Abstract

This paper continues a systematic and comprehensive study on the structural properties of CFL functions, which are in general multi-valued partial functions computed by one-way one-head nondeterministic pushdown automata equipped with write-only output tapes (or pushdown transducers), where CFL refers to a relevance to context-free languages. The CFL functions tend to behave quite differently from their corresponding context-free languages. We extensively discuss containments, separations, and refinements among various classes of functions obtained from the CFL functions by applying Boolean operations, functional composition, many-one relativization, and Turing relativization. In particular, Turing relativization helps construct a hierarchy over the class of CFL functions. We also analyze the computational complexity of optimization functions, which are to find optimal values of CFL functions, and discuss their relationships to the associated languages.

Keywords: multi-valued partial function, oracle, Boolean operation, refinement, many-one relativization, Turing relativization, CFL hierarchy, optimization, pushdown automaton, context-free language

1 Much Ado about Functions

In a traditional field of formal languages and automata, we have dealt primarily with languages because of their practical applications to, for example, a parsing analysis of programming languages. Most fundamental languages listed in many undergraduate textbooks are, unarguably, regular (or rational) languages and context-free (or algebraic) languages.

Opposed to the recognition of languages, translation of words, for example, requires a mechanism of transforming input words to output words. Aho et al. [1] studied machines that produce words on output tapes while reading symbols on an input tape. Mappings on strings (or word relations) that are realized by such machines are known as transductions. Since languages are regarded, from an integrated viewpoint, as Boolean-valued (i.e., \{0, 1\}-valued) total functions, it seems more essential to study the behaviors of those functions. This task is, however, quite challenging, because these functions often demand quite different concepts, technical tools, and proof arguments, compared to those for languages. When underlying machines are particularly nondeterministic, they may produce numerous distinct output values (including the case of no output values). Mappings realized by such machines become, in general, multi-valued partial functions transforming each admissible string to a certain finite (possibly empty) set of strings.

Based on a model of polynomial-time nondeterministic Turing machine, computational complexity theory has long discussed the structural complexity of various NP function classes, including NPMV, NPSV, and NPSV_t (where MV and SV respectively stand for “multi-valued” and “single-valued” and the subscript “t” does for “total”). See, e.g., a survey [14].

Within a scope of formal languages and automata, there is also rich literature concerning the behaviors of nondeterministic finite automata equipped with write-only output tapes (known as rational transducers) and properties of associated multi-valued partial functions (known also as rational transductions). Significant efforts have been made over the years to understand the functionality of such functions. A well-known field of functions include “CFL functions,” which were formally discussed in 1963 by Evey [4] and Fisher [6] and general properties have been since then discussed in, e.g., [3, 10]. CFL functions are generally computed by one-way one-head nondeterministic pushdown automata (or npda’s, in short) equipped with write-only output tapes. For example, the function \( PAL_{\text{sub}}(w) = \{ x \in \{0, 1\}^* \mid \exists u, v [w = uxv], x = x^R \} \) for every \( w \in \{0, 1\}^* \) is a CFL function, where \( x^R \) is \( x \) in reverse. As subclasses of CFL functions, three fundamental function classes CFLMV, CFLSV, and CFLSV_t were recognized explicitly in [19] and further explored in [20].

In recent literature, fascinating structural properties of CFL functions have been extensively investigated. Konstantinidis et al. [10] took an algebraic approach toward the characterization of CFL functions. In relation to cryptography, it was shown that there exists a pseudorandom generator in CFLSV_t that “fools” every
language in a non-uniform (or an advised) version of REG [19]. Another function class CFLMV(2) in [20] contains pseudorandom generators against a non-uniform analogue of CFL. 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 CFLSV can be seen as a functional extension of the language family CFL \cap co-CFL rather than CFL [20]. In stark contrast with a tidy theory of NP functions, circumstances that surround CFL functions differ significantly 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 with output tapes. One such example is concerning a notion of refinement [13] (or uniformization [10]). Unlike language families, a containment between two multi-valued partial functions is customarily replaced by refinement. Konstantinidis et al. [10] posed a basic question of whether every function in CFLMV has a refinement in CFLSV. This question was lately solved negatively [22] and this result clearly contrasts a situation that a similar relationship is not known to hold between NPMV and NPSV.

Amazingly, there still remains a vast range of structural properties that await for further investigation. Thus, we wish to continue a coherent and systematic study on the structural behaviors of the aforementioned CFL functions. This paper aims at exploring fundamental relationships (such as, containment, separation, and refinement) among the above function classes and their natural extensions via four typical operations: (i) Boolean operations, (ii) functional composition, (iii) many-one relativization, and (iv) Turing relativization. The last two operations are a natural generalization of \textit{many-one} and \textit{Turing CFL-reducibilities} among languages [21]. We use the Turing relativization to introduce a hierarchy of function classes \( \Sigma^{k}_{CFL-MV}, \Pi^{k}_{CFL-MV}, \text{ and } \Sigma^{k}_{CFL-SV} \) for each level \( k \geq 1 \), in which the first \( \Sigma \)-level classes coincide with CFLMV and CFLSV, respectively. We show that all functions in this hierarchy have linear space-complexity. With regard to refinement, we show that, if the CFL hierarchy of [21] collapses to the CFLSV, respectively. We show that all functions in this hierarchy have linear space-complexity. Thus, we wish to continue a coherent and systematic study on the structural behaviors of the aforementioned CFL functions. This paper aims at exploring fundamental relationships (such as, containment, separation, and refinement) among the above function classes and their natural extensions via four typical operations: (i) Boolean operations, (ii) functional composition, (iii) many-one relativization, and (iv) Turing relativization. The last two operations are a natural generalization of \textit{many-one} and \textit{Turing CFL-reducibilities} among languages [21]. We use the Turing relativization to introduce a hierarchy of function classes \( \Sigma^{k}_{CFL-MV}, \Pi^{k}_{CFL-MV}, \text{ and } \Sigma^{k}_{CFL-SV} \) for each level \( k \geq 1 \), in which the first \( \Sigma \)-level classes coincide with CFLMV and CFLSV, respectively. We show that all functions in this hierarchy have linear space-complexity. With regard to refinement, we show that, if the CFL hierarchy of [21] collapses to the CFLSV, respectively.
Given any language $A$ (used as an oracle), $\text{CFL}_A^T$ (or $\text{CFL}_T(A)$) expresses a collection of all languages recognized by npda’s equipped with write-only query tapes with which the npda’s make non-adaptive oracle queries to $A$, provided that all computation paths of the npda’s must terminate in $O(n)$ steps no matter what oracle is used [21]. We use the notation $\text{CFL}_F^T$ (or $\text{CFL}_F^T(C)$) for language family $C$ to denote the union $\bigcup_{A \in C} \text{CFL}_A^T$. Its deterministic version is expressed as $\text{DCFL}_F^T$. The $\text{CFL}$ hierarchy $\{\Sigma_k^{\text{CFL}}, \Sigma_k^{\text{CFL}}, \Pi_k^{\text{CFL}} \mid k \in \mathbb{N}^+\}$ is composed of classes $\Sigma_k^{\text{CFL}} = \text{DCFL}$, $\Sigma_k^{\text{CFL}} = \text{CFL}$, $\Pi_k^{\text{CFL}} = \text{co-}\Sigma_k^{\text{CFL}}$, $\Delta_k^{\text{CFL}} = \text{DCFL}_T(\Pi_k^{\text{CFL}})$, and $\Sigma_k^{\text{CFL}} = \text{CFL}_T(\Pi_k^{\text{CFL}})$ for each index $k \geq 1$ [21].

**Functions and Refinement.** Our terminology associated with multi-valued partial functions closely follows the standard terminology in computational complexity theory. Throughout this paper, we adopt the following convention: the generic term “function” always refers to “multi-valued partial function,” provided that single-valued functions are viewed as a special case of multi-valued functions and, moreover, partial functions include total functions. We are 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 $f$, $\text{dom}(f)$ denotes the domain of $f$, defined to be $\{x \in \Sigma^* \mid f(x) \neq \emptyset\}$. If $x \notin \text{dom}(x)$, then $f(x)$ is said to be undefined. The range $\text{ran}(f)$ of $f$ is a set $\{y \in \Gamma^* \mid f^{-1}(y) \neq \emptyset\}$.

For any language $A$, the characteristic function for $A$, denoted by $\chi_A$, is a function defined as $\chi_A(x) = 1$ if $x \in A$ and $\chi_A(x) = 0$ otherwise. We also use a quasi-characteristic function $\eta_A$, which is defined as $\eta_A(x) = 1$ for any string $x$ in $A$ and $\eta_A(x)$ is not defined for all the other strings $x$.

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 $f : \Sigma^* \rightarrow \mathcal{P}(\Gamma^*)$ is called polynomially bounded (resp., linearly bounded) if there exists a polynomial (resp., a linear polynomial) $p$ such that, for any two strings $x \in \text{dom}(f)$ and $y \in \Gamma^*$, if $y \in f(x)$ then $|y| \leq p(|x|)$ holds. In this paper, we understand that all function classes are made of polynomially-bounded functions.

Given two alphabets $\Sigma$ and $\Gamma$, a function $f : \Sigma^* \rightarrow \mathcal{P}(\Gamma^*)$ is called length preserving if, for any $x \in \Sigma^*$ and $y \in \Gamma^*$, $y \in f(x)$ implies $|x| = |y|$. 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 writes down any non-blank symbol, and (iv) the tape head can stay only in a blank cell. 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.

To describe npda’s, we take the following specific definition. For any function $f : \Sigma^* \rightarrow \mathcal{P}(\Gamma^*)$, an npda $N$ equipped with a one-way read-only input tape and a write-only output tape that computes $f$ must have the form $(Q, \Sigma, \{\$, \$\}, \Theta, \Gamma, \delta, q_0, \emptyset, Q_{acc}, Q_{rej})$, where $Q$ is a set of inner states, $\Theta$ is a stack alphabet, $q_0 \in Q$ is the initial state, $\emptyset$ is the stack’s bottom marker, and $\delta : (Q - Q_{halt}) \times (\Sigma \cup \{\$\}) \rightarrow \mathcal{P}(\Gamma^* \times (\Gamma \cup \{\$\}))$ is a transition function, where $Q_{halt} = Q_{acc} \cup Q_{rej}$, $\Sigma = \Sigma \cup \{\$, \$\}$, and $\emptyset$ and $\$\$ are respectively left and right endmarkers. It is important to note that, in accordance with the basic setting of [21], we consider only npda’s whose computation paths are all terminate (i.e., reach halting states) in $O(n)$ steps [3], where $n$ refers to input size. We refer to this specific condition regarding execution time as the termination condition.

A function class CFLMV is composed of all (multi-valued partial) functions $f$, each of which maps $\Sigma^*$ to $\mathcal{P}(\Gamma^*)$ for certain alphabets $\Sigma$ and $\Gamma$ 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 a set of all valid outcomes of $N$ on the input $x$. The termination condition imposed on our npda’s obviously leads to an anticipated containment CFLMV $\subseteq$ NPMV. Another class CFLSV is a subclass of CFLMV consisting of single-valued partial functions. In addition, CFLMV and CFLSV are respectively restrictions of CFLMV and CFLSV onto total functions. Those function classes were discussed earlier in [19].

An important concept associated with multi-valued partial functions is refinement [13]. This concept (denoted by $\sqsubseteq_{re} f$) is more suitable to use than set containment ($\subseteq$). Given two multi-valued partial functions $f$ and $g$, we say that $f$ is a refinement of $g$, denoted by $g \sqsubseteq_{re} f$, if (1) $\text{dom}(f) = \text{dom}(g)$ and (2) for every $x$, $f(x) \subseteq g(x)$ (as a set inclusion). We also say that $g$ is refined by $f$. Given two sets $F$ and $G$ of functions, $G \sqsubseteq_{re} F$ if every function $g$ in $G$ can be refined by a certain function $f$ in $F$. When $f$ is additionally single-valued, we call $f$ a single-valued refinement of $g$. 

---

1To describe a multi-valued partial function $f$, the expression "$f : \Sigma^* \rightarrow \Gamma^*$" is customarily used in the literature. However, the current expression "$f : \Sigma^* \rightarrow (\Gamma^*)^*$" matches a fact that the outcome of $f$ on each input string in $\Sigma^*$ is a subset of $\Gamma^*$. 

2If no execution time bound is imposed, a function computed by an npda that nondeterministically produces every binary string on its output tape for each input becomes a valid CFL function; however, this function is no longer an NP function.
3 Basic Operations for Function Classes

Let us discuss our theme of the structural complexity of various classes of (multi-valued partial) functions by exploring fundamental relationships among those function classes. In the course of our discussion, we will unearth an exquisite nature of CFL functions, which looks quite different from that of NP functions.

We begin with demonstrating basic features of CFL functions. First, let us present close relationships between CFL functions and context-free languages. Notice that, for any function \( f \) in CFLMV, both dom(\( f \)) and ran(\( f \)) belong to CFL. It is useful to recall from [21] a notion of \( \varepsilon \)-extension. Assuming that \( \varepsilon \notin \Sigma \), a \( \varepsilon \)-extension of a given string \( x \in \Sigma^* \) is a string \( \tilde{x} \) over \( \Sigma \cup \{\varepsilon\} \) satisfying the following requirement: \( x \) is obtained directly from \( \tilde{x} \) simply by removing all occurrences of \( \varepsilon \) in \( \tilde{x} \). For example, if \( x = 01101 \), then \( \tilde{x} \) may be 011101 or 011111.

The next lemma resembles Nivat’s representation theorem for rational transductionss (see, e.g., [10 Theorem 2]).

**Lemma 3.1** For any function \( f \in \text{CFLMV} \), there exist a language \( A \in \text{CFL} \) and a linear polynomial \( p \) such that, for every pair \( x \) and \( y \), \( y \in f(x) \) iff \( \{i \mid \frac{|y|}{|x|} \in A, (ii) |y| \leq p(|x|) \), and (iii) \( |\tilde{x}| = |\tilde{y}| \) for certain strings \( \tilde{x} \) and \( \tilde{y} \), which are \( \varepsilon \)-extensions of \( x \) and \( y \), respectively.

An immediate application of Lemma 3.1 leads to a functional version of the well-known pumping lemma [2].

**Lemma 3.2** (functional pumping lemma for CFLMV) Let \( \Sigma \) and \( \Gamma \) be any two alphabets and let \( f : \Sigma^* \to P(\Gamma^*) \) be any function in CFLMV. There exist three numbers \( m \in \mathbb{N}^+ \) and \( c, d \in \mathbb{N} \) such that the following condition. Any string \( w \in \Sigma^* \) with \( |w| \geq m \) and any string \( s \in f(w) \) are decomposed into \( w = wxyz \) and \( s = abpq \) such that \( (i) |wxy| \leq m, (ii) |abpq| \geq 1, (iii) |bq| \leq cm + d, \) and \( (iv) abpq^r \in f(w^rxy^rz) \) for any number \( r \in \mathbb{N} \). In the case where \( f \) is further length preserving, the following condition also holds: \( (v) \ |v| = |b| \) and \( \ |y| = |q| \). Moreover, \( (i)-(iv) \) can be replaced by \( (i') |i'| \geq 1 \).

Boolean operations over languages in CFL have been extensively discussed in the past literature (e.g., [16, 21]). Similarly, it is possible to consider Boolean operations that are directly applicable to functions. In particular, we are focused on three typical Boolean operations: union, intersection, and complement. Let us define the first two operations. Given two function classes \( \mathcal{F}_1 \) and \( \mathcal{F}_2 \), let \( \mathcal{F}_1 \cap \mathcal{F}_2 \) (resp., \( \mathcal{F}_1 \cup \mathcal{F}_2 \)) denote a class of all functions \( f \) defined as \( f(x) = f_1(x) \cap f_2(x) \) (set intersection) (resp., \( f(x) = f_1(x) \cup f_2(x) \), set union) over all inputs \( x \), where \( f_1 \in \mathcal{F}_1 \) and \( f_2 \in \mathcal{F}_2 \). Expanding CFLMV(2) in Section 4, we inductively define a \( k \)-conjunctive function class CFLMV(k) as follows: CFLMV(1) = CFLMV and CFLMV(k + 1) = CFLMV(k) \( \cap \) CFLMV for any index \( k \in \mathbb{N}^+ \). Likewise, CFLSV(k) is defined using CFLSV instead of CFLMV.

**Proposition 3.3** Let \( k, m \geq 1 \).

1. CFLMV(max\{k, m\}) \( \subseteq \) CFLMV(k) \( \cup \) CFLMV(m) \( \subseteq \) CFLMV(km).
2. CFLMV(max\{k, m\}) \( \subseteq \) CFLMV(k) \( \cap \) CFLMV(m) \( \subseteq \) CFLMV(k + m).
3. CFLSV(k) \( \subseteq \) CFLSV(k + 1).

Note that Proposition 3.3(3) follows indirectly from a result in [12]. Fenner et al. [5] considered “complements” of NP functions. Likewise, we can discuss complements of CFL functions. Let \( \mathcal{F} \) be any family of functions whose output sizes are upper-bounded by certain linear polynomials. A function \( f \) is in co-\( \mathcal{F} \) if there are a linear polynomial \( p \), another function \( g \in \mathcal{F} \), and a constant \( n_0 \in \mathbb{N} \) such that, for any pair \( (x, y) \) with \( |x| \geq n_0, y \in f(x) \) iff both \( |y| \leq p(|x|) \) and \( y \notin g(x) \) holds. This condition implies that \( f(x) = \Sigma^{\leq p(|x|) + b} - g(x) \) (set difference) for all \( x \in \Sigma^{\geq n_0} \). The finite portion \( \Sigma^{< n_0} \) of inputs is ignored.

The use of set difference in the above definition makes us introduce another class operator \( \ominus \). Given two sets \( \mathcal{F}, \mathcal{G} \) of functions, \( \mathcal{F} \ominus \mathcal{G} \) denotes a collection of functions \( h \) satisfying the following: for certain two functions \( f \in \mathcal{F} \) and \( g \in \mathcal{G} \), \( h(x) = f(x) - g(x) \) (set difference) holds for any \( x \).

In Proposition 3.4, we will give basic properties of functions in co-CFLMV and of the operator \( \ominus \). To describe the proposition, we need to introduce a new function class, denoted by NFAMV, which is defined in a similar way of introducing CFLMV using, in place of npda’s, one-way (one-head) nondeterministic finite automata (or nfa’s, in short) with write-only output tapes, provided that the termination condition (i.e., all computation paths terminate in linear time) must hold.

**Proposition 3.4**

1. co-(co-CFLMV) = CFLMV.
2. co-CFLMV = NFAMV \( \ominus \) CFLMV.
Proposition 3.5  composition.

Proposition 3.6  exists a language 

Proof Sketch.  We will show only (2).  (≥) Let \( f \in \text{NFAM} \cap \text{CFLMV} \) and take two functions \( h \in \text{NFAM} \) and \( g \in \text{CFLMV} \) for which \( f(x) = h(x) - g(x) \) (set difference) for all inputs \( x \in \Sigma^* \). Choose a linear polynomial \( p \) satisfying that, for every pair \((x, y)\), \( y \in h(x) \cup g(x) \) implies \(|y| \leq p(|x|)|\). By the definition of \( f \), it holds that \( f(x) = \Sigma^{|x|} \setminus (g(x) \cup (\Sigma^{|x|} - h(x))) \) for all \( x \). For simplicity, we define \( r(x) = g(x) \cup (\Sigma^{|x|} - h(x)) \) for every \( x \). It thus holds that \( f(x) = \Sigma^{|x|} \setminus r(x) \). It is not difficult to show that \( r \) is in CFLMV.

(≤) Let \( f \in \text{co-CFLMV} \). There are a linear polynomial \( p \) and a function \( g \in \text{CFLMV} \) satisfying that \( f(x) = \Sigma^{|x|} \setminus g(x) \) for all \( x \). We set \( h(x) = \Sigma^{|x|} \setminus r(x) \). Since \( h \in \text{NFAM} \), we conclude that \( f \) belongs to \( \text{NFAM} \cap \text{CFLMV} \). \( \square \)

Another basic operation used for functions is \textit{functional composition}. The functional composition \( f \circ g \) of two functions \( f \) and \( g \) is defined as \( (f \circ g)(x) = \bigcup_{y \in g(x)} f(y) \) for every input \( x \). For two function classes \( F \) and \( G \), let \( F \circ G = \{ f \circ g \mid f \in F, g \in G \} \). In particular, we inductively define \( \text{CFLSV}^{(1)} = \text{CFLSV} \) and \( \text{CFLSV}^{(k+1)} = \text{CFLSV} \circ \text{CFLSV}^{(k)} \) for each index \( k \geq 1 \). For instance, the function \( f(x) = \{xx\} \) defined for any \( x \in \Sigma^* \) belongs to \( \text{CFLSV}^{(2)} \). This fact yields, e.g., \( \text{CFLSV}^{(2)} \subseteq \text{CFLSV}^{(4)} \). Unlike \( \text{NP} \) function classes (such as, \( \text{NPSV} \) and \( \text{NPMV} \)), \( \text{CFLSV} \) (and therefore \( \text{CFLSV} \) and \( \text{CFLMV} \)) is not closed under functional composition.

\begin{proposition}  \text{CFLSV}_t \not\subseteq \text{CFLSV}_t^{(2)}.  \text{(Also for} \ \text{CFLSV} \ \text{and} \ \text{CFLMV}. \text{)} \end{proposition}

\begin{proof}  Let \( \Sigma = \{0, 1, \sharp\} \) be our alphabet and define \( f_{\text{dup}}(x) = \{xx\} \) for any input \( x \in \{0, 1\}^* \) and \( f_{\text{dup}_y}(x) = \{\} \) for any other inputs \( x \). It is not difficult to show that \( f_{\text{dup}_y} \in \text{CFLSV}_t^{(2)} \). To show that \( f_{\text{dup}} \not\in \text{CFLSV}_t \), we assume that \( f_{\text{dup}} \in \text{CFLSV}_t \). Let \( D U P_2 \) be a “marked” version of \( D U P_2 \) (duplication), defined as \( D U P_2 = \{xx \mid x \in \{0, 1\}^*\} \). It holds that, for every \( w \not= \lambda \), \( w \in \text{ran}(f_{\text{dup}_y}) \) iff there is a string \( x \) such that \( w \in f_{\text{dup}_y}(x) \) iff \( w \in D U P_2 \). Thus, \( D U P_2 \cup \{\lambda\} = \text{ran}(f_{\text{dup}_y}) \). Note that \( D U P_2 \in \text{CFL} \) since \( f_{\text{dup}_y} \in \text{CFLSV}_t \). However, it is well-known that \( D U P_2 \not\subseteq \text{CFL} \). This leads to a contradiction. Therefore, we conclude that \( f_{\text{dup}} \not\in \text{CFLSV}_t \). \( \square \)

To examine the role of functions in a process of recognizing a given language, a \textit{//-advice operator}, defined by K{"o}bler and Thierauf [9], is quite useful. Given a class \( F \) of functions, a language \( L \) is in \( \text{CFL} // F \) if there exists a language \( B \subseteq \text{CFL} \) and a function \( h \in F \) satisfying \( L = \{x \mid \exists y \in h(x) \text{ s.t. } [y] \subseteq B\} \). Analogously, \( \text{REG} // F \) is defined using \( \text{REG} \) instead of \( \text{CFL} \). This operator naturally extends a \textit{\text{//}-advice operator} of [15] [17].

In the polynomial-time setting, it holds that \( \text{NP} \cap \text{co-NP} = \text{P} // \text{NPSV}_t [9] \). A similar equality, however, does not hold for CFL functions.

\begin{proposition} \begin{enumerate}  \item \( \text{REG} // \text{NFASV}_t \not\subseteq \text{CFL} \) and \( \text{CFL} \not\subseteq \text{REG} // \text{NFAMV} \).  \item \( \text{REG} // \text{NFASV}_t \) is closed under complement but \( \text{REG} // \text{NFAMV} \) is not.  \item \( \text{CFL} \cap \text{co-CFL} \not\subseteq \text{REG} // \text{CFLSV}_t \). \end{enumerate} \end{proposition}

\begin{proof}  (1) Note that the language \( D U P_2 = \{x\#x \mid x \in \{0, 1\}^*\} \) (duplication) falls into \( \text{REG} // \text{NFASV}_t \) by setting \( h(x\#y) = y \) and \( B = \{[x] \mid x \in \{0, 1\}^*\} \). The key idea is the following claim. (*

A language \( L \) is in \( \text{REG} // \text{NFAMV} \) iff \( L \) is recognized by a certain \textit{one-way two-head (non-sensing) nfa} (or an \textit{nfa}(2), in short) with \( \lambda \)-moves. See a survey, e.g., [7], for this model. Now, consider \( L_{pal} = \{x\#x^R \mid x \in \{0, 1\}^*\} \) (palindromes). Since \( L_{pal} \) cannot be recognized by any \textit{nfa}(2), it follows that \( L_{pal} \not\in \text{REG} // \text{NFAMV} \).

(2) The non-closure property of \( \text{REG} // \text{NFAMV} \) follows from a fact that the class of languages recognized by \textit{nfa}(2)’s is not closed under complement. Use the above claim (*) to obtain the desired result. \( \square \)

Before closing this section, we exhibit a simple structural difference between languages and functions. It is well-known that all languages over the unary alphabet \{1\} in \text{CFL} belong to \text{REG}. On the contrary, there is a function \( f : \{1\}^* \to \{0, 1\}^* \) such that \( f \) is in \text{CFLMV} but not in \text{NFAMV}.

4 Oracle Computation and Two Relativizations

\textit{Oracle computation} is a natural extension of ordinary stand-alone computation by providing external information by way of \textit{query-and-answer} communication. Such oracle computation can realize various forms
of relativizations, including many-one and Turing relativizations and thus introduce relativized languages and functions. By analogy with relativized NP functions (e.g., \cite{14}, let us consider many-one and Turing relativizations of CFL functions. The first notion of \textit{many-one relativization} was discussed for languages in automata theory \cite{8,21} and we intend to extend it to CFL functions. Given any language \(A\) over alphabet \(\Gamma\), a function \(f : \Sigma^* \to P(\Gamma^*)\) is in CFLMV\(^A\) (or CFLMV\(_m\)(\(A\))) 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, for any \(x \in \Sigma^*\), (i) along any accepting computation path \(p\) of \(M\) on \(x\) (notationally, \(p \in \text{ACC}_M(x)\)), \(M\) produces a query string \(y_p\) on the query tape as well as an output string \(z_p\) on the output tape and (ii) \(f\) equals the set \(\{z_p \mid y_p \in A, p \in \text{ACC}_M(x)\}\). Such an \(M\) is referred to as an oracle npda. Given any language family \(C\), we further set CFLMV\(_m\) (or CFLMV\(_m\)(\(C\))) to be \(\bigcup_{A \in C} \text{CFLMV}^A\(_m\).\) Similarly, we define CFLSV\(_m\) and CFLSV\(_m^A\) by additionally demanding the size of each output string set is at most 1. Using CFLSV\(_m\), a relativized language family CFL\(_m^A\) defined in \cite{21} can be expressed as CFL\(_m^A\) = \(\{L \mid \eta \in \text{CFLSV}^A\(_m\}\}.\)

\textbf{Lemma 4.1} 1. CFLMV\(_m^\text{REG}\) = CFLMV and CFLSV\(_m^\text{REG}\) = CFLSV.

\textbf{Proposition 4.2} 1. REG//CFLSV\(_m^A\) \(\subseteq\) CFL\(_m^A\)\(2\) \(\cap\) co-CFL\(_m^A\)\(2\).

2. CFL//CFLSV\(_m^A\) \(\subseteq\) CFL\(_m^A\)\(3\).

\textbf{Proof Sketch.} We will prove only (2). For convenience, we write \([x,y]^T\) for \([y]\) in this proof. Let CFL\(_m^A\)\(1\) = CFL\(_m^A\), and CFL\(_m^A\)\(k+1\) = CFL\(_m^A\)\(k\) for every index \(k \geq 1\) \cite{21}. Let \(L \in\) CFL//CFLSV\(_m^A\), and take an npda \(M\) and a function \(h \in\) CFLSV\(_m^A\) for which \(L = \{x \mid M \text{ accepts } (x,h(x))^T\}\). Let \(M'\) be an npda with an output tape computing \(h\). An oracle npda \(N\) is also defined as follows. On input \(x\), \(N\) simulates \(M\) on \(x\) and nondeterministically produces \([\tilde{x},\tilde{y}]^T\) on its query tape when \(M'\) outputs \(y\), where \(\tilde{x}\) and \(\tilde{y}\) are appropriate \(k\)-extensions of \(x\) and \(y\), respectively. An oracle \(A\) receives a \(k\)-extension \([\tilde{x},\tilde{y}]^T\) and decides whether \(M\) accepts \([x,y]^T\) by removing all \(\tilde{s}\). Clearly, \(L\) belongs to CFL\(_m^A\) via \(N\). Define another npda \(N_1\). On input \(w = [\tilde{x},\tilde{y}]^T\), \(N_1\) simulates \(M\) on \(z^T\) by guessing \(z\) symbol by symbol. At the same time, it writes \([\tilde{y}',\tilde{z}]^T\) on a query tape and accepts \(w\) exactly when \(M\) enters an accepting state. Let \(B = \{[\tilde{y}',\tilde{z}]^T \mid y = z\}\). Note that \(B\) is in CFL\(_m^A\). Hence, \(A\) is in CFL\(_m^B\) \(\subseteq\) CFL\(_m^{[2]}\), and thus \(L\) is in CFL\(_m^{[3]}\). Since CFL\(_m^{[3]}\) = CFL\(_m^{[3]}\) \cite{21}, the desired conclusion follows. \(\Box\)

The second relativization is \textit{Turing relativization}. A multi-valued partial function \(f\) belongs to CFLMV\(_T\) (or CFLMV\(_T\)(\(A\))) if there exists an oracle npda \(M\) having \(\text{three} \ \text{inner} \ \text{states} \ \{q_{query}, q_{yes}, q_{no}\}\) that satisfies the following three conditions: on each input \(x\), (i) if \(M\) enters a query state \(q_{query}\), then a valid string, \(s\), written on the query tape is sent to \(A\) and, automatically, \(the \ \text{content} \ \text{of} \ \text{the} \ \text{query} \ \text{tape} \ \text{becomes} \ \text{blank} \ \text{and} \ \text{the} \ \text{tape} \ \text{head} \ \text{returns} \ \text{to} \ \text{the} \ \text{start} \ \text{cell}\); (ii) oracle \(A\) sets \(M\)'s inner state to \(q_{yes}\) if \(s \in A\) and \(q_{no}\) otherwise, and (iii) all computation paths of \(M\) terminate in time \(O(n)\) \(\text{no matter} \ \text{what} \ \text{oracle} \ \text{is} \ \text{used}\). Obviously, CFLMV\(_T\)\(A\) = CFLMV\(_T\)(\(A\)) holds for any oracle \(A\). Define CFLMV\(_C\)\(_T\) (or CFLMV\(_T\)(\(C\))) to be the union \(\bigcup_{A \in C} \text{CFLMV}^A\(_T\) for a given language family \(C\).

Analogously to the well-known NPMV hierarchy, composed of \(\Sigma_k^P\)MV and \(\Pi_k^P\)MV for \(k \in \mathbb{N}^+\) \cite{14}, we inductively define CFLMV\(_T\)\(k\) = CFLMV, \(\Pi_k^CFLMV = \text{co-} \Sigma_k^CFLMV\), and \(\Sigma_k^{CFLMV} = \text{CFLMV}(\Pi_k^{CFLMV})\) for every index \(k \geq 1\). In a similar fashion, we define CFLSV\(_T\)\(k\) using CFLSV\(_T\)\(k\) in place of CFLMV\(_T\)\(k\). The above CFLMV\(_T\) hierarchy is useful to scaling the computational complexity of given functions. For example, the function \(f(w) = \{x \in \{0,1\}^\star \mid \exists u, v : w = u|xv|\}\) for every \(w \in \{0,1\}^\star\) belongs to \(\Sigma_k^{CFLMV}\). Moreover, it is possible to show that CFLMV \(\cup\) co-CFLMV \(\subseteq\) CFLMV \(\subseteq\) CFL\(_m^CFLMV\) for every \(k \geq 1\).

\textbf{Proposition 4.3} Each function in \(\bigcup_{k \in \mathbb{N}^+} \Sigma_k^{CFLSV}\) can be computed by an appropriate \(O(n)\) space-bounded multi-tape deterministic Turing machine.

\textbf{Proof Sketch.} It is known in \cite{21} that \(\Sigma_k^{CFL} \subseteq \text{DSPACE}(O(n))\) for every \(k \geq 1\). It therefore suffices to show that, for any fixed language \(A \in \text{DSPACE}(O(n))\), every function \(f\) in CFLSV\(_T\)\(A\) can be computed using \(O(n)\) space. This is done by a direct simulation of \(f\) on a multi-tape Turing machine. \(\Box\)

For \(k \geq 3\), it is possible to give the exact characterization of REG//\(\Sigma_k^{CFLSV}\). This makes a sharp contrast with Proposition 3\((\text{iii})\) 3\((\text{iii})\).

\textbf{Proposition 4.4} For every index \(k \geq 3\), \(\Sigma_k^{CFL} \cap \Pi_k^{CFL} = \text{REG//} \Sigma_k^{CFLSV}\).

\textbf{Proof Sketch.} By extending the proof of Proposition 3\((\text{iii})\) 3\((\text{iii})\), it is possible to show that \(\Sigma_k^{CFL} \cap \Pi_k^{CFL} \subseteq\)
**Proposition 5.1**

CFLSV \( \subseteq \Sigma_k^{CFL} \). Similarly to Proposition 3.3, it holds that CFLSV \( \subseteq \Sigma_k^{CFL} \) is closed under complementation. It thus suffices to show that CFLSV \( \subseteq \Sigma_k^{CFL} \). This can be done by a direct simulation.

**Lemma 4.5**

Let \( e \geq 2 \) and \( k \geq 1 \). \( \Sigma_k^{CFL} \subseteq \Sigma_{k+1}^{CFL} \Rightarrow \Sigma_k^{CFL} = \Sigma_{k+1}^{CFL} \Rightarrow \Sigma_{k+1}^{CFL} \subseteq \Sigma_k^{CFL} \).

**Proof Sketch.**

Assuming \( \Sigma_k^{CFL} = \Sigma_{k+1}^{CFL} \), let us take any language \( A \in \Sigma_{k+1}^{CFL} \) and consider \( \eta_A \). It is not difficult to show that, for any index \( d \in \mathbb{N}^+ \), \( A \in \Sigma_k^{CFL} \) iff \( \eta_A \in \Sigma_k^{CFL} \). Thus, \( \eta_A \in \Sigma_{k+1}^{CFL} = \Sigma_k^{CFL} \). This implies that \( A \in \Sigma_k^{CFL} \). Next, assume that \( \Sigma_k^{CFL} = \Sigma_{k+1}^{CFL} \). It is proven in [21] that \( \Sigma_k^{CFL} = \Sigma_{k+1}^{CFL} \) iff \( \Sigma_{k+2}^{CFL} = \Sigma_{k+1}^{CFL} \) for all \( e \geq 1 \). Hence, for every \( e \geq 2 \), we obtain \( \Sigma_k^{CFL} = \Sigma_{k+e-1}^{CFL} \), which is equivalent to \( \Pi_2^{CFL} = \Pi_2^{CFL} \) for \( k+e-1 \). It then follows that \( \Sigma_{k+e}^{CFL} = \Sigma_{k+e}^{CFL} \) for \( e \geq 1 \). Hence, \( \Sigma_{k+e}^{CFL} \subseteq \Sigma_{k+e}^{CFL} \).

Regarding refinement, from the proof of [8, Theorem 3] follows NFAMV \( \subseteq_{ref} \) NFASV. This result leads to NFAMV \( \circ \) CFLSV \( \subseteq_{ref} \) CFLSV in [10]. By a direct simulation, nevertheless, it is possible to show that CFLMV \( \subseteq_{ref} \) CFLSV for every \( k \geq 1 \). Lemma 4.3 together with this fact leads to the following consequence.

**Proposition 4.6**

Let \( k \geq 2 \). If \( \Sigma_k^{CFL} = \Sigma_{k+1}^{CFL} \), then \( \Sigma_{k+1}^{CFL} \subseteq \Sigma_k^{CFL} \).

Recently, it was shown in [22] that CFLMV \( \subseteq_{ref} \) CFLSV holds. However, it is not known if this can be extended to every level of the CFLMV hierarchy.

## 5 Optimization Functions

An optimization problem is to find an optimal feasible solution that satisfy a given condition. Krentel [11] studied the complexity of those optimization problems. Analogously to OptP of Krentel, we define OptCFL as a collection of single-valued total functions \( f : \Sigma^* \rightarrow \Gamma^* \) such that there exists an npda \( M \) and an \( opt \in \{ \text{maximum, minimum} \} \) for which, for every string \( x \in \Sigma^* \), \( f(x) \) denotes the \( opt \) output string of \( M \) on input \( x \) along an appropriate accepting computation path, assuming that \( M \) must have at least one accepting computation path. Here, we use the dictionary (or alphabetical) order \( \prec \) over \( \Gamma^* \) (e.g., \( abbc \prec abcc \) and \( ab \prec aba \)) instead of the lexicographic order to compensate the npda’s limited ability of comparing two strings from left to right. For example, the function \( f(x) = \max \{ PAL_{sub}(w) \} \) for \( w \in \{ \text{0, 1} \} \), where \( PAL_{sub} \) is defined in Section 4 is a member of OptCFL. It holds that CFLMV \( \subseteq_{ref} \) OptCFL.

**Proposition 5.1**

\( \text{CFLSV} \subseteq \text{OptCFL} \subseteq \Sigma_2^{CFL} \).

The first part of Proposition 5.1 comes from a fact that the function \( f(w) = \max \{ g(w) \} \) where \( g(w) = \{ \lambda \} \cup \{ x, y, z \mid x = x_1 x_2 x_3 \# y_1 y_2 y_3, x_i \in \{ 1, 2, 3 \} \} \), is in OptCFL but not in CFLSV. This latter part is proven by applying the functional pumping lemma.

Note that, in the polynomial-time setting, a much sharper upper-bound of OptP \( \subseteq \Sigma_3^{SV} \) is known. Similarly to OptCFL, let us define OptNFA\( _{EL} \) using nfa’s \( M \) instead of npda’s with an extra condition that \( M(x) \) outputs only strings of the equal length. This new class is located within \( \Sigma_2^{CFL} \).

**Proposition 5.2**

\( \text{OptNFA}_{EL} \subseteq \Sigma_2^{CFL} \).

**Proof Sketch.**

Let \( f \in \text{OptNFA}_{EL} \) and take an underlying nfa \( N \) that forces \( f(x) \) to equal \( \max \{ N(x) \} \) for every \( x \). Define an oracle npda \( M_1 \) to simulate \( N \) on \( x \) and output, say, \( y \). Simultaneously, query \( y \# x^R \) using a stack wisely. If its answer is 1, enter an accepting state; otherwise, reject. Make another npda \( M_2 \) that receives \( y \# x^R \), simulate \( N^R \) on \( x^R \), and compare its outcome with \( y^R \), where the notation \( N^R \) refers to an nfa that reverses the computation of \( N \).

**Proposition 5.3**

1. \( \text{CFL} \cup \text{co-CFL} \subseteq \text{REG} \cup \text{OptCFL} \subseteq \Sigma_1^{CFL} \cap \Pi_2^{CFL} \).
2. \( \text{CFL} \cup \text{OptCFL} \subseteq \Sigma_0^{SV} \).

**Proof Sketch.**

We will show only (1). Note that REG/\( \text{OptCFL} \) is closed under complementation. Let \( L \in \text{CFL} \) and take an npda \( M \) recognizing \( L \). Define \( N_1 \) as follows. On input \( x \), guess a bit \( b \). If \( b = 0 \), then output 0 in an accepting state. Otherwise, simulate \( M \) on \( x \) and output 1 along only accepting computation paths of \( M \). Let \( h(x) = \max \{ N_1(x) \} \) for all \( x \). It follows that \( L = \{ x \mid h(x) \neq \emptyset \} \). This proves that \( L \in \text{REG/\( \text{OptCFL} \)} \). Next, let \( L \in \text{REG/\( \text{OptCFL} \)} \). Since \( \text{OptCFL} \subseteq \Sigma_0^{SV} \), we obtain
$L \in \text{REG}//\Sigma_4^{\text{CFL}}\text{-SV}_1$. By Proposition 4.4, this implies that $L$ belongs to $\Sigma_4^{\text{CFL}} \cap \Pi_4^{\text{CFL}}$. \hfill \Box

References

[1] Aho, A.V., Hopcroft, J.E., Ullman, J.D.: A general theory of translation. Math. Systems Theory, 3, 193–221 (1967)
[2] Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase-structure grammars. Z. Phonetik Sprachwiss. Kommunik., 14, 143–172 (1961)
[3] Choffrut, C., Culik, K.: Properties of finite and pushdown transducers. SIAM J. Comput., 12, 300–315 (1983)
[4] Evey, R.J.: Application of pushdown-store machines. In: Proc. 1963 Fall Joint Computer Conference, AFIPS Press, pp.215–227, 1963
[5] Fenner, S.A., Homer, S., Ogihara, M., Selman, A.L.: Oracles that compute values. SIAM J. Comput., 26, 1043–1065 (1997)
[6] Fisher, P.C.: 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
[7] Holzer, M., Kutrib, M., Malcher, A.: Multi-head finite automata: characterizations, concepts and open problems. In: Proc. of the Workshop on The Complexity of Simple Programs, number 1 in EPTCS, pp.93–107, 2008
[8] Kobayashi, K.: Classification of formal languages by functional binary transductions. Inform. Control, 15, 95–109 (1969)
[9] Kötler, J., Thierauf, T.: Complexity-restricted advice functions. SIAM J. Comput., 23, 261–275 (1994)
[10] Konstantinidis, S., Santean, N., Yu, S.: Representation and uniformization of algebraic transductions. Acta Inform., 43, 395–417 (2007)
[11] Krentel, M.: The complexity of optimization problems. J. Comput. System Sci., 36, 490–509 (1988)
[12] Liu, L.Y., Weiner, P.: An infinite hierarchy of intersections of context-free languages. Math. Systems Theory, 7, 185–192 (1973)
[13] Selman, A.L.: A taxonomy of complexity classes of functions. J. Comput. System Sci., 48, 357–381 (1994)
[14] Selman, A.L.: Much ado about functions. In: Proc. of the 11th Annual IEEE Conference on Computational Complexity, pp.198–212, 1996
[15] Tadaki, K., Yamakami, T., Lin, J. C. H.: Theory of one-tape linear-time Turing machines. Theor. Comput. Sci., 411, 22–43 (2010)
[16] Wotschke, D.: Nondeterminism and Boolean operations in pda’s. J. Comp. System Sci., 16, 456–461 (1978)
[17] Yamakami, T.: Swapping lemmas for regular and context-free languages. Available at arXiv:0808.4122 (2008)
[18] Yamakami, T.: The roles of advice to one-tape linear-time Turing machines and finite automata. Int. J. Found. Comput. Sci., 21, 941–962 (2010)
[19] Yamakami, T.: Immunity and pseudorandomness of context-free languages. Theor. Comput. Sci., 412, 6432–6450 (2011)
[20] Yamakami, T.: Pseudorandom generators against advised context-free languages. See arXiv:0902.2774 (2009)
[21] Yamakami, T.: Oracle pushdown automata, nondeterministic reducibilities, and the hierarchy over the family of context-free languages. In: Proc. of SOFSEM 2014, V. Geffert et al. (eds.), LNCS, vol. 8327, pp. 514–525 (2014). A complete version appears at arXiv:1303.1717
[22] Yamakami, T.: Not all multi-valued partial CFL functions are refined by single-valued functions. In: Proc. of IFIP TCS 2014, J. Diaz et al. (eds), LNCS vol. 8705, pp. 136–150 (2014)
[23] Yamakami, T.: Structural complexity of multi-valued partial functions computed by nondeterministic pushdown automata. In: Proc. of the 15th Italian Conference of Theoretical Computer Science (ICTCS 2014), CEUR Workshop Proceedings, vol.1231, pp.225–236, 2014.