CFL}}\mathrm{MV}, \Pi_k^{\mathrm{CFL}}\mathrm{MV} \mid k \geq 1\}$ of multi-valued partial functions built upon CFL functions by applying Turing relativization and a complementation operation. Its single-valued version is customarily denoted by $\{\Sigma_k^{\mathrm{CFL}}\mathrm{SV}, \Pi_k^{\mathrm{CFL}}\mathrm{SV} \mid k \geq 1\}$. Our focal question, Question 1, can be further generalized to the following.

Question 2 *Does* $\Sigma_k^{\mathrm{CFL}}\mathrm{MV} \sqsubseteq_{ref} \Sigma_k^{\mathrm{CFL}}\mathrm{SV}$ *hold for each index* $k \geq 1$*?*

Yamakami [14] also shed partial light on this general question when $k \geq 3$. He showed that, for every index $k \geq 3$, $\Sigma_{k-1}^{\mathrm{CFL}} = \Sigma_k^{\mathrm{CFL}}$ implies $\Sigma_k^{\mathrm{CFL}}\mathrm{MV} \sqsubseteq_{ref} \Sigma_k^{\mathrm{CFL}}\mathrm{SV}$, where Σ_k^{CFL} is the kth level of the *CFL hierarchy* [13], which is a natural analogue of the well-known polynomial(-time) hierarchy. Since the collapse of the CFL hierarchy is closely related to that of the polynomial hierarchy, an answer to Question 2 (when $k \geq 3$) could be quite difficult to obtain. Nevertheless, the remaining cases of $k = 1, 2$ have been left unsolved.

In this paper, without relying on any unproven assumptions, we solve Question 2 negatively when $k = 1$; therefore, our result completely settles Question 1. Our solution actually gives an essentially stronger statement than what we have discussed so far. To clarify this point, we first introduce a function class CFL2V as a collection of all functions f in CFLMV satisfying the condition that the number of output values of f on each input should be at most 2.

Theorem 3. CFL2V $\not\sqsubseteq_{ref}$ CFLSV.

Since CFLSV $\subseteq$ CFL2V $\subseteq$ CFLMV holds, Theorem 3 clearly leads to a negative answer to Question 1. The proof of the theorem is essentially a manifestation of the following intuition: an npda relying on limited functionality of its memory device cannot simulate two independent computation paths simultaneously.

Instead of providing a detailed proof for Theorem 3, we wish to present a simple and clear argument to demonstrate a slightly stronger result regarding a subclass of CFL2V. To explain such a subclass, we first address that even if a function f is single-valued, its underlying npda may have numerous computation paths producing the same value of f on each input. Let us call an npda N with a write-only output tape *unambiguous* if, for every input x and any output value y, N has exactly one accepting computation path producing y. Let UCFL2V denote a class of all 2-valued partial functions computed in linear time by unambiguous npda's with output tapes. Succinctly, those functions are called *unambiguously 2-valued*. Obviously, UCFL2V $\subseteq$ CFL2V holds.

Throughout this paper, we wish to show the following stronger separation result (than Theorem 3), which is referred to as the "main theorem."

Theorem 4 (main theorem). UCFL2V $\not\sqsubseteq_{ref}$ CFLSV.

Following a brief explanation of key notions and notations in Section 2, we give in Section 3 the proof of Theorem 4, completing the proof of Theorem 3 as

well. Our proof starts in Sections 3.1 with a presentation of our example function h_3, a member of UCFL2V. The proof then proceeds, by contradiction, with an assumption that a certain refinement, say, g of h_3 belongs to CFLSV. In Section 3.2, the proof requires an introduction of "colored" automaton—a new type of automaton having no output tape—which simulates any npda equipped with an output tape that computes g. To lead to the desired contradiction, the proof further exploits special properties of such a colored automaton by analyzing the behaviors of its stack history (i.e., time transitions of stack contents) generated by this colored automaton. The detailed analysis is presented in Sections 3.3–3.6. All proofs omitted here will appear in a forthcoming complete paper.

2 Preliminaries

We wish to explain key notions and notations necessary to read through the rest of this paper.

Let $\mathbb{N}$ denote a set of all nonnegative integers and define $\mathbb{N}^+ = \mathbb{N} - \{0\}$. Given a number $n \in \mathbb{N}^+$, the notation $[n]$ expresses an integer set $\{1, 2, 3, \ldots, n\}$. An *alphabet* is a finite nonempty set of "symbols" or "letters." Given alphabet Σ, a *string over* Σ is a finite series of symbols taken from Σ and $|x|$ denotes the *length* (or *size*) of string x. We use λ for the *empty string*. A *language over* Σ is a subset of Σ^*, where Σ^* is a set of all strings over Σ. Given two strings x and y over the same alphabet, $x \sqsubseteq y$ indicates that x is a *substring* of y; namely, for certain two strings u and v, y equals uxv. Moreover, given a string x and an index $i \in [|x|]$, the notation $(x)_i$ expresses a unique substring made up only of the first i symbols of x. Clearly, $(x)_i \sqsubseteq x$ holds. The notation $|C|$ for a finite set A refers to its *cardinality*.

Let us consider multi-valued partial functions, each of which maps elements of a given set to subsets of another set. Slightly different from a conventional notation (e.g., [7, 8]), we write $f : A \to \mathcal{P}(B)$ for two sets A and B to refer to a multi-valued partial function that takes an element in A as input and produces a certain number of elements in B, where $\mathcal{P}(A)$ denotes the *power set* of A. In particular, when $f(x) = \emptyset$, we briefly say that $f(x)$ is *undefined*. The *domain* of f, denoted by $\mathrm{dom}(f)$, is the set $\{x \in A \mid f(x) \text{ is not undefined }\}$. Given a constant $k \in \mathbb{N}^+$, f is *k-valued* if $|f(x)| \leq k$ holds for every input x in A. For two multi-valued partial functions $f, g : A \to \mathcal{P}(B)$, we say that g is a *refinement* of f (or f is *refined by* g), denoted $f \sqsubseteq_{ref} g$, if (i) $\mathrm{dom}(f) = \mathrm{dom}(g)$ and (ii) $g(x) \subseteq f(x)$ (set inclusion) holds for every $x \in \mathrm{dom}(g)$. For any function classes $\mathcal{F}$ and $\mathcal{G}$, the succinct notation $\mathcal{F} \sqsubseteq_{ref} \mathcal{G}$ means that every function in $\mathcal{F}$ has a refinement in $\mathcal{G}$.

Our mechanical model of computation is a *one-way one-head nondeterministic pushdown automaton* (or an *npda*, in short) with/without a write-only output tape, allowing λ-moves (or λ-transitions). We use an infinite input tape, which holds two special endmarkers: the left endmarker ¢ and the right endmarker \$. In addition, we use a semi-infinite output tape on which its tape head is initially positioned at the first (i.e., the leftmost) tape cell and moves in one

direction to the right unless it stays still. Formally, an npda M with an output tape is a tuple $(Q, \Sigma, \{\mathfrak{c}, \$\}, \Gamma, \Theta, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$ with a finite set Q of inner states, an input alphabet Σ, a stack alphabet Γ, an output alphabet Θ, the initial state $q_0 \in Q$, the bottom marker $Z_0 \in \Gamma$, a set Q_{acc} (resp., Q_{rej}) of accepting (resp., rejecting) states with $Q_{halt} \subseteq Q$, and a transition function $\delta : (Q - Q_{halt}) \times (\check{\Sigma} \cup \{\lambda\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^* \times (\Theta \cup \{\lambda\}))$, where $\check{\Sigma} = \Sigma \cup \{\mathfrak{c}, \$\}$ and $Q_{halt} = Q_{acc} \cup Q_{rej}$. We demand that M should neither remove Z_0 nor replace it with any other symbol at any step of its computation. Furthermore, the output tape is *write-only*; namely, whenever M writes a non-blank symbol on this tape, its tape head must move to the right. It is important to recognize two types of λ-move. When δ is applied to (q, λ, γ), M modifies the current contents of its stack and its output tape while neither scanning input symbols nor moving its input-tape head. When $(p, w, \lambda) \in \delta(q, \sigma, \gamma)$ holds, M neither moves its output-tape head nor writes any non-blank symbol onto the output tape.

Whenever we need to discuss an npda *having no output tape*, we automatically drop "Θ" as well as "$\Theta \cup \{\lambda\}$" from the above definition of M and δ. As stated in Section 1, we consider only npda's whose computation paths are all terminate within $O(n)$ steps, where n refers to any input size, and this particular condition concerning the termination of computation is conventionally called the *termination condition* [13]. Throughout this paper, all npda's are implicitly assumed to satisfy this termination condition.

In general, an *output* (outcome or output string) of M along a given computation path refers to a string over Θ written down on the output tape when the path terminates. Such an output is classified as being *valid* (or *legitimate*) if the corresponding computation path is an accepting computation path (i.e., M enters an accepting state along this path). We say that an npda M with an output tape *computes* function f if, on every input x, M produces exactly all the strings in $f(x)$ as valid outputs; namely, for every pair x, y, $y \in f(x)$ if and only if y is a valid outcome of M on input y. Notice that an npda can generally produce more than one valid output strings, its computed function inherently becomes multi-valued. Because invalid outputs produced by M are all discarded from our arguments in the subsequent sections, we will refer to valid outputs as just "outputs" unless otherwise stated.

The notation CFLMV (resp., CFLkV for a fixed $k \in \mathbb{N}^+$) stands for a class of multi-valued (resp., k-valued) partial functions that can be computed by npda's with write-only output tapes in linear time. When $k = 1$, we customarily write CFLSV instead of CFL1V. In addition, we define UCFLkV as a collection of all functions f in CFLkV for which a certain npda with an output tape computes f with the extra condition (called the *unambiguous computation condition*) that, for every input x and every value $y \in f(x)$, there exists exactly one accepting computation path producing y on input x. It then follows that UCFLkV $\subseteq$ CFLkV $\subseteq$ CFLMV. Since any function producing exactly $k + 1$ values cannot be in CFLkV by definition, CFLkV $\neq$ CFL$(k + 1)$V holds; thus, in particular, we obtain CFLSV $\neq$ CFLMV. Notice that this inequality does not directly lead to the desired conclusion CFLMV $\not\sqsubseteq_{ref}$ CFLSV.

To describe behaviors of an npda's stack, we closely follow terminology from [10, 11]. A *stack content* is formally a series $z_m z_{m-1} \cdots z_1 z_0$ of stack symbols sequentially stored into a stack in such a way that z_0 is the bottom marker Z_0 and z_m is a symbol at the top of the stack. We sometimes refer to a stack content obtained just after the tape head scans and moves off the ith cell of the input tape as a *stack content at the ith position*.

3 Proof of the Main Theorem

Our ultimate goal is to solve negatively a question that was posed in [6] and reformulated in [14] as in the form of Question 1. In what follows, we will present an example function, called h_3, which belongs to UCFL2V, and then give an explanation of why no refinement of this function is found in CFLSV, resulting in the main theorem, namely, UCFL2V $\not\sqsubseteq_{ref}$ CFLSV.

3.1 An Example Function

Our example function h_3 is a natural extension of a well-recognized deterministic context-free language $\{x\#x^R \mid x \in \{0,1\}^*\}$ (marked even-length palindromes), where $\#$ is a distinguished symbol not in $\{0,1\}$. Let us define two supporting languages $L = \{x_1\#x_2\#x_3 \mid x_1, x_2, x_3 \in \{0,1\}^*\}$ and $L_3 = \{w \mid w = x_1\#x_2\#x_3 \in L, \exists (i,j) \in I_3\ [x_i^R = x_j]\}$, where $I_3 = \{(i,j) \mid i,j \in \mathbb{N}^+, 1 \leq i < j \leq 3\}$. We then introduce the desired function h_3 by setting $h_3(w) = \{0^i 1^j \mid (i,j) \in I_3, x_i^R = x_j\}$ if $w = x_1\#x_2\#x_3 \in L$, and $h_3(w) = \emptyset$ if w is not in L. It thus follows that $L_3 = \{w \in L \mid h_3(w) \neq \emptyset\}$. Now, let us claim the following assertion.

Proposition 1. *The above function h_3 is in* UCFL2V.

Proof. Obviously, h_3 is 2-valued. Let us consider the following npda M equipped with a write-only output tape. On any input w, M checks whether w is of the form $x_1\#x_2\#x_3$ in L by moving its input-tape head from left to right by counting the number of $\#$ in w. At the same time, M nondeterministically chooses a pair $(i,j) \in I_3$, writes $0^i 1^j$ onto its output tape, stores x_i into a stack, and then checks whether x_i^R matches x_j by retrieving x_i in reverse from the stack. If $x_i^R = x_j$ holds, then M enters an accepting state; otherwise, it enters a rejecting state. It follows by this definition that, for each choice of (i,j) in I_3, there is at most one accepting computation path producing $0^i 1^j$. It is not difficult to show that M computes h_3. Therefore, h_3 belongs to UCFL2V.

To complete the proof of the main theorem, it suffices to verify the following proposition regarding the existence of refinements of the function h_3.

Proposition 2. *The function h_3 has no refinement in* CFLSV.

3.2 Colored Automata

Our proof of Proposition 2 proceeds by contradiction. To lead to the desired contradiction, we first assume that h_3 has a refinement, say, g in CFLSV. Since g is single-valued, we rather write $g(x) = y$ instead of $g(x) = \{y\}$ for $x \in \mathrm{dom}(f)$. Take an npda N computing g with a write-only output tape. Let N have the form $(Q, \Sigma, \{\mathfrak{c}, \$\}, \Gamma, \Theta, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$ with $\delta : (Q - Q_{halt}) \times (\check{\Sigma} \cup \{\lambda\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^* \times (\Theta \cup \{\lambda\}))$, where $\Sigma = \Theta = \{0, 1\}$.

Unfortunately, we find it difficult to directly analyze the moves of its output-tape head. To overcome this difficulty, we then try to modify N into a new variant of *npda having no output tape*, say, M. As seen later, this modification is possible because g's output values are limited to strings of constant lengths. Now, let us introduce this new machine, dubbed as "colored" automaton, which has no output tapes but uses "colored" stack symbols. Using a finite set C of "colors," a *colored automaton* $M = (Q', \Sigma, \{\mathfrak{c}, \$\}, \Gamma', C, \delta', q'_0, Z_0, Q'_{acc}, Q'_{rej})$ partitions its stack alphabet Γ', except for the bottom marker, into sets $\{\Gamma_\xi\}_{\xi \in C}$; namely, $\bigcup_{\xi \in C} \Gamma_\xi = \Gamma - \{Z_0\}$ and $\Gamma_\xi \cap \Gamma_{\xi'} = \emptyset$ for any distinct pair $\xi, \xi' \in C$. We define a *color* of stack symbol γ to be ξ in C if γ is in Γ'_ξ $(= \Gamma_\xi \cup \{Z_0\})$. Notice that Z_0 has three colors. Given a color $\xi \in C$, we call a computation path of M a *ξ-computation path* if all configurations along this computation path use only stack symbols in color ξ. An *output* of M on input x is composed of all colors ξ in C for which there is an accepting ξ-computation path of M on x.

Lemma 1. *There exists a colored automaton M that computes g.*

Proof Sketch. Recalling the set I_3, we introduce a set $\bar{I}_3 = \{0^i 1^j \mid (i, j) \in I_3\}$ and another set $\bar{I}_3^{part}$ composed of all substrings of any strings in $\bar{I}_3$. Recall the given npda N with a write-only output tape. Now, we want to define a new colored automata $M = (Q', \Sigma, \{\mathfrak{c}, \$\}, \Gamma', \bar{I}_3, \delta', q'_0, Z_0, Q'_{acc}, Q'_{rej})$ that simulates N as follows. Roughly speaking, on any input x, M first *guesses* (i.e., nondeterministically chooses) an output string t of $g(x)$. Whenever N pushes u, M pushes its corresponding color-t symbol $u^{(t)}$ into a stack. Further along this computation path, M keeps using only color-t stack symbols. Instead of having an output tape, M remembers the currently produced string on N's output tape. Whenever N enters an accepting state with an output string that matches the firstly guessed string t of M, M also enters an appropriate accepting state. In other cases, M rejects the input. □

To simplify notations in our argument, we describe the colored automaton M guaranteed by Lemma 1 as $(Q, \Sigma, \{\mathfrak{c}, \$\}, \Gamma, I_3, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$. It is also useful to restrict the "shape" of M. A colored automaton M is said to be *in an ideal shape* if M satisfies all of the following six conditions.

1. There are only one accepting state q_{acc} and one rejecting state q_{rej}. Moreover, the set Q of inner states equals $\{q_0, q, q_{acc}, q_{rej}\}$. The machine M is in state q during its computation except for the initial and final configurations.
2. The input-tape head of M always moves.

3. The machine M never aborts its computation; that is, δ is a total function (i.e., $\delta(q, \sigma, \gamma) \neq \emptyset$ holds for any $(q, \sigma, \gamma) \in (Q - Q_{halt}) \times \check{\Sigma} \times \Gamma$).
4. Every stack operation either modifies a single top stack symbol or pushes extra one symbol onto the top of the stack after (possibly) altering the then-top symbol; that is, δ's range is $\mathcal{P}(Q \times (\Gamma \cup \Gamma^2) \times (\Theta \cup \{\lambda\}))$.
5. The stack never becomes empty (excluding the bottom marker Z_0) at any step of the computation except for the initial and the final configurations. In addition, at the first step of reading ¢, M must push a stack symbol onto Z_0 and this stack symbol determines the stack color in the rest of its computation path. After reading \$, M's stack becomes empty.
6. The machine never enters any halting state before scanning the endmarker.

It is well-known that, for any context-free language L, there always exists an npda (with no output tape) in an ideal shape that recognizes L (see, e.g., [4]). Similarly, we can assert the following statement for colored automata.

Lemma 2. *Given any colored automaton, there is always another colored automaton in an ideal shape that produces the same set of output values.*

In the rest of this paper, we fix a colored automaton in an ideal shape, guaranteed by Lemma 2, which computes g correctly.

Hereafter, let us focus on inputs of the form $x\#x^R\#y$ for $x, y \in \{0,1\}^*$. For any $x \in \{0,1\}^*$, we abbreviate the set $\{y \in \{0,1\}^{|x|} \mid y \notin \{x, x^R\}\}$ as H_x. Given $n \in \mathbb{N}^+$, $D^{(n)}_{(i,j)}$ denotes a set of all strings $x \in \{0,1\}^n$ for which there exists an accepting (i,j)-computation path of M on input $x\#x^R\#x$. Obviously, it holds that $D^{(n)}_{(1,2)} \cup D^{(n)}_{(2,3)} = \{0,1\}^n$. It therefore holds, for every length n, that either $|D^{(n)}_{(1,2)}| \geq 2^n/2$ or $|D^{(n)}_{(2,3)}| \geq 2^n/2$. We will discuss the case of $|D^{(n)}_{(2,3)}| \geq 2^n/2$ in Section 3.3 and the case of $|D^{(n)}_{(1,2)}| \geq 2^n/2$ in Section 3.6.

3.3 Case 1: $D_{(2,3)}$ is Large

Let us consider the first case where the inequality $|D^{(n)}_{(2,3)}| \geq 2^n/2$ holds for infinitely many lengths $n \in \mathbb{N}$. Take an arbitrary number $n \in \mathbb{N}$ that is significantly larger than $3^{|Q|+|\Sigma|+|\Gamma|}$ and also satisfies $|D^{(n)}_{(2,3)}| \geq 2^n/2$. We fix such a number n throughout our proof and we thus tend to drop script "n" whenever its omission is clear from the context; for instance, we often write $D_{(2,3)}$ instead of $D^{(n)}_{(2,3)}$.

By the property of the colored automaton M computing g, it follows that, for any pair $x, y \in \{0,1\}^n$, if $y \notin \{x, x^R\}$, then there always exists a certain accepting $(1,2)$-computation path on input $x\#x^R\#y$; however, there is no accepting $(1,2)$-computation path on input $x\#x^R\#x$ for every x in $D_{(2,3)}$. In addition, no accepting $(1,2)$-computation path exists on input $x\#z\#y$ if $z \neq x^R$. Since there could be a large number of accepting $(1,2)$-computation paths of M on $x\#x^R\#y$, we need to choose one of them arbitrarily and take a close look at this particular path.

For convenience, let $PATH_n$ denote a set of all possible accepting $(1,2)$-computation paths of M on inputs of the form $x\#x^R\#y$ for certain strings $x, y \in \{0,1\}^n$. We arbitrarily fix a *partial assignment* $\pi : D_{(1,2)} \times \{0,1\}^n \to PATH_n$ that, for any element (x,y), if $y \in H_x$, then π picks an accepting $(1,2)$-computation path of M on input $x\#x^R\#y$; otherwise, let $\pi(x,y)$ be undefined for simplicity. For brevity, we abbreviate $\pi(x,y)$ as $p_{x,y}$. Note that $p_{x,y}$ is uniquely determined from (x,y) whenever $\pi(x,y)$ is defined.

Given an accepting $(1,2)$-computation path $p_{x,y}$ of M on input $x\#x^R\#y$, the notation $\gamma_{i,y}^{(x)}$ denotes a stack content obtained by M just after reading off the first i symbols of $x\#x^R\#y$ along this particular path $p_{x,y}$. Furthermore, we abbreviate as $\gamma_y^{(x)}$ the stack content $\gamma_{|x\#x^R\#|,y}^{(x)}$, which is produced just after reading $x\#x^R\#$ of the input $x\#x^R\#y$. Note that, for each $x \in D_{(2,3)}$ and any $y \in H_x$, along an accepting $(1,2)$-computation path $p_{x,y}$ on input $x\#x^R\#y$, M produces unique stack contents $\gamma_{|x\#|,y}^{(x)}$ and $\gamma_y^{(x)}$.

In Sections 3.4–3.6, we plan to evaluate how many strings in $D_{(2,3)}$ satisfy each of the following conditions.

1. Strings x in $D_{(2,3)}$ that make $\gamma_y^{(x)}$ small in size for all $y \in H_x$.
2. Strings x in $D_{(2,3)}$ that make $\gamma_y^{(x)}$ relatively large in size for certain strings $y \in H_x$.

Proposition 3 gives a lower bound of the number of strings in (1), whereas Propositions 4 and 5 provide lower bounds for (2). Those bounds, moreover, guarantee the existence of a string that satisfies both conditions, clearly leading to the desired contradiction.

3.4 Fundamental Properties of a Stack History

In the following series of lemmas and propositions, we will explore fundamental properties of a stack history of M along computation path $p_{x,y}$ on input of the form $x\#x^R\#y$. Those properties are essential in proving the main theorem.

Lemma 3. *Fix $x, y \in \{0,1\}^n$. For any accepting $(1,2)$-computation path $p_{x,y}$ of M on input $x\#x^R\#y$, there is no pair (i_1, i_2) of positions such that $|x| < i_1 < i_2 \leq |x\#x^R\#|$ and $\gamma_{i_1,y}^{(x)} = \gamma_{i_2,y}^{(x)}$. Moreover, the same statement is true when $1 \leq i_1 < i_2 \leq |x|$.*

Lemma 3 can be generalized as follows.

Lemma 4. *Let $x_1, x_2, y_1, y_2 \in \{0,1\}^n$, $i_1, i_2 \in \mathbb{N}$ with $1 \leq i_1, i_2 \leq |x_1\#x_1^R\#|$. Assume that one of the following conditions holds: (i) $i_1 \neq i_2$, (ii) $1 \leq i_1 = i_2 \leq |x_1\#|$ and $(x_1)_{i_1} \neq (x_2)_{i_2}$, and (iii) $(x_1)_{|x_1\#|} = (x_2)_{|x_1\#|}$, $|x_1\#| < i_1 = i_2 \leq |x_1\#x_1^R\#|$, and $(x_1)_{i_1} \neq (x_2)_{i_2}$. It then holds that $\gamma_{i_1,y_1}^{(x_1)} \neq \gamma_{i_2,y_2}^{(x_2)}$.*

Now, we start estimating the lower bound of the number of strings x in $D_{(2,3)}$ for which their corresponding stack contents $\gamma_y^{(x)}$ are small in size for an arbitrary string y in H_x. More specifically, we will verify the following statement.

Proposition 3. *There exist two constants $d_1, d_2 \in \mathbb{N}^+$, independent of (n, x, y), such that $|\{x \in D_{(2,3)} \mid \forall y \in H_x\, [|\gamma_y^{(x)}| < d_1]\}| \geq |D_{(2,3)}| - d_2$.*

Hereafter, we will aim at proving Proposition 3.

Given two strings $u, v \in (\Gamma_{(1,2)})^*$ and a string $z \in \{0,1\}^*$, we say that M *transforms u to v while reading z (along computation (sub)path p)* if M behaves as follows along this subpath p: (i) M starts in state q with uZ_0 in stack, scanning the leftmost input symbol of z, (ii) M then reads z, with no endmarkers, from the input tape, (iii) after reading off z, M enters state q with vZ_0 in stack, and (iv) M does not empty the stack (except for Z_0). The notation $TF_M(\tau, \sigma)$ expresses a set of all strings of the form $z\#z'$ for $z, z' \in \{0,1\}^*$ such that M transforms τ to σ while reading $z\#z'$.

Lemma 5. *Given any pair (u, v), there is at most one string x' such that x' is a substring of a certain string x in $D_{(2,3)}$ and M transforms u to v while reading x' along a subpath of $p_{x,y}$ for a certain $y \in \{0,1\}^n$.*

Next, we will show a key lemma, necessary to prove Proposition 3. Given a pair (x, y), we define $MSC_{x,y}$ (minimal stack contents) to be a collection of all stack contents γ satisfying the following: there exists a position ℓ with $|x\#| \leq \ell \leq |x\#x^R\#|$ such that (i) $\gamma = \gamma_{\ell,y}^{(x)}$ and (ii) $|\gamma| \leq |\gamma_{\ell',y}^{(x)}|$ holds for any ℓ' satisfying $|x\#| \leq \ell' \leq |x\#x^R\#|$. Condition (ii) indicates that the size of γ is minimum. Note that, when $y \in H_x$, $MSC_{x,y}$ cannot be empty. In addition, by Lemma 4, all elements in $\{\gamma_{i,y}^{(x)} \mid 1 \leq i \leq |x\#x^R\#|\}$ are mutually distinct.

Lemma 6. *There exists a constant $d > 0$, independent of (n, x, y), that satisfies the following statement. Let $x \in \{0,1\}^n$, $y \in H_x$, and $\gamma_{\ell,y}^{(x)} \in MSC_{x,y}$. Moreover, let $x = rz$, $x^R = z^R sr'$, $\ell = |x\#z^R s|$, $\gamma_{|r|,y}^{(x)} = \tau v Z_0$, and $\gamma_{\ell,y}^{(x)} = \sigma v Z_0$ for an appropriate tuple $(r, r', z, s, \sigma, \tau, u, v)$. If $\ell \neq |x\#|$ and $z\#z^R s \in TF_M(\tau, \sigma)$, then $|\gamma_y^{(x)}| \leq d$ holds. Moreover, when n is sufficiently large, $\ell \neq |x\#|$ holds.*

Assuming that Lemma 6 is true, we can prove Proposition 3 in the following manner. Since $MSC_{x,y}$ is non-empty, take an element $\gamma_{\ell,y}^{(x)}$ from $MSC_{x,y}$ with $|x\#| \leq \ell \leq |x\#x^R\#|$. By the size-minimality of $\gamma_{\ell,y}^{(x)}$, there exists an appropriate tuple $(r, r', z, s, \sigma, \tau, u, v)$ that satisfies

(*) $x = rz, x^R = z^R sr', \ell = |x\#z^R s|, \gamma_y^{(x)} = uvZ_0, \gamma_{|r|,y}^{(x)} = \tau v Z_0, \gamma_{\ell,y}^{(x)} = \sigma v Z_0$, and $z\#z^R s \in TF_M(\tau, \sigma)$.

By the second part of Lemma 6, except for a certain constant number of x's, it always holds that $\ell \neq |x\#|$. The first part of Lemma 6 provides the desired constant d_1 that upper-bounds $|\gamma_y^{(x)}|$. We therefore obtain the proposition.

To complete the proof of Proposition 3, we still need to verify Lemma 6. This lemma follows from Lemmas 7 and 8. In the first lemma, we want to show that the size of s in (*) is bounded from above by a certain absolute constant.

Lemma 7. *There exists a constant $d_1 > 0$, independent of (n, x, y), satisfying the following statement. Let $x \in \{0,1\}^n$, $y \in H_x$, and $\gamma^{(x)}_{\ell,y} \in MSC_{x,y}$. Moreover, let $x = rz$, $x^R = z^R sr'$, $\ell = |x\#z^R s|$, $\gamma^{(x)}_{|r|,y} = \tau v Z_0$, and $\gamma^{(x)}_{\ell,y} = \sigma v Z_0$. If $\ell \neq |x\#|$ and $z\#z^R s \in TF_M(\tau, \sigma)$, then $|s| \leq d_1$ holds.*

Proof. Let $x = rz$, $x^R = z^R sr'$, $\ell = |x\#z^R s|$, $\gamma^{(x)}_{|r|,y} = \tau v Z_0$, and $\gamma^{(x)}_{\ell,y} = \sigma v Z_0$. Since $\ell \neq |x\#|$, it follows that $z \neq \lambda$. Assume that $\gamma^{(x)}_{\ell,y} \in MSC_{x,y}$ and $z\#z^R s \in TF_M(\tau, \sigma)$. We first claim that s can be uniquely determined from (τ, σ).

Claim. Let $z_1 \in \{0,1\}^+$ and $s_1 \in \{0,1\}^*$. If $z_1\#z_1^R s_1 \in TF_M(\tau, \sigma)$, then $s = s_1$.

Let us show this claim. Toward a contradiction, we assume that $s \neq s_1$. Assume that M has an accepting $(1,2)$-computation path p_1 while reading $rz\#z^R sr'$. Replace a portion of this path associated with $z\#z^R s$ by a subpath corresponding to $z_1\#z_1^R s_1$. We then obtain a new accepting $(1,2)$-computation path on $rz_2\#z_2^R s_2 r'$. However, we obtain $(rz_1)^R = z_1^R r^R = z_1 sr' \neq z_1^R s_1 r'$ because $s \neq s_1$. This means that there is no accepting $(1,2)$-computation path on $rz_1\#z_1^R s_1 r'$, a contradiction. Therefore, the claim is true.

The above claim helps us define a map from (τ, σ) to s. Thus, the number of all possible strings s is at most $|\Gamma'_{(1,2)}|^2$. This implies that $|s|$ is upper-bounded by an appropriately chosen constant, independent of (n, x, y). □

In the second lemma, we want to show that the size of r' in (*) is also upper-bounded by a certain absolute constant.

Lemma 8. *There exists a constant $d_2 > 0$, independent of (n, x, y), that satisfies the following statement. Let $x \in \{0,1\}^n$, $y \in H_x$, and $\gamma^{(x)}_{\ell,y} \in MSC_{x,y}$. Moreover, let $x = rz$, $x^R = z^R sr'$, $y = r''z'$, $\ell = |x\#z^R s|$, $\ell' = |x\#x^R\#r''|$, $\gamma^{(x)}_{|r|,y} = \tau v Z_0$, $\gamma^{(x)}_{\ell,y} = \sigma v Z_0$, and $\gamma^{(x)}_{\ell',y} = v Z_0$. If $r'\#r'' \in TF_M(\sigma, \lambda)$, then $|r'| \leq d_2$ holds.*

Finally, we will prove Lemma 6 with the help of Lemmas 7 and 8.

Proof of Lemma 6. Let $x = rz$ and $x^R = z^R sr'$. Let $\gamma^{(x)}_y = uvZ_0$, $\gamma^{(x)}_\ell = \sigma v Z_0$ with $\ell = |x\#z^R s|$. Assume that M transforms σ to u while reading r'. We first claim that $\ell \neq |x\#|$. Assume that $\ell = |x\#|$. This implies that $z = s = \lambda$. Hence, $x^R = r'$. By Lemma 8, we obtain $|r'| \leq d_2$. However, x must be sufficiently large in size, a contradiction. Therefore, $\ell \neq |x\#|$ holds.

Lemma 7 yields an appropriate constant d_1 such that $|s| \leq d_1$. Lemma 8 also shows that $|r'|$ is upper-bounded by a certain constant, say, d_2. Since $|r| = |sr'| = |s| + |r'|$ by definition, $|r|$ is bounded from above by $d_1 + d_2$. Let σ_0 be the stack symbol pushed into the stack at the first step of M. Since M transforms σ_0 to τv while reading r for a certain τ and the stack increases by at most one, it follows that $|v|$ (and therefore $|uvZ_0|$) is upper-bounded by an appropriately chosen constant. □

In the subsequent argument, the notation E_x expresses a collection of all stack contents $\gamma^{(x)}_y$ at the $|x\#x^R\#|$-th position (i.e., just after reading off $x\#x^R\#$)

along any accepting $(1,2)$-computation path $p_{x,y}$ of M on input $x\#x^R\#y$ for an *arbitrary* string $y \in H_x$. Since π is fixed, it holds that $1 \leq |E_x| \leq 2^{|x|} - 2$.

Before proceeding further, we want to prove a useful lemma.

Lemma 9. *Let $x_1, x_2, y \in \{0,1\}^n$. If $x_2 \in D_{(2,3)}$ and $x_1 \neq x_2$, then there is no position i such that $|x_1| \leq i \leq |x_1\#x_1^R\#|$ and $\gamma_{i,x_2}^{(x_1)} = \gamma_{i,y}^{(x_2)}$.*

Proof. Assume that such a position i actually exists. We then swap between substrings $x_1\#(x_1^R)_j$ and $x_2\#(x_2^R)_j$, where $j = |x_1\#x_1^R\#| - i$, and we then obtain another accepting $(1,2)$-computation path on input $x_2\#(x_2^R)_j(x_1^R)_{n-j}\#x_2$. (Case 1) If $(x_2^R)_j(x_1^R)_{n-j} \neq x_2^R$, then such an accepting path cannot be a $(1,2)$-computation path, a contradiction. (Case 2) If $(x_2^R)_j(x_1^R)_{n-j} = x_2^R$, then the obtained accepting $(1,2)$-computation path on $x_2\#x_2^R\#x_2$ must be a rejecting path by the choice of $x_2 \in D_{(2,3)}$, a contradiction. □

3.5 Size of Stack Contents

Notice that $|E_x| \geq 1$ holds for all $x \in D_{(2,3)}$. Prior to a discussion on this general case, we intend to consider a special case, which exemplifies an essence of our proof, where $|E_x| = 1$ holds for any $x \in D_{(2,3)}$.

I) Special Case of $|E_x| = 1$. Since the choice of $y \in H_x$ is irrelevant, it is possible to drop subscript "y" and express $\gamma_{i,y}^{(x)}$, $\gamma_y^{(x)}$, and $u_{x,y}$, as $\gamma_i^{(x)}$, $\gamma^{(x)}$, and u_x, respectively. To lead to the desired contradiction, let us examine two stack contents, $\gamma_{|x\#|}^{(x)}$ and $\gamma^{(x)}$.

Proposition 4. *Given any number $\epsilon \geq 0$, it holds that $|\{x \in D_{(2,3)} \mid \exists y \in H_x\, [|\gamma_y^{(x)}| \geq (n-2-\epsilon)/\log|\Gamma'_{(1,2)}|]\}| \geq |D_{(2,3)}|(1-2^{-\epsilon})$.*

To prove Proposition 4, let us consider two stack contents $\gamma_{x_2}^{(x_1)}$ and $\gamma_{x_1}^{(x_2)}$ for any distinct pair $x_1, x_2 \in D_{(2,3)}$. Lemma 9 implies that $\gamma_{x_2}^{(x_1)} \neq \gamma_{x_1}^{(x_2)}$. We thus obtain the following.

Lemma 10. *For every distinct pair $x_1, x_2 \in D_{(2,3)}$, it holds that $\gamma^{(x_1)} \neq \gamma^{(x_2)}$.*

Recall the set $\Gamma'_{(1,2)} = \Gamma_{(1,2)} \cup \{Z_0\}$. Given a number $d \in \mathbb{N}^+$, we further define $A_d = \{x \in D_{(2,3)} \mid \exists y \in H_x\, [|\gamma_y^{(x)}| \geq d]\}$.

Lemma 11. *For any constant $d \in \mathbb{N}^+$, it holds that $|A_d| \geq |D_{(2,3)}| - 2|\Gamma'_{(1,2)}|^d$.*

Proof. Let $B_d = \{x \in D_{(2,3)} \mid \forall y \in H_x\, [|\gamma_y^{(x)}| < d]\}$. Notice that B_d coincides with $\{x \in D_{(2,3)} \mid |\gamma^{(x)}| < d\}$. It holds that $\gamma^{(x)}$ belongs to $(\Gamma'_{(1,2)})^m$ for a certain number m with $m \leq d-1$. Consider a mapping h from x to $\gamma^{(x)}$. Let $\bar{B}_d = \{x \in B_d \mid x^R = x\}$. The function h is 1-to-1 on $\bar{B}_d$ and also 1-to-1 on at least a half of elements in $B_d - \bar{B}_d$ by Lemma 10. Hence, it follows that

$|B_d|/2 \leq \sum_{j=1}^{d-1} |\Gamma'_{(1,2)}|^j = |\Gamma'_{(1,2)}|^d$. We conclude that, since $D_{(2,3)} = A_d \cup B_d$, $|A_d| = |D_{(2,3)}| - |B_d| \geq |D_{(2,3)}| - 2|\Gamma'_{(2,3)}|^d$, as requested. □

With the help of Lemma 11, Proposition 4 can be easily proven as follows.

Proof of Proposition 4. For simplicity, write d for $(n - 2 - \epsilon)/\log|\Gamma'_{(1,2)}|$, which equals $\log_{|\Gamma'_{(1,2)}|} 2^{n-2-\epsilon}$. It suffices to show that $|A_d| \geq |D_{(2,3)}|(1 - 2^{-\epsilon})$. By Lemma 11, we obtain $|A_d| \geq |D_{(2,3)}| - 2|\Gamma'_{(1,2)}|^d \geq |D_{(2,3)}|(1 - 2^{-\epsilon})$. □

To complete the proof for the special case, let $x = rz$, $x^R = z^R sr'$, $\gamma^{(x)} = uvZ_0$, and $\gamma_\ell^{(x)} = \sigma v Z_0$ with $\ell = |x\#z^R s|$. Assume that M transforms σ to u while reading r'. Proposition 3 shows that, for most of x's, $|uvZ_0|$ is upper-bounded by a certain constant, independent of (n, x, y). However, by setting, e.g., $\epsilon = 98$, Proposition 4 implies that $|uvZ_0| \geq (n - 100)/\log|\Gamma'_{(1,2)}|$ for at least the 2/3-fraction of x's in $D_{(2,3)}$. Since n is sufficiently large, we obtain a clear contradiction.

II) General Case of $|E_x| \geq 1$. We have already shown how to deal with the case where $|E_x| = 1$ holds for all $x \in D_{(2,3)}$. Now, let us discuss a general case where $|E_x| \geq 1$ holds for any $x \in D_{(2,3)}$. Our goal is to show the following statement.

Proposition 5. *There are at least the $|D_{(2,3)}|^{-1/2}$-fraction of x's in $D_{(2,3)}$ such that, for a certain stack content $\tau \in E_x$, τ contains at least $\log_{|\Gamma|} n/2$ symbols.*

We start with the following lemma regarding E_x's, which can be seen as a generalization of Lemma 10.

Lemma 12. *Let $x_1, x_2 \in D_{(2,3)}$. If $x_2 \in H_{x_1}$, then $E_{x_1} \neq E_{x_2}$.*

Proof. Assume to the contrary that $E_{x_1} = E_{x_2}$ holds for two particular elements $x_1, x_2 \in D_{(2,3)}$ satisfying $x_2 \in H_{x_1}$. Take a stack content $\tau \in E_{x_1}$ satisfying $\tau = \gamma_{x_2}^{(x_1)}$ for a certain accepting $(1,2)$-computation path p_{x_1,x_2} of M on $x_1\#x_1^R\#x_2$. Since $E_{x_1} = E_{x_2}$, there exists another y in H_{x_2} that satisfies $\tau = \gamma_y^{(x_2)}$ along an appropriate accepting $(1,2)$-computation path $p_{x_2,y}$ on $x_2\#x_2^R\#y$. By swapping two parts of the above computation paths p_{x_1,x_2} and $p_{x_2,y}$ properly, we then obtain another accepting $(1,2)$-computation path of M on $x_2\#x_2^R\#x_2$ satisfying $\tau = \gamma_{x_2}^{(x_2)}$. This is an obvious contradiction against the choice of $x_2 \in D_{(2,3)}$. □

Write U_n for $\{x \in D_{(2,3)} \mid |E_x| > n/2\}$ and consider two separate cases.

Case 1: Assume that $|U_n| \geq |D_{(2,3)}|^{1/2}$. By taking an arbitrary $x \in U_n$, we want to claim that a certain stack content $\tau \in E_x$ must be made up of more than $\log_{|\Gamma|} n/2$ symbols. For this purpose, let us assume otherwise. Since any τ in E_x has at most $\log_{|\Gamma|} n/2$ symbols, there must be at most $n/2$ different elements in E_x. This implies that $x \notin U_n$, a contradiction against the choice of x. Hence, we obtain $|\tau| > \log_{|\Gamma|} n/2$, as stated in Proposition 5.

Case 2: Next, we assume that $|U_n| < |D_{(2,3)}|^{1/2}$. We first prove the following combinatorial lemma.

Lemma 13. *Let $n \in \mathbb{N}^+$ be sufficiently large and let X, Y satisfy $X \subseteq Y$. Let A be an $X \times Y$ matrix whose entries are taken from Θ^*, where Θ is an alphabet. Assume that (i) $|X| \geq 2^{n-2}$ and $|Y| = 2^n$, (ii) for any $(x, y) \in X \times Y$, $A_{(x,y)} = \lambda$ iff $y \in \{x, x^R\}$, and (iii) for any $x, y \in X$, if $A_{(x,y)} \neq \lambda$, then $A_{(x,y)} \neq A_{(y,z)}$ for any $z \in Y$. Then, the set $\tilde{X} = \{x \in X \mid \exists y \in Y\, [|A_{(x,y)}| \geq \log_{|\Theta|} \log_{|\Theta|} n\,]\}$ has cardinality at least $|X|^{1/2}$.*

Proof. Let us assume that the premise of the lemma is satisfied. For convenience, we define $X' = \{x \in X \mid \max_{y \in Y}\{|A_{(x,y)}|\} < \log_{|\Theta|} \log_{|\Theta|} n\}$, which satisfies $X = \tilde{X} \cup X'$. To show that $|\tilde{X}| \geq |X|/2$, we assume to the contrary that $|\tilde{X}| < |X|/2$. This implies that $|X'| = |X| - |\tilde{X}| > |X| - |X|/2 = |X|/2 \geq 2^{n-3}$ since $|X| \geq 2^{n-2}$. Let $E'_x = \{A_{(x,y)} \mid y \in Y\}$ for every $x \in X$. Analogously to Lemma 12, it holds that $E'_{x_1} \neq E'_{x_2}$ for every distinct pair $x_1, x_2 \in X$.

Let $x \in X'$. Since $|A_{(x,y)}| < \log_{|\Theta|} \log_{|\Theta|} n$ for all $y \in Y$, the total number of strings $A_{(x,y)}$ in E'_x is upper-bounded by $|\Theta|^{\log_{|\Theta|} \log_{|\Theta|} n} = \log_{|\Theta|} n$; that is, $|E'_x| \leq \log_{|\Theta|} n$. For convenience, let $\mathcal{E} = \bigcup_{x \in X'} E'_x$ and set $\alpha = |\mathcal{E}|$. Notice that $\alpha \geq 2$. Hereafter, we want to claim that $\alpha \geq 2^{(n-3)/(2 \log_{|\Theta|} n)}$. Toward a contradiction, we assume that $\alpha < 2^{(n-3)/(2 \log_{|\Theta|} n)}$. Now, let us estimate the upper bound of $|X'|$. Note that there are $|X'|$ different E'_x's in $\mathcal{E}$ and that E'_x is a subset of $\mathcal{E}$ of cardinality at most $\log_{|\Theta|} n$. It follows that $|X'|$ does not exceed the total number of $\mathcal{E}$'s nonempty subsets of size at most $\log_{|\Theta|} n$. We then conclude that $|X'| \leq \sum_{i=1}^{\log_{|\Theta|} n} \binom{\alpha}{i} \leq (\log_{|\Theta|} n) \cdot \alpha^{\log_{|\Theta|} n} \leq \alpha^{2 \log_{|\Theta|} n} \leq 2^{n-3}$, where the second inequality comes from $i < n/2$ and $\binom{\alpha}{i} \leq \alpha^i / i!$. This is a clear contradiction against $|X'| \geq 2^{n-3}$. Therefore, we obtain $\alpha \geq 2^{(n-3)/(2 \log_{|\Theta|} n)}$.

However, this contradicts the bound of $|\mathcal{E}| \leq \log_{|\Theta|} n$. □

Let us return to the proof of Proposition 5. To apply Lemma 13, we simply set X to be $D_{(2,3)}$, $\{0,1\}^n$ to be Y, and $\tilde{\gamma}_y^{(x)}$ to be $A_{(x,y)}$, where $\tilde{\gamma}_y^{(x)}$ is obtained from $\gamma_y^{(x)}$ by simply deleting Z_0. It is not difficult to show that the obtained triplet (A, X, Y) satisfies Conditions (i)–(iii) of the lemma. The lemma ensures that there are at least $2^{n/3}$ x's in $D_{(2,3)}$ satisfying $|\gamma_y^{(x)}| \geq \log_{|\Theta|} \log_{|\Theta|} n$ for a certain string $y \in \{0,1\}^n$.

Nonetheless, Proposition 3 indicates that $|\gamma_y^{(x)}| \leq d_1$ for all $y \in H_x$. We then obtain a contradiction, as requested, and therefore this closes Case 1.

3.6 Case 2: $D_{(1,2)}$ is Large

We have already proven Case 1 in Sections 3.3–3.5. To complete the proof of Proposition 2, however, we still need to examine the remaining case where $\{n \in \mathbb{N}^+ \mid |D_{(2,3)}| \geq 2^n/2\}$ is a finite set; in other words, $|D_{(1,2)}| > 2^n/2$ holds for all but finitely many n. Recall from Section 3.2 the introduction of our colored

automaton $M = (Q, \Sigma, \{\text{¢}, \$\}, \Gamma, I_3, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$ with $Q_{acc} = \{q_{acc}\}$ and $Q_{rej} = \{q_{rej}\}$ that computes g. Before starting the intended proof, we will present a general lemma regarding inputs in reverse form.

Lemma 14. *There exists a colored automaton M^R that satisfies the following: M accepts $x_1\#x_2\#x_3$ along an accepting (i,j)-computation path if and only if M^R accepts $x_3^R\#x_2^R\#x_1^R$ along an accepting $(4-j, 4-i)$-computation path.*

Let us return to our proof for the case of $|D_{(1,2)}| > 2^n/2$. Note that, by running M on inputs of the form $x\#y\#z$ for $x, y, z \in \{0,1\}^n$, we then obtain $|D_{(1,2)}| > 2^n/2$. We consider a counterpart of $D_{(1,2)}$, denoted by $D^R_{(2,3)}$, which is obtained by running M^R instead of M. Lemma 14 also implies that $|D^R_{(2,3)}| > 2^n/2$. Apply to $D^R_{(2,3)}$ an argument used for Case 1. This is an obvious contradiction. We have therefore completed the proof of Proposition 2.

References

1. C. Choffrut and K. Culik. Properties of finite and pushdown transducers. *SIAM J. Comput.*, 12 (1983) 300–315.
2. R. J. Evey. Application of pushdown-store machines. In *Proc. 1963 Fall Joint Computer Conference*, AFIPS Press, pp.215–227, 1963.
3. P. C. Fisher. On computability by certain classes of restricted Turing machines. In *Proc. 4th Annual IEEE Symp. on Switching Circuit Theory and Logical Design* (SWCT'63), IEEE Computer Society, pp.23–32, 1963.
4. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Second Edittion. Addison-Wesley, 2001.
5. K. Kobayashi. Classification of formal langauges by functional binary transductions. *Inform. Control*, 15 (1969) 95–109.
6. S. Konstantinidis, N. Santean, and S. Yu. Representation and uniformization of algebraic transductions. *Acta Inform.*, 43 (2007) 395–417.
7. A. L. Selman. A taxonomy of complexity classes of functions. *J. Comput. System Sci.*, 48 (1994) 357–381.
8. A. L. Selman. Much ado about functions. In *Proc. of the 11th Annual IEEE Conference on Computational Complexity*, pp.198–212, 1996.
9. K. Tadaki, T. Yamakami, and J. C. H. Lin. Theory of one-tape linear-time Turing machines. *Theoret. Comput. Sci.*, 411 (2010) 22–43. An extended abstract appeared in SOFSEM 2004, LNCS vol.2932, pp.335–348, 2004.
10. T. Yamakami. Swapping lemmas for regular and context-free languages. Available at arXiv:0808.4122, 2008.
11. T. Yamakami. Pseudorandom generators against advised context-free languages. See arXiv:0902.2774, 2009.
12. T. Yamakami. Immunity and pseudorandomness of context-free languages. *Theor. Comput. Sci.*, 412 (2011) 6432–6450.
13. T. Yamakami. Oracle pushdown automata, nondeterministic reducibilities, and the hierarchy over the family of context-free languages. In *Proc. 40th of SOFSEM 2014*, LNCS, vol. 8327, pp. 514–525, 2014. See also arXiv:1303.1717.
14. T. Yamakami. Structural complexity of multi-valued partial functions computed by nondeterministic pushdown automata. Unpublished manuscript, 2014.