Vol.2012-AL-142 No.4
2012/11/2

# Nondeterministic Pushdown Automata with Write-Only Output Tapes and Definable Function Classes

***(Preliminary Report)***

Tomoyuki Yamakami

**Abstract:** This paper reports preliminary results obtained from a systematic study on the behaviors of multi-valued partial CFL functions, which are computed under various constraints by one-way nondeterministic pushdown automata equipped with additional write-only output tapes. The CFL functions tend to behave quite differently from their corresponding context-free languages. We show several containments and separations among natural classes of CFL functions. We also analyze the computational complexity of languages having properties of selectivity, in which strings (or words) in the languages are selected by appropriate CFL functions. A notion of many-one reducibility introduces relativized CFL functions, which form a hierarchy of CFL function classes. To separate function classes in this hierarchy, we further examine the roles of special oracles that compute multiple values. For a finer analysis, we consider restricted CFL functions having at most $k$ distinct output values and also functions having only values that $k$ CFL functions simultaneously output.

**Keywords:** pushdown automaton, oracle, many-one reduction, truth-table reduction, polynomial hierarchy, selective set, low set, advice, regular language, context-free language

## 1. Much Ado about Functions

In a traditional field of formal languages and automata, we have dealt primarily with *languages*, including now-basic regular languages and context-free languages, whose words (or strings) are described both in the means of generation by grammars and of acceptance by automata. In most literature, languages have been a center piece of intensive research because of their practical applications to, for instance, a parsing analysis of programming languages. By providing additional output tapes, automata can be thought as a mechanism of transforming input words to output words, and this mechanism has paved a general way to *(multi-valued partial) functions* in the study of formal languages and automata. From a slightly different viewpoint, languages can be naturally regarded as $\{0,1\}$-valued (or Boolean-valued) total functions, by assigning 1 (or TRUE) to "acceptance" and 0 (or FALSE) to "rejection" of each input instance given to the languages. Nevertheless, when outcomes of functions become more general than just $\{0,1\}$, those functions demand quite different technical tools and proof arguments, compared to those for corresponding languages. For instance, unlike language families, a containment between two multi-valued partial functions is customarily replaced by a notion of *refinement*, rather than a standard set inclusion.

In recent literature, a group of meaningful function classes has been introduced to formal languages and automata theory and it has exhibited its usefulness in an exploration of structural complexity of languages. In 2004, two one-tape linear-time function classes 1-FLIN and 1-FLIN(partial) were introduced by Tadaki, Yamakami, and Li [13], who proved the non-existence of length-preserving one-way functions in 1-FLIN, where the term "partial" in FL(partial) refers to partial functions). Those function classes are based on deterministic computations and thus each function produces at most one outcome per each input as a consequence of the deterministic procedure. On the contrary, nondeterministic computations may possibly produce numerous distinct output values (including the case of no output values), and functions computed by such computations become, in general, *multi-valued partial functions*. Four associated 1-NLIN function classes 1-NLINMV, 1-NLINMV$_t$, 1-NLINSV, and 1-NLINSV$_t$ were introduced also in [13] (where MV and SV respectively stand for "multi-valued" and "single-valued" and the subscript "t" does for "total").

Using a model of nondeterministic pushdown automaton (or npda, in short) equipped with one-way read-only input tapes as well as write-only output tapes, additional three CFL function classes CFLMV, CFLSV, and CFLSV$_t$ were defined and discussed first in [17] and explored in [18]. Those functions are also inherently multi-valued partial functions mapping every string to a certain finite set of strings. It was shown in [17] that there exists a pseudorandom generator in CFLSV$_t$ that fool every language in REG$/n$, which is a non-uniform version of the family REG of regular languages, defined in [13] and later characterized in [16] in terms of machine-independent equivalence classes. Another function class CFL(2)MV appeared in [18] in connection to pseudorandom generators against CFL$/n$, which is a non-

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

uniform analogue of the family CFL of context-free languages in [15]. The behaviors of functions in those function classes seem to look quite different from what we have known for context-free languages. For instance, the single-valued total function class $\mathrm{CFLSV_t}$ can be seen as a functional extension of the language family CFL ∩ co-CFL, rather than CFL alone [18]. In stark contrast with a tidy theory of NP functions (see a survey [10]), circumstances that surround CFL functions are rather in disarray mainly because of mechanical constraints (e.g., a use of stacks, one-way moves of tape heads, and $\lambda$-moves) that harness the behaviors of underlying npda's.

What remains needed is a coherent and systematic study on the structural behaviors of multi-valued partial functions. As a follow-up to the early study, this paper aims at exploring fundamental relationships among the aforementioned function classes and their natural extensions via various notions of reducibility. Earlier, a limited notion of *many-one reducibility* among languages as well as functions was introduced in [13] using a model of one-tape linear-time computation. Here, we adapt this notation to accommodate our model of npda's. The many-one reducibility helps us introduce more general function classes $\Sigma_k^{\mathrm{CFL}}\mathrm{MV}$ and $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}$ for each level $k \geq 1$, where those in the first level respectively coincide with CFLMV and CFLSV. We are mostly focused on containments and separations of those functions classes. In particular, in order to discuss separations among the function classes, it is useful to consider special oracles that compute multiple values. A more general treatment of such oracles was initiated by Fenner, Homer, Ogihara, and Selman [2]. Later, we expand the many-one reducibility to *k-truth-table reducibility*, where $k$ is a fixed positive integer. An accompanied notion of *many-one lowness* given in [13] is also extended into *truth-table lowness*.

Polynomial-time computable functions play a special role in a notion of P-selective sets, which was first introduced into complexity theory by Selman [8] in 1979 as an analogue of semi-recursive sets in recursion theory. Later, this notion was extended to NPSV-selective sets [3] and beyond. We discuss a similar notion under a specific requirement that all input instances are of the same length. To emphasize this length requirement, we coin the term "lengthwise $\mathcal{F}$-selective languages."

This paper reports merely preliminary results on CFL functions, and a full report on the same subject will appear shortly in a different medium.

## 2. A Starting Point

We will briefly explain existing notions and notations that mark a starting point of the subsequent sections.

### 2.1 Alphabets, Strings, and Languages

Given a finite set $A$, the notation $\|A\|$ expresses the number of elements in $A$. Let $\mathbb{N}$ be the set of all *natural numbers* (i.e., non-negative integers) and set $\mathbb{N}^+ = \mathbb{N} - \{0\}$. Throughout this paper, we use the term " polynomial" to mean polynomials on $\mathbb{N}$ with non-negative coefficients. In particular, a *linear polynomial* is of the form $ax + b$ with $a, b \in \mathbb{N}$. The notation $A - B$ for two sets $A$ and $B$ indicates the *difference* $\{x \mid x \in A, x \notin B\}$. Given a set $A$, $\mathcal{P}(A)$ denotes the *power set* of $A$.

An *alphabet* is a nonempty finite set $\Sigma$ and its elements are called *symbols*. A *string* $x$ over $\Sigma$ is a finite series of symbols chosen from $\Sigma$ and its *length*, denoted $|x|$, is the total number of symbols in $x$. The *empty string* $\lambda$ is a special string whose length is zero. A collection of strings over $\Sigma$ is called a *language* over $\Sigma$. A set $\Sigma^k$, where $k \in \mathbb{N}$, consists only of strings of length $k$. In particular, $\Sigma^0 = \{\lambda\}$. Here, we set $\Sigma^* = \bigcup_{k\in\mathbb{N}} \Sigma^k$. Given a language $A$ over $\Sigma$, its *complement* is $\Sigma^* - A$, which is also denoted by $\overline{A}$ as long as $\Sigma$ is clear from the context. In general, for two language families $\mathcal{C}_1$ and $\mathcal{C}_2$ over the same alphabet, let $\mathcal{C}_1 \wedge \mathcal{C}_2 = \{A \cap B \mid A \in \mathcal{C}_1, B \in \mathcal{C}_2\}$, $\mathcal{C}_1 \vee \mathcal{C}_2 = \{A \cup B \mid A \in \mathcal{C}_1, B \in \mathcal{C}_2\}$, and $\mathcal{C}_1 - \mathcal{C}_2 = \{A - B \mid A \in \mathcal{C}_1, B \in \mathcal{C}_2\}$.

To treat a tuple of strings, we adopt a *track notation* $[\,{x \atop y}\,]$ of [13]. For two symbols $\sigma$ and $\tau$, $[\,{\sigma \atop \tau}\,]$ expresses a new symbol. For two strings $x = x_1x_2\cdots x_n$ and $y = y_1y_2\cdots y_n$ of length $n$, $[\,{x \atop y}\,]$ denotes a string $[\,{x_1 \atop y_1}\,][\,{x_2 \atop y_2}\,]\cdots[\,{x_n \atop y_n}\,]$. Whenever $|x| \neq |y|$, we follow a convention introduced in [13]: if $|x| < |y|$, then $[\,{x \atop y}\,]$ actually means $[\,{x\#^m \atop y}\,]$, where $m = |y| - |x|$ and # is a designated new symbol. Similarly, when $|x| > |y|$, the notation $[\,{x \atop y}\,]$ expresses $[\,{x \atop y\#^m}\,]$ with $m = |x| - |y|$.

Given a language $A$, the *characteristic function* for $A$, denoted $\chi^A$, is a function defined as $\chi^A(x) = 1$ if $x \in A$ and $\chi^A(x) = 0$ otherwise. We expand this notion to a *k-ary characteristic function* $\chi_k^A$ for each number $k \geq 1$. We write $\chi_k^A(x_1, x_2, \ldots, x_k)$ as a $k$-bit string $\chi^A(x_1)\chi^A(x_2)\cdots\chi^A(x_k)$ in $\{0,1\}^k$. For convenience sake, we also write $(\chi^A(x_1), \chi^A(x_2), \ldots, \chi^A(x_k))$ to denote this string.

As our basic computation models, we use *one-way deterministic finite automata* (or dfa's, in short) with $\lambda$-moves, *one-way nondeterministic pushdown automata* (or npda's) with $\lambda$-moves, *two-way deterministic Turing machines* (or DTMs), and *two-way nondeterministic Turing machines* (or NTMs). For any of those machines $M$, we write $PATH_M(x)$ to express a collection of all computation paths produced by $M$ on input $x$ and we use $ACC_M(x)$ to denote a set of all accepting computation paths of $M$ on input $x$. Moreover, for each length $n \in \mathbb{N}$, let $ACC_{M,n}$ denote a set $\bigcup_{x:|x|=n} ACC_M(x)$. Whenever $M$ is clear from the context, we often drop the subscript "$M$" from $PATH_M(x)$, $ACC_M(x)$ and $ACC_{M,n}$.

Whenever we refer to a *write-only tape*, we always assume that (i) initially, all cells of the tape are blank, (ii) a tape head starts at the so-called *start cell*, (iii) the tape head steps forward whenever it write down any non-blank symbol, and (iv) the tape head can stay still only in a blank cell. For convenience, we assume that any cells of the output tape from the start cell to any cells that its tape head passes during a computation must contain no blank symbols. An *output* (outcome or output string) along a computation path is a string produced on the output tape after the computation path is terminated. We call an output string *valid* (or legitimate) if it is produced along a certain accepting computation path. When we refer to the machine's outputs, we normally disregard any strings left on the output tape on a rejecting computation path.

The notations REG and CFL stand for the families of all regular languages and of all context-free languages, respectively. For each number $k \in \mathbb{N}^+$, the *k-conjunctive closure* of CFL, denoted CFL($k$), is defined recursively as follows: CFL(1) = CFL and CFL($k+1$) = CFL($k$) ∧ CFL. See, e.g., [17]. An advised language family REG$/n$ consists of languages $L$ such that there exist an advice alphabet $\Gamma$, a length-preserving (total) advice function $h : \mathbb{N} \to \Gamma^*$, and a language $A \in$ REG satisfying $L = \{x \mid [\,^{x}_{h(|x|)}] \in A\}$ [13], where $h$ is *length preserving* if $|h(n)| = n$ for all numbers $n \in \mathbb{N}$. By replacing REG with CFL or CFL(2), we can define CFL$/n$ [15] and CFL(2)$/n$ [18], respectively. It is known that CFL $\nsubseteq$ REG$/n$ [13], co-CFL $\nsubseteq$ CFL$/n$ [15], and CFL(2) $\nsubseteq$ CFL$/n$ [18]. The *Boolean hierarchy over CFL* was introduced in [19] by setting $\mathrm{CFL}_1 = \mathrm{CFL}$, $\mathrm{CFL}_{2k} = \mathrm{CFL}_{2k-1} \wedge$ co-CFL, and $\mathrm{CFL}_{2k+1} = \mathrm{CFL}_{2k} \vee \mathrm{CFL}$.

Moreover, let P (resp., NP) be composed of all languages recognized by DTMs (resp., NTMs) in polynomial time. Given each index $k \in \mathbb{N}$, we define $\Sigma^{\mathrm{P}}_0 = \Pi^{\mathrm{P}}_0 = \mathrm{P}$, $\Sigma^{\mathrm{P}}_{k+1} = \mathrm{NP}^{\Sigma^{\mathrm{P}}_k}$, and $\Pi^{\mathrm{P}}_{k+1} = \text{co-}\Sigma^{\mathrm{P}}_{k+1}$, where $\mathrm{NP}^{C} = \bigcup_{A\in C} \mathrm{NP}^{A}$ and $\mathrm{NP}^{A}$ is the family of languages recognized by polynomial-time NTMs with adaptive queries to $A$ as an oracle. Those language families constitute the so-called *polynomial(-time) hierarchy* [6]. Let $\mathrm{PH} = \bigcup_{k\in\mathbb{N}} \Sigma^{\mathrm{P}}_k$. We denote by L a family of languages recognized by DTMs with a two-way read-only input tape and a two-way read/write work tape using $O(\log n)$ cells of the work tape.

Let $\mathrm{SAC}_1$ be a class of languages whose characteristic functions are computed by logarithmic-space uniform families of polynomial-size Boolean circuits of $O(\log n)$ depth and semi-bounded fan-in (that is, having *AND* gates of bounded fan-in and *OR* gates of unbounded fan-in), provided that the negations appear only at the input level.

### 2.2 Functions and Refinement

We will discuss terminology associated with *multi-valued partial functions*. Throughout this paper, we will adopt the following convention: the generic term "function" refers to "multi-valued partial function," provided that *single-valued* functions are viewed as a special case of multi-valued functions and partial functions include *total* functions. We are mostly interested in multi-valued partial functions mapping[*] $\Sigma^*$ to $\mathcal{P}(\Gamma^*)$ for certain alphabets $\Sigma$ and $\Gamma$. When $f$ is single-valued, we often write $f(x) = y$ instead of $y \in f(x)$. Associated with $f$, dom$(f)$ denotes the *domain* of $f$ defined as $\mathrm{dom}(f) = \{x \in \Sigma^* \mid f(x) \neq \emptyset\}$. If $x \notin \mathrm{dom}(x)$, $f(x)$ is said to be *undefined*.

Concerning all function classes discussed in this paper, it is natural to concentrate only on functions whose outcomes are bounded in size by fixed polynomials. More precisely, a multi-valued partial function $f : \Sigma^* \to \mathcal{P}(\Gamma^*)$ is called *polynomially bounded* (resp., *linearly bounded*) if there exists a nonnegative polynomial $p$ (resp., two positive constants $c, d$) such that, for any two strings $x \in \Sigma^*$ and $y \in \Gamma^*$, if $y \in f(x)$ then $|y| \leq p(|x|)$ (resp., $|y| \leq c|x| + d$) holds. In this paper, we understand that all function classes are made of polynomially-bounded functions. Given two alphabets $\Sigma$ and $\Gamma$, a multi-valued partial function $f : \Sigma^* \to \mathcal{P}(\Gamma^*)$ is called *length preserving* if, for any two strings $x \in \Sigma^*$ and $y \in \Gamma$, $y \in f(x)$ implies $|x| = |y|$.

A function class CFLMV is composed of all multi-valued partial functions $f$, each of which maps $\Sigma^*$ to $\Sigma^*$ for a certain alphabet $\Sigma$ and there exists an npda $N$ with a one-way read-only input tape and a write-only output tape such that, for every input $x \in \Sigma^*$, $f(x)$ is the set of all *valid* outcomes of $N$ on the input $x$. Another class CFLSV is a subclass of CFLMV consisting of single-valued partial functions. In addition, $\mathrm{CFLMV}_{\mathrm{t}}$ and $\mathrm{CFLSV}_{\mathrm{t}}$ are respectively restrictions of CFLMV and CFLSV onto *total functions*. Those function classes were introduced in [17]. It was shown in [18] that $\mathrm{CFL} \cap \text{co-CFL} = \{A \mid \chi_A \in \mathrm{CFLSV}_{\mathrm{t}}\}$. A function class CFL(2)MV contains all multi-valued partial functions $f$ such that there are two functions $g_1, g_2 \in$ CFLMV satisfying $f(x) = g_1(x) \cap g_2(x)$ for any string $x$ [18].

An important concept of this paper is *refinement*. Given two multi-valued partial functions $f$ and $g$, we say that $f$ is a *refinement* of $g$, denoted $g \sqsubseteq_{ref} f$, if (1) $\mathrm{dom}(f) = \mathrm{dom}(g)$ and (2) for every input $x$, $f(x) \subseteq g(x)$ (as a set inclusion). For two sets $\mathcal{F}$ and $\mathcal{G}$ of functions, $\mathcal{G} \sqsubseteq_{ref} \mathcal{F}$ if, for every function $g \in \mathcal{G}$, there exists a function $f \in \mathcal{F}$ such that $g \sqsubseteq_{ref} f$. When $f$ is additionally single-valued, we often call $f$ a *single-valued refinement* of $g$.

Like CFL$/n$, an advised function class CFLMV$/n$ is composed of all functions $f$ such that there exist an advice alphabet $\Gamma$, a length-preserving (total) advice function $h : \mathbb{N} \to \Gamma^*$, and a function $g \in$ CFLMV satisfying $f(x) = g([\,^{x}_{h(|x|)}])$ for every input $x$ [17]. Note that $\mathrm{CFL}/n \cap \text{co-CFL}/n = \{A \mid \chi_A \in \mathrm{CFLMV}/n\}$ [18]. Similarly, CFL(2)MV$/n$ is defined in [18] using CFL(2)MV in place of CFLMV.

The notation FL denotes a collection of polynomially-bounded single-valued total functions computed by two-way DTMs with a read-only input tape, a read/write work tape, and a write-only output tape using $O(\log n)$ work space. If we allow DTMs to enter rejecting states, in which any output value is automatically invalidated, those DTMs can naturally compute *partial functions*. In this way, we can obtain FL(partial) from FL by permitting underlying DTMs to compute partial functions. For the precise definitions of a one-tape linear-time deterministic complexity class 1-DLIN and its functional versions 1-FLIN and 1-FLIN(partial), refer to [13].

## 3. Many-One and Truth-Table Reductions

A notion of *many-one relativization* of language families as well as counting function classes was introduced into one-tape linear-time computations in [13]. In a similar fashion, we will define a language family $\mathrm{CFL}^{A}_{m}$ for a given language $A$ over an alphabet $\Gamma$, where the language $A$ is customarily called an *oracle*. A language $L$ over an alphabet $\Sigma$ belongs to $\mathrm{CFL}^{A}_{m}$ if there exists an npda $M$ with a write-only output tape using an input alphabet $\Sigma$ and an output alphabet $\Gamma$ such that, for every input $x \in \Sigma^*$, (1) along each computation path $p \in ACC_M(x)$, $M$ produces a valid output string $y_p \in \Gamma^*$ and (2) $x \in L$ iff $y_p \in A$ for an appropriate computation path $p \in ACC_M(x)$. This $M$ is often called a *reduction machine*. For a language family $C$, let $\mathrm{CFL}^{C}_{m} = \bigcup_{A\in C} \mathrm{CFL}^{A}_{m}$. Given a language $A$, a language $L$ is in $\mathrm{L}^{A}_{m}$ if there exists a logarithmic-space DTM $M$ with a write-only query tape such that, $x \in L$ iff $M(x)$ produces a string $y$ for which $y \in A$.

As a non-uniform extension of $\mathrm{CFL}^{A}_{m}$, we define $\mathrm{CFL}^{A}_{m}/n$; that

[*] To describe a multi-valued function $f$, the expression "$f : \Sigma^* \to \Gamma^*$" is customarily used in the literature. However, the current expression "$f : \Sigma^* \to \mathcal{P}(\Gamma^*)$" matches a fact that the outcome of $f$ on each input string in $\Sigma^*$ is a subset of $\Gamma^*$.

is, $L \in \mathrm{CFL}_m^A/n$ iff there exist a length-preserving advice function $h$ and an npda $M$ with a write-only output tape such that, for every input $x$, (i) $M([\,{}^{x}_{h(|x|)}])$ produces $y_p$ along every accepting path $p$ and (ii) $x \in L$ iff $y_p \in A$ for an appropriate path $p \in ACC_M([\,{}^{x}_{h(|x|)}])$.

***Lemma 3.1*** *For every index $k \geq 1$,* $\mathrm{CFL}(k+1) \subseteq \mathrm{CFL}_m^{\mathrm{CFL}(k)}$.

**Proof.** Let $L \in \mathrm{CFL}(k+1)$. Take two languages $L_1 \in \mathrm{CFL}$ and $L_2 \in \mathrm{CFL}(k)$ such that $L = L_1 \cap L_2$. There exists an npda $M_1$ that recognizes $L_1$. Without loss of generality, we assume that $M_1$ enters a final state (either an accepting state or a rejecting state) when it scans the right endmarker \$. Now, we define a new npda $N$ with a write-only output tape as follows. On input $x$, $N$ start simulating $M_1(x)$. While reading each symbol of $x$, $N$ copies it down to the output tape. When $M_1$ halts in a final state, $N$ enters the same inner state. Now, we take $L_2$ as an oracle. It holds that, for any input $x$, $x \in L$ iff $N(x)$ outputs $x$ in an accepting state and $x \in L_2$. This implies that $L$ belongs to $\mathrm{CFL}_m^{L_2}$, which is a subclass of $\mathrm{CFL}_m^{\mathrm{CFL}(k)}$. □

Using the many-one reductions, we define the *(many-one) CFL hierarchy* $\{\Sigma_k^{\mathrm{CFL}}, \Pi_k^{\mathrm{CFL}} \mid k \in \mathbb{N}\}$ recursively as follows: $\Sigma_0^{\mathrm{CFL}} = \Pi_0^{\mathrm{CFL}} = \mathrm{DCFL}$, $\Sigma_1^{\mathrm{CFL}} = \mathrm{CFL}$, $\Pi_k^{\mathrm{CFL}} = \text{co-}\Sigma_k^{\mathrm{CFL}}$, and $\Sigma_{k+1}^{\mathrm{CFL}} = \mathrm{CFL}_m^{\Pi_k^{\mathrm{CFL}}}$ for any $k \geq 1$. Reinhardt [7] discussed a similar hierarchy built on a special language called $L_{pp}$ using nondeterministic (as well as deterministic) finite automata with output tapes (which are called transducers therein) as reduction machines. We begin with a simple containment.

***Lemma 3.2*** $\mathrm{CFL}_2 \subseteq \Sigma_2^{\mathrm{CFL}}$.

***Lemma 3.3*** *For ever index $k \geq 2$, $\Sigma_2^{\mathrm{CFL}} \nsubseteq \mathrm{CFL}/n$ and $\Sigma_k^{\mathrm{CFL}} \subseteq \Sigma_{k+1}^{\mathrm{CFL}} \subseteq \mathrm{DSPACE}(O(n))$.*

**Proof.** For the first claim, note that $\text{co-CFL} \subseteq \Sigma_2^{\mathrm{CFL}}$. Since $\text{co-CFL} \nsubseteq \mathrm{CFL}/n$ [15], it follows that $\Sigma_2^{\mathrm{CFL}} \nsubseteq \mathrm{CFL}/n$ and thus $\mathrm{CFL} \neq \Sigma_2^{\mathrm{CFL}}$. It is not difficult to show the second claim. □

The CFL hierarchy becomes a useful tool because it is closely related to the polynomial hierarchy $\{\Sigma_k^{\mathrm{P}}, \Pi_k^{\mathrm{P}} \mid k \in \mathbb{N}\}$. Following an argument in [7], it is rather straightforward to establish the following connection between $\Sigma_{k+1}^{\mathrm{CFL}}$ and $\Sigma_k^{\mathrm{P}}$. Notice that $\mathrm{L}_m^{\mathrm{CFL}}$ is well-known as LOGCFL (LogCFL or LOG(CFL)) in the past literature. The first equation of the following lemma was proven by Venkateswaran [14].

***Lemma 3.4*** $\mathrm{L}_m^{\mathrm{CFL}} = \mathrm{SAC}_1$ *and* $\mathrm{L}_m^{\Sigma_{k+1}^{\mathrm{CFL}}} = \Sigma_k^{\mathrm{P}}$ *for every index $k \geq 1$.*

Since the many-one reductions are quite restrictive, we want to expand the type of relativization from "many-one" to "$k$-truth table." A language $L$ is in $\mathrm{CFL}_{ktt}^A$ if there are a language $B \in \mathrm{REG}$ and an npda $N$ with $k$ write-only output tapes besides a read-only input tape such that (1) $ACC_N(x) \neq \emptyset$ for all inputs $x$, (2) along a computation path $p \in ACC_N(x)$, on the $i$th output tape, $N$ produces a string $y_p^{(i)} \in \Gamma^*$, (3) from a vector $(y_p^{(1)}, y_p^{(2)}, \ldots, y_p^{(k)})$ of query words, we generate a $k$-bit string $z_p = \chi_k^A(y_p^{(1)}, y_p^{(2)}, \ldots, y_p^{(k)}) \in \{0,1\}^k$, and (4) $x \in L$ iff $[\,{}^{x}_{z_p}] \in B$ for an appropriate computation path $p \in ACC_N(x)$. Here, the set $B$ is called a *truth table* for $A$. In addition, we set $\mathrm{CFL}_{btt}^A$ to be $\bigcup_{k \in \mathbb{N}^+} \mathrm{CFL}_{ktt}^A$.

***Lemma 3.5*** *For every language $A$ and index $k \geq 1$,* $\mathrm{CFL}_m^A \cup \mathrm{CFL}_m^{\overline{A}} \subseteq \mathrm{CFL}_{ktt}^A = \mathrm{CFL}_{ktt}^{\overline{A}}$.

**Proof.** It is clear that $\mathrm{CFL}_m^A \cup \mathrm{CFL}_m^{\overline{A}} \subseteq \mathrm{CFL}_{1tt}^A$. Now, we show that $\mathrm{CFL}_{ktt}^A \subseteq \mathrm{CFL}_{ktt}^{\overline{A}}$. This is done by flipping the outcome of the last computation by dfa's. In symmetry, $\mathrm{CFL}_{ktt}^{\overline{A}} \subseteq \mathrm{CFL}_{ktt}^A$ also holds. □

When playing oracles, languages in REG have no power to increase the complexity of relativized CFL.

***Lemma 3.6*** $\mathrm{CFL}_{btt}^{\mathrm{REG}} = \mathrm{CFL}_m^{\mathrm{REG}} = \mathrm{CFL}$.

Note that $\text{1-DLIN}_m^C$ was introduced in [13]. In a similar way, we can introduce $\text{1-DLIN}_{ktt}^C$.

***Lemma 3.7*** *For every fixed $k \in \mathbb{N}^+$,* $\mathrm{CFL}_k \cup \text{co-CFL}_k \subseteq \text{1-DLIN}_{ktt}^{\mathrm{CFL}}$. *In particular,* $\mathrm{CFL} \cup \text{co-CFL} \subseteq \text{1-DLIN}_{1tt}^{\mathrm{CFL}}$.

**Proof.** We show the lemma by induction on $k \geq 1$. As the base case $k = 1$, it is clear that $\mathrm{CFL} \subseteq \text{1-DLIN}_m^{\mathrm{CFL}} \subseteq \text{1-DLIN}_{1tt}^{\mathrm{CFL}}$. Consider the induction step $k > 1$. Assume that $k$ is even. Note that $\mathrm{CFL}_k = \mathrm{CFL}_{k-1} \wedge \text{co-CFL}$. Take $L_1 \in \mathrm{CFL}_{k-1}$ and $L_2 \in \text{co-CFL}$ and let $L = L_1 \cap L_2$. Assume that an npda $M_2$ computes $\overline{L_2}$. y the induction hypothesis, $L_1 \in \text{1-DLIN}_{(k-1)tt}^{\mathrm{CFL}}$. Now, let $M_1$ be a one-tape linear-time DTM computing $L_1$ with $A$ as an oracle, where $A \in \mathrm{CFL}$. On input $x$, $M_1$ produces $(k-1)$-tuple $(y_1, y_2, \ldots, y_{k-1})$ on a query tape and it holds that $B(x, \chi_{k-1}^A(y_1, y_2, \ldots, y_{k-1})) = 1$ iff $x \in L_2$. We define a new machine $N$ as follows. We simulate $M_1$ and generate query words $(y_1, y_2, \ldots, y_{k-1})$ as well as a new query word $\#x$, where $\#$ is a fresh symbol. Define $A' = A \cup \{\#x \mid M_2(x) = 1\}$, which is in CFL. We define $B'$ as follows: $B'(x, b_1 b_2 \cdots b_k) = 1$ iff $B(x, b_1 b_2 \cdots b_{k-1}) = 1$ and $b_k = 0$. Note that $B'(x, \chi_k^{A'}(y_1, y_2, \ldots, y_{k-1}, \#x)) = 1$ iff $B(x, \chi_{k-1}^A(y_1, y_2, \ldots, y_{k-1})) = 1$ and $x \in L_2$. □

Before closing this section, we will briefly discuss a notion of lowness. Let $C$ be any language family that is *many-one relativizable*. A language $A$ is called *many-one low* for $C$ if $C_m^A \subseteq C$ holds. We define $\mathrm{low}_m C$ to be the set of all languages that are low for $C$; that is, $\mathrm{low}_m C = \{A \mid C_m^A \subseteq C\}$. Similarly, we define $\mathrm{low}_{ktt} C$ as a collection of all languages that are"$k$-tt low for $C$."

***Lemma 3.8*** *Let $k \geq 1$. (1)* $\mathrm{low}_{ktt}\mathrm{CFL} \subseteq \mathrm{low}_m\mathrm{CFL} \subsetneq \mathrm{CFL}$ *and (2)* $\mathrm{REG} \subseteq \mathrm{low}_{ktt}\mathrm{CFL} \subseteq \mathrm{CFL} \cap \text{co-CFL}$.

**Proof.** (1) The last containment is shown as follows. Let $A \in \mathrm{low}_m\mathrm{CFL}$. This means that $\mathrm{CFL}_m^A \subseteq \mathrm{CFL}$. Note that $A$ belongs to $\mathrm{CFL}_m^A$. Thus, we obtain $A \in \mathrm{CFL}_m^A \subseteq \mathrm{CFL}$. Next, we wish to show that $\mathrm{CFL} \neq \mathrm{low}_m\mathrm{CFL}$. Assume that $\mathrm{CFL} = \mathrm{low}_m\mathrm{CFL}$. This implies that $\mathrm{CFL}_m^{\mathrm{CFL}} \subseteq \mathrm{CFL}$. Since $\mathrm{CFL}(2) \subseteq \mathrm{CFL}_m^{\mathrm{CFL}}$ by Lemma 3.1, we immediately obtain $\mathrm{CFL}(2) = \mathrm{CFL}$. This is a contradiction against the well-known result that $\mathrm{CFL}(2) \neq \mathrm{CFL}$. Thus, we conclude that $\mathrm{low}_m\mathrm{CFL} \neq \mathrm{CFL}$.

(2) Lemma 3.6 implies $\mathrm{CFL}_{ktt}^{\mathrm{REG}} = \mathrm{CFL}$. The first containment follows immediately. For the second containment, take

any language $A \in \mathrm{low}_{ktt}\mathrm{CFL}$; that is, $\mathrm{CFL}^{A}_{ktt} \subseteq \mathrm{CFL}$. Since $A, \overline{A} \in \mathrm{CFL}^{A}_{ktt}$, we then obtain $A, \overline{A} \in \mathrm{CFL}$. Thus, $A$ is in $\mathrm{CFL} \cap \text{co-CFL}$. □

## 4. Relations among Function Classes

We will discuss our theme of various classes of multi-valued partial functions and exhibit basic relationships among those function classes. We begin with the following simple observation on functions in CFLMV.

***Lemma 4.1*** *If $f$ is in* CFLMV*, then dom($f$) belongs to* CFL.

Let us consider an npda $M$ with a write-only output tape. Since the tape heads of $M$ may stay still at any moments (by making $\lambda$-moves) on both input and output tapes, it seems difficult to synchronize the moves of those two heads so that we can split the output tape into two tracks and produce a string $[\,{}^{x}_{y}]$ from input string $x$ and output string $y$ of $M$. The best we can do is to insert a fresh symbol, say, $\natural$ between input symbols as well as output symbols to adjust the speeds of two heads, as shown in the next lemma.

***Lemma 4.2*** *For any function $f \in$* CFLMV*, there exists a language $A \in$* CFL *such that, for every pair $x$ and $y$, $y \in f(x)$ iff $[\,{}^{\tilde{x}}_{\tilde{y}}] \in A$ for certain strings $\tilde{x}$ and $\tilde{y}$, from which $x$ and $y$ can be obtained simply by removing the symbol $\natural$.*

Let us recall the notion of refinement ($\sqsubseteq_{ref}$), which is often more suitable to use than set containment ($\subseteq$) is. Hereafter, we will consider containment relations among function classes. First, we are focused on the number of distinct output strings of functions in $\Sigma^{\mathrm{CFL}}_{k}\mathrm{MV}$ with $k \geq 1$. Given a fixed number $e \in \mathbb{N}^{+}$, we introduce a new function class $\Sigma^{\mathrm{CFL}}_{k}e\mathrm{V}$ as a subclass of $\Sigma^{\mathrm{CFL}}_{k}\mathrm{MV}$ whose elements $f$ satisfy that $\| f(x) \| \leq e$ for every input $x$.

***Lemma 4.3*** *Let $k \geq 1$. If $\Sigma^{\mathrm{CFL}}_{k}2\mathrm{V} \sqsubseteq_{ref} \Sigma^{\mathrm{CFL}}_{k}\mathrm{SV}$, then the following condition holds: for every two languages $A, B \in \Sigma^{\mathrm{CFL}}_{k}$, there are two languages $A', B' \in \Sigma^{\mathrm{CFL}}_{k}$ satisfying that (i) $A' \subseteq A$ and $B' \subseteq B$, (ii) $A' \cap B' = \emptyset$, and (iii) $A \cup B = A' \cup B'$.*

**Proof.** We will show the case of $k = 1$. Assume that $\mathrm{CFL2V} \sqsubseteq_{ref} \mathrm{CFLSV}$. Let $A$ and $B$ be any two languages in CFL. Let $\Gamma = \{a, b\}$ and define $f(x) = \{a \mid x \in A\} \cup \{b \mid x \in B\}$. Clearly, $f$ belongs to CFL2V. Now, we choose a function $g \in \mathrm{CFLSV}$ that is a refinement of $f$. We define $A' = \{x \mid g(x) = \{a\}\}$ and $B' = \{x \mid g(x) = \{b\}\}$. It is easy to show that $A'$ and $B'$ are both context-free. By the choice of $g$, it holds that $A' \subseteq A$, $B' \subseteq B$, $A' \cup B' = A \cup B$, and $A' \cap B' = \emptyset$. □

Next, we turn our attention to a function class CFL($k$)MV composed of functions $f$ of the form $f(x) = \bigcap_{i=1}^{k} g_i(x)$ for $g_1, g_2, \ldots, g_k \in \mathrm{CFLMV}$. Similarly, CFL($k$)SV can be defined using CFLSV instead of CFLMV. The class separation stated below follows indirectly from a result in [5].

***Proposition 4.4*** *For any number $k \geq 1$,* CFL$(k+1)$MV $\not\sqsubseteq_{ref}$ CFL($k$)SV.

When $k = 1$, it is possible to strengthen the above proposition.

***Proposition 4.5*** CFL(2)MV $\not\sqsubseteq_{ref}$ CFLSV$/n$.

**Proof.** To show the desired separation, we start with the following claim.

***Claim 1*** CFL(2)SV $\sqsubseteq_{ref}$ CFLSV$/n$ *implies* CFL(2) $\subseteq$ CFL$/n$.

To lead to a contradiction, we assume that CFL(2)MV $\sqsubseteq_{ref}$ CFLSV$/n$. Since CFL(2)SV $\subseteq$ CFL(2)MV, Claim 1 implies CFL(2) $\subseteq$ CFL$/n$. However, this contradicts a known fact that CFL(2) $\nsubseteq$ CFL$/n$ [18]. Therefore, CFL(2)MV $\not\sqsubseteq_{ref}$ CFLSV$/n$ must hold. □

Let us consider functional compositions. The *functional composition* $f \circ g$ of two multi-valued partial functions $f$ and $g$ is defined as $(f \circ g)(x) = \bigcup_{y \in g(x)} f(y)$. For convenience, we define $\mathrm{CFLSV}^{(1)} = \mathrm{CFLSV}$ and $\mathrm{CFLSV}^{(k+1)} = \mathrm{CFLSV} \circ \mathrm{CFLSV}^{(k)}$ for any number $k \geq 1$. For instance, the function $f(x) = \{xx\}$ for any $x$ belongs to CFLSV $\circ$ CFLSV. This fact yields the following simple containment.

***Lemma 4.6*** CFL(2)SV $\subseteq$ $\mathrm{CFLSV}^{(5)}$.

Given two function classes $\mathcal{F}$ and $\mathcal{G}$, the notation $\mathcal{F} \ominus \mathcal{G}$ denotes the set $\{f - g \mid f \in \mathcal{F}, g \in \mathcal{G}\}$, where $(f - g)(x) = f(x) - g(x)$ (set difference) for any $x$. Let $LEN_{lin} = \{f \mid \exists \Sigma\, \exists a, b \in \mathbb{N}\, \forall x\, [f(x) = \Sigma^{\leq a|x|+b}]\}$. Although the function class $LEN_{lin} \ominus \mathrm{CFLMV}$ looks like the "complement" of CFLMV in the sense of [2], the double "complements" of CFLMV may not be the same as CFLMV.

***Lemma 4.7*** CFLMV $\subseteq$ $LEN_{lin} \ominus (LEN_{lin} \ominus \mathrm{CFLMV})$ $\subseteq$ CFL(2)MV.

In the previous section, we discussed a many-one relativization $\mathrm{CFL}^{A}_{m}$. Here, we will extend $\mathrm{CFL}^{A}_{m}$ to $\mathrm{CFLMV}^{A}_{m}$. Given a language $A$ over an alphabet $\Gamma$, a function $f : \Sigma^{*} \to \Gamma^{*}$ is in $\mathrm{CFLMV}^{A}_{m}$ if there exists an npda $M$ with two write-only tapes (one of which is a standard output tape and the other is a query tape) such that, on any input $x \in \Sigma^{*}$, (1) along each computation path $p \in ACC_{M}(x)$, $M$ produces a query string $y_p$ on the query tape and also an output string $z_p$ on the output tape and (2) $f(x)$ equals the set $\{z_p \mid y_p \in A, p \in ACC_{M}(x)\}$. It is obvious that $\mathrm{CFLMV} \subseteq \mathrm{CFLMV}^{\Sigma^{*}}_{m}$ for any alphabet $\Sigma$ and, in particular, $\mathrm{CFLMV}^{\emptyset}_{m} = \{f \mid \forall x\, [f(x) = \emptyset]\}$. Similarly, we can define another relativized function class $\mathrm{CFLSV}^{A}_{m}$. Given a language family $\mathcal{C}$, we set $\mathrm{CFLMV}^{\mathcal{C}}_{m} = \bigcup_{A \in \mathcal{C}} \mathrm{CFLMV}^{A}_{m}$ and $\mathrm{CFLSV}^{\mathcal{C}}_{m} = \bigcup_{A \in \mathcal{C}} \mathrm{CFLSV}^{A}_{m}$.

The next lemma is compared with Lemma 4.6.

***Lemma 4.8*** $\mathrm{CFLSV}^{\mathrm{CFL}}_{m} \subseteq \mathrm{CFLSV}^{(4)}$.

In Section 3, we have defined $\{\Sigma^{\mathrm{CFL}}_{k}, \Pi^{\mathrm{CFL}}_{k} \mid k \in \mathbb{N}\}$. Analogously, we define $\Sigma^{\mathrm{CFL}}_{1}\mathrm{MV} = \mathrm{CFLMV}$ and $\Sigma^{\mathrm{CFL}}_{k}\mathrm{MV} = \mathrm{CFLMV}^{\Pi^{\mathrm{CFL}}_{k}}_{m}$ for each $k \geq 2$. In a similar way, we can define $\Sigma^{\mathrm{CFL}}_{k}\mathrm{SV}$. We will consider the refinement of multi-valued functions by single-valued ones.

***Lemma 4.9*** *For every index $k \geq 1$, $\Sigma_k^{\mathrm{CFL}}\mathrm{MV} \sqsubseteq_{ref} \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV}$.*

**Proof.** For readability, we will show the lemma only for $k = 1$. Take an arbitrary function $f \in$ CFLMV and its underlying npda $M$ with a write-only output tape. We define a new npda $N$ equipped with two write-only tapes, as described before. On input $x$, $N$ tries to simulate $M(x)$. Let $p$ be any computation path in $PATH_M(x)$. While following this path $p$, $N$ writes down a query string $[{x \atop p}]$ on $N$'s query tape and also an outcome of $M(x)$ along $p$ on $N$'s output tape. For a technical reason, we need to adjust the length of $x$ by inserting extra new symbols "$\lambda$," indicating a $\lambda$-move.

We define another npda $N'$ as follows. On input $[{x \atop p}]$, $N'$ guesses a computation path $q$ of $M(x)$, simulates $M(x)$ along this computation path $q$. While this simulation step, if $N'$ discovers that $q \geq p$ (in a fixed lexicographic order on $PATH_M(x)$), then $N'$ immediately rejects the input. Otherwise, $N'$ reaches the same final state as $M$ does. Note that $N'$ accepts $[{x \atop p}]$ iff there exists a computation path $q$ such that $q < p$ and $M$ accepts $x$ along $q$. Now, let $A = L(N')$. Since $N'$ uses no output tape, $A$ clearly belongs to CFL. Let $y_0$ be the outcome of an accepting computation path, say, $p_0$ that is the smallest in a lexicographic order among $PATH_M(x)$. It is not difficult to show that $y_0 \in f(x)$ iff there exists an accepting computation path $p$ such that $[{x \atop p}] \notin A$ and $N$ outputs $y_0$ along $p$. Define $g$ to be a function that is computed by $N$ using $\overline{A}$ as an oracle. This implies that $g \in \mathrm{CFLSV}_m^{\overline{A}}$, which yields $g \in \mathrm{CFLSV}_m^{\text{co-CFL}}$. Moreover, $g$ is single-valued. It also holds that $y_0 \in f(x)$ iff $y_0 \in g(x)$. □

We want to show a separation $\Sigma_k^{\mathrm{CFL}}\mathrm{MV} \not\sqsubseteq_{ref} \Sigma_k^{\mathrm{CFL}}\mathrm{SV}$, where $k \geq 2$, under the assumption that the polynomial hierarchy does not collapse down to the $k$th level.

***Theorem 4.10*** *For any number $k \geq 2$, if* $\mathrm{PH} \neq \Sigma_k^{\mathrm{P}}$, *then* $\Sigma_k^{\mathrm{CFL}}\mathrm{MV} \not\sqsubseteq_{ref} \Sigma_k^{\mathrm{CFL}}\mathrm{SV}$.

Since the proof of the above theorem requires a discussion on oracles that compute multi-valued partial functions, we postpone the proof until Section 5. Later in Section 6, we will demonstrate that CFLMV $\not\sqsubseteq_{ref}$ CFLSV; meanwhile, we prove a slightly weaker separation by employing a proof that is similar in essence to the proof of Proposition 4.5.

***Proposition 4.11*** $\Sigma_2^{\mathrm{CFL}}\mathrm{MV} \not\sqsubseteq_{ref} \mathrm{CFLSV}/n$.

**Proof.** Toward a contradiction, we assume that $\Sigma_2^{\mathrm{CFL}}\mathrm{MV} \sqsubseteq_{ref} \mathrm{CFLSV}/n$. First, we claim the following.

***Claim 2*** *If* $\Sigma_2^{\mathrm{CFL}}\mathrm{SV} \sqsubseteq_{ref} \mathrm{CFLSV}/n$, *then* co-CFL $\subseteq$ CFL$/n$.

Since $\Sigma_2^{\mathrm{CFL}}\mathrm{MV} \sqsubseteq_{ref} \mathrm{CFLSV}/n$, the above claim implies that co-CFL $\subseteq$ CFL$/n$. However, this contradicts the result co-CFL $\nsubseteq$ CFL$/n$ [15]. Therefore, we conclude that $\Sigma_2^{\mathrm{CFL}}\mathrm{MV} \not\sqsubseteq_{ref} \mathrm{CFLSV}/n$. The second assertion of the lemma follows from the first assertion. □

Finally, we will discuss the many-one lowness of CFLMV and CFLSV. Given a function class $\mathcal{F}$ that is *many-one relativizable* (i.e., $\mathcal{F}_m^A$ is defined for every set $A$), $\mathrm{low}_m\mathcal{F}$ denotes the set of all languages $A$ satisfying the containment $\mathcal{F}_m^A \subseteq \mathcal{F}$ [13]. It was proven in [13] that, for instance, REG $=$ $\mathrm{low}_m$1-#LIN $=$ $\mathrm{low}_m$1-GapLIN, where 1-#LIN and 1-GapLIN are one-tape linear-time analogues of #P and GapP, respectively.

***Proposition 4.12*** REG $\subseteq$ $\mathrm{low}_m$CFLSV $\subseteq$ $\mathrm{low}_m$CFL$k$V $\subseteq$ $\mathrm{low}_m$CFLMV $\subsetneq$ CFL *for any $k \geq 1$.*

**Proof.** Note that $\mathrm{CFLSV}_m^{\mathrm{REG}} = \mathrm{CFLSV}$ by Lemma 3.6. Hence, the first containment follows. The second and the third containments are trivial. Next, we will focus on the last containment. Let $A \in \mathrm{low}_m\mathrm{CFLMV}$. Let us define $g_A(x) = \{1 \mid x \in A\}$ for every string $x$. This function $g_A$ belongs to $\mathrm{CFLSV}_m^A$. By the choice of $A$, it follows that $g_A \in \mathrm{CFLSV}_m^A \subseteq \mathrm{CFLMV}_m^A \subseteq \mathrm{CFLMV}$. Since $g_A$ is single valued, $g_A$ must be in CFLSV. Thus, we conclude that $A$ is in CFL.

If $\mathrm{low}_m\mathrm{CFLMV} = \mathrm{CFLMV}$, then we obtain $\mathrm{CFLMV}_m^{\mathrm{CFL}} = \mathrm{CFLMV}$, which implies CFL(2)MV $=$ CFLMV. However, this contradicts Proposition 4.5. Therefore, $\mathrm{low}_m\mathrm{CFLMV} \neq \mathrm{CFLMV}$ follows. □

## 5. Oracles That Compute Multiple Values

Up to now, we have treated oracles in various reductions as sets (or equivalently, languages), which are also seen as single-valued total functions whose output values are limited to $\{0, 1\}$. Here, we want to expand those oracles to multi-valued partial functions. Earlier, Fenner, Homer, Ogihara, and Selman [2] proposed a coherent treatment of multi-valued partial functions within a special framework of relativization with oracles that compute multiple values. In this section, we will adopt their definitions (with appropriate modifications) and discuss relations among function classes that have appeared in the previous sections. However, we will limit our attention only on logarithmic-space many-one reductions between two multi-valued partial functions. To distinguish this reducibility from the reducibilities in the previous sections, we will intentionally use a slightly different notation. Given a multi-valued partial function $g$, we define a function class $\mathrm{FL}_{m*}^g$ as follows. A multi-valued partial function $f$ is in $\mathrm{FL}_{m*}^g$ if there exists a function $h_1 \in$ FL and another function $h_2 \in$ FL(partial) that satisfy the following condition: for any single-valued partial refinement $\tilde{g}$ of $g$, a single-valued partial function $\tilde{f}_{\tilde{g}}$ defined as $\tilde{f}_{\tilde{g}}(x) = h_2(x, \tilde{g}(h_1(x)))$ for any input $x$ satisfies that (i) $f \sqsubseteq_{ref} \tilde{f}_{\tilde{g}}$ and (ii) $f(x) \subseteq \bigcup_{\tilde{g} \in R(g)} \tilde{f}_{\tilde{g}}(x)$ for all $x$'s, where $R(g)$ contains all single-valued partial refinements of $g$, provided that, whenever $\tilde{g}(h_1(x))$ is undefined, $h_2(x, \tilde{g}(h_1(x)))$ is always thought to be undefined. Condition (ii) in the above definition intuitively ensures that $f(x)$ does not contain any values that cannot be produced by any triplet $(h_1, h_2, \tilde{g})$. With this condition, our framework differs from that of [2]. For a function class $\mathcal{F}$, we set $\mathrm{FL}_{m*}^{\mathcal{F}}$ to be the union $\bigcup_{g \in \mathcal{F}} \mathrm{FL}_{m*}^g$.

***Lemma 5.1*** *(1) For any multi-valued partial functions $f$ and $g$, if $f \sqsubseteq_{ref} g$, then* $\mathrm{FL}_{m*}^f \sqsubseteq_{ref} \mathrm{FL}_{m*}^g$.
*(2) For any function classes $\mathcal{F}$ and $\mathcal{G}$, if $\mathcal{F} \sqsubseteq_{ref} \mathcal{G}$, then* $\mathrm{FL}_{m*}^{\mathcal{F}} \sqsubseteq_{ref} \mathrm{FL}_{m*}^{\mathcal{G}}$.

**Proof.** (1) Let $h \in \mathrm{FL}^{f}_{m*}$ and take two functions $h_1 \in \mathrm{FL}$ and $h_2 \in \mathrm{FL}(\text{partial})$ such that $f \sqsubseteq_{ref} \tilde{f}$, $h \sqsubseteq_{ref} \tilde{h}_{\tilde{f}}$, and $\tilde{h}_{\tilde{f}}(x) = h_2(x, \tilde{f}(h_1(x)))$ for any single-valued partial refinement $\tilde{f}$ of $f$. Let $\tilde{g}$ be any single-valued refinement of $g$. From $f \sqsubseteq_{ref} g$ follows $f \sqsubseteq_{ref} \tilde{g}$. This entails that $\tilde{h}_{\tilde{f}} \sqsubseteq_{ref} \tilde{h}_{\tilde{g}}$; thus, $h \sqsubseteq_{ref} \tilde{h}_{\tilde{g}}$ holds. Let us define $r(x) = \bigcup_{\tilde{g}\in R(g)} \tilde{h}_{\tilde{g}}(x)$ for every $x$. This implies $r \in \mathrm{FL}^{g}_{m*}$. Moreover, since $h \sqsubseteq_{ref} \tilde{h}_{\tilde{g}}$ for any single-valued partial refinement $\tilde{g}$ of $g$, we obtain $h \sqsubseteq_{ref} r$. As a consequence, $\mathrm{FL}^{f}_{m*} \sqsubseteq_{ref} \mathrm{FL}^{g}_{m*}$. (2) This follows from (1). □

In accordance with the set oracles discussed in the previous sections, when $g$ coincides with the characteristic function $\chi_A$ for a certain language $A$, we succinctly write $\mathrm{FL}^{A}_{m*}$ instead of $\mathrm{FL}^{\chi_A}_{m*}$. Given a language family $C$, $\mathrm{FL}^{C}_{m*}$ denotes $\bigcup_{A\in C} \mathrm{FL}^{A}_{m*}$.

***Lemma 5.2*** $\mathrm{FL}^{\mathrm{CFL}}_{m*} \subseteq \mathrm{FL}^{\mathrm{CFLSV}}_{m*}$.

**Proof.** Let $h \in \mathrm{FL}^{A}_{m*}$ for a certain language $A \in \mathrm{CFL}$. Note that, for any partial function $g$, $\chi_A \sqsubseteq_{ref} g$ implies $g = \chi_A$. There exist functions $h_1 \in \mathrm{FL}$ and $h_2 \in \mathrm{FL}(\text{partial})$ such that the function $\tilde{h}_A$ defined as $\tilde{h}_A(x) = h_2(x, \chi_A(h_1(x)))$ for every string $x$ satisfies that $h \sqsubseteq_{ref} \tilde{h}_A$ and $h(x) \subseteq \tilde{h}_A(x)$. This yields $h \in \mathrm{FL}^{\chi_A}_{m*}$. Since $\chi_A \in \mathrm{CFLSV_t}$, it follows that $h \in \mathrm{FL}^{\chi_A}_{m*} \subseteq \mathrm{FL}^{\mathrm{CFLSV}}_{m*}$. □

In Section 3, we have left Theorem 4.10 unproven. In what follows, we want to give the proof of Theorem 4.10. To prove this theorem, we first introduce two function classes $\Sigma^{\mathrm{P}}_k\mathrm{MV}$ and $\Sigma^{\mathrm{P}}_k\mathrm{SV}$ in a way similar to $\Sigma^{\mathrm{P}}_k$ using NPMV and NPSV in place of NP; that is, $\Sigma^{\mathrm{P}}_k\mathrm{MV} = \mathrm{NPMV}^{\Sigma^{\mathrm{P}}_{k-1}}$ and $\Sigma^{\mathrm{P}}_k\mathrm{SV} = \mathrm{NPSV}^{\Sigma^{\mathrm{P}}_{k-1}}$ for every index $k \geq 1$. The following relations similar to Lemma 3.4 hold.

***Proposition 5.3*** $\mathrm{FL}^{\Sigma^{\mathrm{CFL}}_{k+1}\mathrm{MV}}_{m*} = \Sigma^{\mathrm{P}}_k\mathrm{MV}$ *and* $\mathrm{FL}^{\Sigma^{\mathrm{CFL}}_{k+1}\mathrm{SV}}_{m*} = \Sigma^{\mathrm{P}}_k\mathrm{SV}$ *for any index* $k \geq 1$.

***Lemma 5.4*** *Let* $k \geq 1$.
*( 1 )* $\Sigma^{\mathrm{CFL}}_{k+1}\mathrm{MV} \sqsubseteq_{ref} \Sigma^{\mathrm{CFL}}_{k+1}\mathrm{SV}$ *implies* $\Sigma^{\mathrm{P}}_k\mathrm{MV} \sqsubseteq_{ref} \Sigma^{\mathrm{P}}_k\mathrm{SV}$.
*( 2 )* $\Sigma^{\mathrm{P}}_k\mathrm{MV} \sqsubseteq_{ref} \Sigma^{\mathrm{P}}_k\mathrm{SV}$ *implies* $\mathrm{PH} = \Sigma^{\mathrm{P}}_{k+1}$.

**Proof.** (1) Assume that $\Sigma^{\mathrm{CFL}}_{k+1}\mathrm{MV} \sqsubseteq_{ref} \Sigma^{\mathrm{CFL}}_{k+1}\mathrm{SV}$. Lemma 5.1 implies that $\mathrm{FL}^{\Sigma^{\mathrm{CFL}}_{k+1}\mathrm{MV}}_{m*} \sqsubseteq_{ref} \mathrm{FL}^{\Sigma^{\mathrm{CFL}}_{k+1}\mathrm{SV}}_{m*}$. By Proposition 5.3, the conclusion $\Sigma^{\mathrm{P}}_k\mathrm{MV} \sqsubseteq_{ref} \Sigma^{\mathrm{P}}_k\mathrm{SV}$ immediately follows.

(2) When $k = 1$, the claim was already proven in [4] using an analysis of NPSV-selective sets. The case of $k > 1$ can be treated in essence in a way similar to the first case. □

Now, we are ready to give its proof of Theorem 4.10 with a help of Lemma 5.4.

**Proof of Theorem 4.10.** Let $k \geq 1$. We want to show contrapositive of the theorem. Assume that $\Sigma^{\mathrm{CFL}}_{k+1}\mathrm{MV} \sqsubseteq_{ref} \Sigma^{\mathrm{CFL}}_{k+1}\mathrm{SV}$. By Lemma 5.4(1), we conclude that $\Sigma^{\mathrm{P}}_k\mathrm{MV} \sqsubseteq_{ref} \Sigma^{\mathrm{P}}_k\mathrm{SV}$. This result implies $\mathrm{PH} = \Sigma^{\mathrm{P}}_{k+1}$ by Lemma 5.4(2). □

## 6. Lengthwise Selective Languages

To accommodate specifications of npda's, we slightly modify the notion of $\mathcal{F}$-selectivity by demanding that two arbitrary inputs to a selector must have the same length to obtain its meaningful outcomes. Because of this length requirement for the selector, we prefer to use a new term "lengthwise $\mathcal{F}$-selective."

To treat input instances to a 2-ary partial function $f$ on a model of one-way machine $M$, we need to take a slightly unconventional approach. When a pair $(x, y)$ of strings is given as an input instance to the machine $M$, we split the input tape into two tracks and place $x$ in the upper track and $y$ in the lower track as $[\,^{x}_{y}]$ so that the tape head can read $x$ and $y$ simultaneously from the left to the right of them. More formally, if we wish to compute $f$ on input $(x, y)$, we start its underlying machine $M$ with its input tape that contains the string of the form $[\,^{x}_{y}]$. From this formalism, we might have written $f([\,^{x}_{y}])$ to describe the function $f$ taking an input of the form $[\,^{x}_{y}]$; however, we prefer to follow a standard notation of $f(x, y)$ throughout this paper.

Given a function class $\mathcal{F}$, a language $A$ is said to be *lengthwise* $\mathcal{F}$*-selective* if there exists a 2-ary function $f : \Sigma^* \times \Sigma^* \to \mathcal{P}(\Sigma^*)$ (called a *selector*) in $\mathcal{F}$ such that, for every two strings $x, y \in \Sigma^*$, (i) $f(x, y) \subseteq \{x, y\}$ and (ii) if $\{x, y\} \cap A \neq \emptyset$ and $|x| = |y|$, then $f(x, y) \neq \emptyset$ and $f(x, y) \subseteq A$. Note that, whenever $|x| \neq |y|$, we may assume that $f(x, y) = \emptyset$. We write $\mathcal{F}\text{-SEL}_{\mathrm{lw}}$ for the collection of all lengthwise $\mathcal{F}$-selective languages.

***Lemma 6.1*** *Let* $k \geq 1$. *For every language* $A \in \Sigma^{\mathrm{CFL}}_k\mathrm{SV_t}\text{-SEL}_{\mathrm{lw}}$, *there always exists a* $\Sigma^{\mathrm{CFL}}_k\mathrm{SV_t}$*-selector* $f$ *for* $A$ *such that* $f(x, y) = f(y, x)$ *holds for any pair* $(x, y)$. *This claim also holds for* $\Sigma^{\mathrm{CFL}}_k\mathrm{SV}\text{-SEL}_{\mathrm{lw}}$ *and* $\Sigma^{\mathrm{CFL}}_k\mathrm{MV}\text{-SEL}_{\mathrm{lw}}$.

**Proof.** We will show only the case of $k = 1$. Take any $\mathrm{CFLSV_t}$-selector $f$ for $A$. Let $M$ be an npda with a write-only output tape computing $f$. Now, we define $g$ as $g(x, y) = f(\min\{x, y\}, \max\{x, y\})$, provided that we have a lexicographic order on $\Sigma^n$ for every $n$. Obviously, $g$ is also a selector for $A$ and also satisfies $g(x, y) = g(y, x)$ for every pair $(x, y)$. Next, we argue that $g$ is indeed in $\mathrm{CFLSV_t}$. Consider the npda $N$ defined as follows. On input $[\,^{x}_{y}]$ with $|x| = |y|$, $N$ guesses one of the three possibilities: $x < y$, $y < x$, and $x = y$. If either "$x < y$" or "$x = y$" is guessed, then $N$ simulates $M([\,^{x}_{y}])$. If "$y < x$" is guessed, then $N$ simulates $M([\,^{y}_{x}])$ by exchanging the roles of $x$ and $y$. During reading the entire input, $N$ checks if the guess at the first step is actually correct by comparing $x$ and $y$ symbol by symbol. Whenever $N$ discovers that the guess is wrong, it enters a rejecting state immediately. Now, assume that the guess is correct. In this case, $N$ correctly simulates $M$, and thus $N$ computes $g$. Clearly, $g$ belongs to $\mathrm{CFLSV_t}$. □

An obvious upper bound of the computational complexity of $\mathrm{CFLSV\text{-}SEL_{lw}}$ is given in the following lemma. Its proof is similar in essence to that of NPSV-SEL $\subseteq$ NP/*poly* [4]. A language family 1-NTIME($t$) consists of languages recognized by one-tape NTMs within time $t(n)$, where $n$ is the length of inputs.

***Lemma 6.2*** $\mathrm{CFLSV\text{-}SEL_{lw}} \subseteq \text{1-NTIME}(O(n^2))/O(n^2)$.

Let us consider a relation between $\Sigma^{\mathrm{CFL}}_k$ and $\Sigma^{\mathrm{CFL}}_k\mathrm{SV}\text{-SEL}_{\mathrm{lw}}$.

***Proposition 6.3*** *For each level* $k \geq 2$, $\Sigma^{\mathrm{CFL}}_k\mathrm{MV} \sqsubseteq_{ref} \Sigma^{\mathrm{CFL}}_k\mathrm{SV}$ *implies* $\Sigma^{\mathrm{CFL}}_k \subseteq \Sigma^{\mathrm{CFL}}_k\mathrm{SV}\text{-SEL}_{\mathrm{lw}}$.

This proposition follows immediately from two lemmas, Lemmas 6.4 and 6.5. We start with the first lemma regarding a relation between $\Sigma_k^{\mathrm{CFL}}$ and $\Sigma_k^{\mathrm{CFL}}2\mathrm{V}$-$\mathrm{SEL}_{\mathrm{lw}}$

***Lemma 6.4*** *For every index $k \geq 1$, $\Sigma_k^{\mathrm{CFL}} \subseteq \Sigma_k^{\mathrm{CFL}}2\mathrm{V}$-$\mathrm{SEL}_{\mathrm{lw}}$.*

**Proof.** For readability, we will show the case of $k = 1$. For each language $A \in \mathrm{CFL}$, take an npda $M$ that recognize $A$. To show that $A \in \mathrm{CFL2V}$-$\mathrm{SEL}_{\mathrm{lw}}$, it suffices to construct a selector for $A$ using an appropriate npda with a write-only output tape. Let $(x, y)$ be an input pair with $|x| = |y|$. Assume that $[{x \atop y}]$ is given to an input tape. In what follows, we will describe an npda $N$ equipped with a write-only output tape. On the input $[{x \atop y}]$, $N$ nondeterministically choose $x$ or $y$ and run $M$ on the chosen input string. Consider the case where $N$ has guessed $x$. While reading $x$, $N$ copies it onto the output tape. When $M$ halts in either an accepting state or a rejecting state, $N$ enters the same inner state. Let $f(x, y)$ be the outcome of $N$ on the input $[{x \atop y}]$. It is obvious that $f$ is a multi-valued partial function. Since $f(x, y) \subseteq \{x, y\}$ for any pair $(x, y)$, $f$ belongs to CFL2V. Moreover, it is easy to show that $f(x, y) \subseteq A$ when $\{x, y\} \cap A \neq \emptyset$. Thus, $A \in \mathrm{CFL2V}$-$\mathrm{SEL}_{\mathrm{lw}}$. □

The second lemma links function classes and lengthwise selective sets.

***Lemma 6.5*** *For each index $k \geq 1$, if $\Sigma_k^{\mathrm{CFL}}2\mathrm{V} \sqsubseteq_{ref} \Sigma_k^{\mathrm{CFL}}\mathrm{SV}$, then $\Sigma_k^{\mathrm{CFL}}2\mathrm{V}$-$\mathrm{SEL}_{\mathrm{lw}} \subseteq \Sigma_k^{\mathrm{CFL}}\mathrm{SV}$-$\mathrm{SEL}_{\mathrm{lw}}$.*

**Proof.** Only the case of $k = 1$ will be shown here. Assume that $\mathrm{CFL2V} \sqsubseteq_{ref} \mathrm{CFLSV}$. Let us take any language $A \in \mathrm{CFL2V}$-$\mathrm{SEL}_{\mathrm{lw}}$ and consider a CFL2V-selector for $A$. Since $f \in \mathrm{CFL2V}$, we choose its refinement $g \in \mathrm{CFLSV}$, for which $f(x, y) \neq \emptyset$ implies $g(x, y) \neq \emptyset$ and $g(x, y) \subseteq f(x, y)$, and $f(x, y) = \emptyset$ implies $g(x, y) = \emptyset$. Moreover, since $f$ is a selector for $A$, if $\{x, y\} \cap A \neq \emptyset$, then it follows that $f(x, y) \subseteq A$ and $f(x, y) \neq \emptyset$. By combining those relations, it holds that, if $\{x, y\} \cap A \neq \emptyset$, then $g(x, y) \subseteq f(x, y) \subseteq A$ and $g(x, y) \neq \emptyset$. Hence, $g$ is also a selector for $A$. □

As a main result of this section, we will show a separation between CFL and $\mathrm{CFLSV}$-$\mathrm{SEL}_{\mathrm{lw}}$. This separation is obtained by exploiting a so-called *swapping property* of npda's (see [15], [18]).

***Theorem 6.6*** $\mathrm{CFL} \nsubseteq \mathrm{CFLSV}$-$\mathrm{SEL}_{\mathrm{lw}}$.

By Proposition 6.3, Theorem 6.6 yields the following corollary.

***Corollary 6.7*** $\mathrm{CFLMV} \not\sqsubseteq_{ref} \mathrm{CFLSV}$.

Finally, we will show a power of $\mathrm{CFLSV}_{\mathrm{t}}$-$\mathrm{SEL}_{\mathrm{lw}}$ against advised language families.

***Proposition 6.8*** $\mathrm{CFLSV}_{\mathrm{t}}$-$\mathrm{SEL}_{\mathrm{lw}} \nsubseteq \mathrm{REG}/n$.

**Proof.** In [17] appeared a special language $IP_* = \{ax \mid a \in \{\lambda, 0, 1\}, x = x_1x_2, |x_1| = |x_2|, x_1 \odot x_2^R = 1 \pmod 2\}$, where $x \odot y$ denotes the *(binary) inner product* of $x$ and $y$. It was shown in [17] that $IP_*$ is $\mathrm{REG}/n$*-pseudorandom* and therefore it is not in $\mathrm{REG}/n$. The remaining task is to show that $IP_*$ belongs to $\mathrm{CFLSV}$-$\mathrm{SEL}_{\mathrm{lw}}$. Let $(x, y)$ be any pair of strings with $|x| = |y|$. On the input $[{x \atop y}]$, we first guess either $x$ or $y$ and start copying the chosen string on the output tape. Let $x = x_1x_2$ and $y = y_1y_2$. We simultaneously guess a boundary between $[{x_1 \atop y_1}]$ and $[{x_2 \atop y_2}]$ and store $[{x_1 \atop y_1}]$ into the stack. While reading $[{x_2 \atop y_2}]$, we compute $\ell_x = x_1 \odot x_2^R \pmod 2$ and $\ell_y = y_1 \odot y_2^R \pmod 2$ using the stack content of $[{x_1 \atop y_1}]$. If $|x_1| \neq |x_2|$ is detected, we immediately enter a rejecting state. Assume otherwise. In the case where $\ell_x = \ell_y$, if we have guessed $x$, then we enter an accepting state; otherwise, we enter a rejecting state. In the case where $\ell_x < \ell_y$ (resp., $\ell_y < \ell_x$), if we have guessed $x$, then we enter a rejecting state (resp., an accepting state); otherwise, we enter an accepting state (resp., a rejecting state). It is clear that we always output a single value, and thus the function $f$ computed by this algorithm belongs to $\mathrm{CFLSV}_{\mathrm{t}}$. Moreover, $f(x, y) \subseteq A$ holds if either $x \in A$ or $y \in A$. Therefore, $A$ is in $\mathrm{CFLSV}_{\mathrm{t}}$-$\mathrm{SEL}_{\mathrm{lw}}$. □

## References

[1] Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase-structure grammars. *Z. Phonetik Sprachwiss. Kommunik.*, 14, 143–172, 1961.

[2] S. Fenner, S. Homer, M. Ogihara, and A. Selman. On using oracles that compute values. In *Proc. of the 10th Symp. on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, Springer, vol.665, pp.398–407, 1993.

[3] L. A. Hemachandra, A. Hoene, M. Ogiwara, A. L. Selman, T. Thierauf, and J. Wang. Selectivity. In *Proc. of the 5th International Conference on Computing and Information*, pp55–59, 1993.

[4] L. A. Hemaspaandra, A. V. Naik, M. Ogihara, and A. L. Selman. Computing solutions uniquely collaspses the polynomial hierarchy. *SIAM J. Comput.*, 25 (1996) 697–708.

[5] L. Y. Liu and P. Weiner. An infinite hierarchy of intersections of context-free languages. *Math. Systems Theory*, 7 (1973) 185–192.

[6] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proc. of the 13th Annual IEEE Symposium on Switching and Automata Theory*, pp.125–129, 1972.

[7] K. Reinhardt. Hierarchies over the context-free languages. In *Proc. of the 6th International Meeting of Young Computer Scientists on Aspects and Prospects of Theoretical Computer Science* (IMYCS), Lecture Notes in Computer Science, Springer, vol.464, pp.214–224, 1990.

[8] A. L. Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. *Math. Systems Theory*, 13 (1979) 55–65.

[9] A. L. Selman. A taxonomy of complexity classes of functions. *J. Comput. System Sci.*, 48 (1994) 357–381.

[10] A. L. Selman. Much ado about functions. In *Proc. of the 11th Annual IEEE Conference on Computational Complexity*, pp.198–212, 1996.

[11] L. J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3 (1976) 1–22.

[12] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *J. ACM*, 25 (1978) 405–414.

[13] K. Tadaki, T. Yamakami, and J. C. H. Lin. Theory of one-tape linear-time Turing machines. *Theor. Comput. Sci.*, 411 (2010) 22–43. An extended abstract appeared in the Proc. of the 30th SOFSEM Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2004), Lecture Notes in Computer Science, Springer, vol.2932, pp.335–348, 2004.

[14] H. Venkateswaran. Properties that characterize LOGCFL. *J. Comput. System Sci.*, 42 (1991) 380–404.

[15] T. Yamakami. Swapping lemmas for regular and context-free languages. Available at arXiv:0808.4122, 2008.

[16] T. Yamakami. The roles of advice to one-tape linear-time Turing machines and finite automata. *Int. J. Found. Comput. Sci.*, 21 (2010) 941–962. An early version appeared in the Proc. of the 20th International Symposium on Algorithms and Computation (ISAAC 2009), Lecture Notes in Computer Science, Springer, vol.5878, pp.933–942, 2009.

[17] T. Yamakami. Immunity and pseudorandomness of context-free languages. *Theor. Comput. Sci.*, 412 (2011) 6432–6450.

[18] T. Yamakami. Pseudorandom generators against advised context-free languages. Available at arXiv:0902.2774, 2009.

[19] T. Yamakami and Y. Kato. The dissecting power of regular languages. Available at arXiv:1202.4883, 2012.