Structural complexity of multi-valued partial functions computed by nondeterministic pushdown automata
(Extended Abstract)

Tomoyuki Yamakami

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

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 $\mathrm{NPSV_t}$ (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 $PAL_{sub}(w) = \{x \in \{0,1\}^* \mid \exists u, v\,[w = uxv], x = x^R\}$ 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 $\mathrm{CFLSV_t}$ 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 $\mathrm{CFLSV_t}$ 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 $\mathrm{CFL} \cap \text{co-CFL}$ 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 λ-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 *many-one and Turing CFL-reducibilities* among languages [21]. We use the Turing relativization to introduce a hierarchy of function classes $\Sigma_k^{\mathrm{CFL}}\mathrm{MV}$, $\Pi_k^{\mathrm{CFL}}\mathrm{MV}$, and $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}$ for each level $k \geq 1$, in which the first Σ-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 kth level, every function in $\Sigma_{k+1}^{\mathrm{CFL}}\mathrm{MV}$ has a refinement in $\Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV}$ for every index $k \geq 2$.

Every nondeterministic machine with an output tape can be naturally seen as a process of generating a set of feasible "solutions" of each instance. Among those solutions, it is useful to study the complexity of finding "optimal" solutions. This gives rise to *optimization functions.* Earlier, Krentel [11] discussed properties of OptP that is composed of polynomial-time optimization functions. Here, we are focused on similar functions induced by npda's with output tapes. We denote by OptCFL a class of those optimization CFL functions. This function class is proven to be located between $\mathrm{CFLSV_t}$ and $\Sigma_4^{\mathrm{CFL}}\mathrm{SV_t}$. Moreover, we show the class separation between $\mathrm{CFLSV_t}$ and OptCFL.

To see a role of functions during a process of recognizing languages, Köbler and Thierauf [9] introduced a *//-advice operator* by generalizing the Karp-Lipton /-advice operator, and they argued the computational complexity of languages in $\mathrm{P}//\mathcal{F}$ and $\mathrm{NP}//\mathcal{F}$ induced by any given function class $\mathcal{F}$. Likewise, we discuss the complexity of $\mathrm{REG}//\mathcal{F}$ and $\mathrm{CFL}//\mathcal{F}$ when $\mathcal{F}$ are various subclasses of CFL functions, particularly $\mathrm{CFLSV_t}$ and OptCFL.

All omitted or abridged proofs, because of the page limit, will appear shortly in a complete version of this paper.

2 A Starting Point

Formal Languages. Let $\mathbb{N}$ be the set of all *natural numbers* (i.e., nonnegative integers) and set $\mathbb{N}^+ = \mathbb{N} - \{0\}$. Throughout this paper, we use the term " polynomial" to mean polynomials on $\mathbb{N}$ with nonnegative 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 *set difference* $\{x \mid x \in A, x \notin B\}$. Given any set A, $\mathcal{P}(A)$ denotes the *power set* of A. A set Σ^k (resp., $\Sigma^{\leq k}$), where $k \in \mathbb{N}$, consists only of strings of length k (resp., at most k). Here, we set $\Sigma^* = \bigcup_{k\in\mathbb{N}} \Sigma^k$. The *empty string* is always denoted λ. Given any language A over Σ, its *complement* is $\Sigma^* - A$, which is also denoted by $\overline{A}$ as long as Σ is clear from the context.

We adopt a *track notation* $[\begin{smallmatrix}x\\y\end{smallmatrix}]$ from [15]. For two symbols σ and τ, $[\begin{smallmatrix}\sigma\\\tau\end{smallmatrix}]$ 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, $[\begin{smallmatrix}x\\y\end{smallmatrix}]$ denotes a string $[\begin{smallmatrix}x_1\\y_1\end{smallmatrix}][\begin{smallmatrix}x_2\\y_2\end{smallmatrix}]\cdots[\begin{smallmatrix}x_n\\y_n\end{smallmatrix}]$. Whenever $|x| \neq |y|$, we follow a convention introduced in [15]: if $|x| < |y|$, then $[\begin{smallmatrix}x\\y\end{smallmatrix}]$ actually means $[\begin{smallmatrix}x\#^m\\y\end{smallmatrix}]$, where $m = |y|-|x|$ and $\#$ is a designated new symbol. Similarly, when $|x| > |y|$, the notation $[\begin{smallmatrix}x\\y\end{smallmatrix}]$ expresses $[\begin{smallmatrix}x\\y\#^m\end{smallmatrix}]$ with $m = |x| - |y|$.

As our basic computation model, we use *one-way one-head nondeterministic pushdown automata* (or npda's, in short) allowing λ-moves (or λ-transitions) of their input-tape heads. 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 by CFL(k), is defined to be $\{\bigcap_{i=1}^{k} A_i \mid A_1, A_2, \ldots, A_k \in \mathrm{CFL}\}$ (see, e.g., [19]).

Given any language A (used as an oracle), CFL_T^A (or $\mathrm{CFL}_T(A)$) 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 $\mathrm{CFL}_T^{\mathcal{C}}$ (or $\mathrm{CFL}_T(\mathcal{C})$) for language family $\mathcal{C}$ to denote the union $\bigcup_{A\in\mathcal{C}} \mathrm{CFL}_T^A$. Its deterministic version is expressed as $\mathrm{DCFL}_T^{\mathcal{C}}$. The *CFL hierarchy* $\{\Delta_k^{\mathrm{CFL}}, \Sigma_k^{\mathrm{CFL}}, \Pi_k^{\mathrm{CFL}} \mid k \in \mathbb{N}^+\}$ is composed of classes $\Delta_1^{\mathrm{CFL}} = \mathrm{DCFL}$, $\Sigma_1^{\mathrm{CFL}} = \mathrm{CFL}$, $\Pi_k^{\mathrm{CFL}} = \text{co-}\Sigma_k^{\mathrm{CFL}}$, $\Delta_{k+1}^{\mathrm{CFL}} = \mathrm{DCFL}_T(\Pi_k^{\mathrm{CFL}})$, and $\Sigma_{k+1}^{\mathrm{CFL}} = \mathrm{CFL}_T(\Pi_k^{\mathrm{CFL}})$ 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[1] Σ^* to $\mathcal{P}(\Gamma^*)$ for certain alphabets Σ and Γ. When f is single-valued, we often write $f(x) = y$ instead of $y \in f(x)$. Associated with f, $\mathrm{dom}(f)$ denotes the *domain* of f, defined to be $\{x \in \Sigma^* \mid f(x) \neq \emptyset\}$. If $x \notin \mathrm{dom}(x)$, then $f(x)$ is said to be *undefined.* The *range* $\mathrm{ran}(f)$ 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 χ_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* η_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 $f : \Sigma^* \to \mathcal{P}(\Gamma^*)$ is called *polynomially bounded* (resp., *linearly bounded*) if there exists a polynomial (resp., a linear polynomial) p such that, for any two strings $x \in \mathrm{dom}(f)$ and $y \in \Gamma^*$, if

[1] To describe a multi-valued partial 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 Σ^* is a subset of Γ^*.

$y \in f(x)$ then $|y| \leq p(|x|)$ holds. In this paper, we understand that *all function classes are made of polynomially-bounded functions.* Given two alphabets Σ and Γ, a function $f : \Sigma^* \to \mathcal{P}(\Gamma^*)$ 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 still 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 given function $f : \Sigma^* \to \mathcal{P}(\Gamma^*)$, 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 $(Q, \Sigma, \{¢, \$\}, \Theta, \Gamma, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$, where Q is a set of inner states, Θ is a stack alphabet, q_0 ($\in Q$) is the initial state, Z_0 ($\in \Theta$) is the stack's bottom marker, and $\delta : (Q - Q_{halt}) \times (\check{\Sigma} \cup \{\lambda\}) \times \Theta \to \mathcal{P}(Q \times \Theta^* \times (\Gamma \cup \{\lambda\}))$ is a transition function, where $Q_{halt} = Q_{acc} \cup Q_{rej}$, $\check{\Sigma} = \Sigma \cup \{¢, \$\}$, and ¢ 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*,[2] 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 Σ^* to $\mathcal{P}(\Gamma^*)$ for certain alphabets Σ and Γ 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, $\mathrm{CFLMV_t}$ and $\mathrm{CFLSV_t}$ 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 $\sqsubseteq_{ref}$) 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 $g \sqsubseteq_{ref} f$, if (1) $\mathrm{dom}(f) = \mathrm{dom}(g)$ 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 $\mathcal{F}$ and $\mathcal{G}$ of functions, $\mathcal{G} \sqsubseteq_{ref} \mathcal{F}$ if every function g in $\mathcal{G}$ can be refined by a certain function f in $\mathcal{F}$. When f is additionally single-valued, we call f a *single-valued refinement* of g.

[2] If 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 $\mathrm{dom}(f)$ and $\mathrm{ran}(f)$ belong to CFL. It is useful to recall from [21] a notion of $\natural$-extension. Assuming that $\natural \notin \Sigma$, a $\natural$-*extension* of a given string $x \in \Sigma^*$ is a string $\tilde{x}$ over $\Sigma \cup \{\natural\}$ satisfying the following requirement: x is obtained directly from $\tilde{x}$ simply by removing all occurrences of $\natural$ in $\tilde{x}$. For example, if $x = 01101$, then $\tilde{x}$ may be $01\natural 1\natural 01$ or $\natural 0110\natural\natural 1$. The next lemma resembles Nivat's representation theorem for rational transductions (see, e.g., [10, Theorem 2]).

Lemma 1. *For any function $f \in$ CFLMV, there exist a language $A \in$ CFL and a linear polynomial p such that, for every pair x and y, $y \in f(x)$ iff (i) $[\,{\tilde{x} \atop \tilde{y}}\,] \in A$, (ii) $|y| \leq p(|x|)$, and (iii) $|\tilde{x}| = |\tilde{y}|$ for certain strings $\tilde{x}$ and $\tilde{y}$, which are $\natural$-extensions of x and y, respectively.*

An immediate application of Lemma 1 leads to a functional version of the well-known pumping lemma [2].

Lemma 2. (functional pumping lemma for CFLMV) *Let Σ and Γ be any two alphabets and let $f : \Sigma^* \to \mathcal{P}(\Gamma^*)$ be any function in* CFLMV*. There exist three numbers $m \in \mathbb{N}^+$ and $c, d \in \mathbb{N}$ that satisfy the following condition. Any string $w \in \Sigma^*$ with $|w| \geq m$ and any string $s \in f(w)$ are decomposed into $w = uvxyz$ and $s = abpqr$ such that (i) $|vxy| \leq m$, (ii) $|vybq| \geq 1$, (iii) $|bq| \leq cm + d$, and (iv) $ab^ipq^ir \in f(uv^ixy^iz)$ for any number $i \in \mathbb{N}$. In the case where f is further length preserving, the following condition also holds: (v) $|v| = |b|$ and $|y| = |q|$. Moreover, (i)–(ii) can be replaced by (i') $|bq| \geq 1$.*

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 $\mathcal{F}_1 \wedge \mathcal{F}_2$ (resp., $\mathcal{F}_1 \vee \mathcal{F}_2$) 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 1, we inductively define a *k-conjunctive function class* CFLMV(k) as follows: CFLMV(1) = CFLMV and CFLMV($k+1$) = CFLMV(k) $\wedge$ CFLMV for any index $k \in \mathbb{N}^+$. Likewise, CFLSV(k) is defined using CFLSV instead of CFLMV.

Proposition 3. *Let $k, m \geq 1$.*

1. CFLMV($\max\{k, m\}$) $\subseteq$ CFLMV(k) $\vee$ CFLMV(m) $\subseteq$ CFLMV(km).

2. $\mathrm{CFLMV}(\max\{k,m\}) \subseteq \mathrm{CFLMV}(k) \wedge \mathrm{CFLMV}(m) \subseteq \mathrm{CFLMV}(k+m)$.
3. $\mathrm{CFLSV}(k) \subsetneq \mathrm{CFLSV}(k+1)$.

Note that Proposition 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 $f(x) = \Sigma^{\leq a|x|+b} - g(x)$ (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 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 4. *1.* co-(co-CFLMV) = CFLMV.
2. co-CFLMV = NFAMV $\ominus$ CFLMV.
3. CFLMV $\ominus$ CFLMV = CFLMV $\wedge$ co-CFLMV.
4. CFLMV $\neq$ co-CFLMV. *The same holds for* $\mathrm{CFLMV_t}$.

Proof Sketch. We will show only (2). ($\supseteq$) Let $f \in$ NFAMV $\ominus$ CFLMV and take two functions $h \in$ NFAMV and $g \in$ 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 $f(x) = \Sigma^{\leq p(|x|)} - (g(x) \cup (\Sigma^{\leq p(|x|)} - h(x)))$ for all x. For simplicity, we define $r(x) = g(x) \cup (\Sigma^{\leq p(|x|)} - h(x))$ for every x. It thus holds that $f(x) = \Sigma^{\leq p(|x|)} - r(x)$. It is not difficult to show that r is in CFLMV.

($\subseteq$) Let $f \in$ co-CFLMV. There are a linear polynomial p and a function $g \in$ CFLMV satisfying that $f(x) = \Sigma^{\leq p(|x|)} - g(x)$ for all x. We set $h(x) = \Sigma^{\leq p(|x|)}$. Since $h \in$ NFAMV, we conclude that f belongs to NFAMV $\ominus$ CFLMV. □

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

Proposition 5. $\mathrm{CFLSV_t} \neq \mathrm{CFLSV_t}^{(2)}$. *(Also for* CFLSV *and* CFLMV.*)*

Proof Sketch. Let $\Sigma = \{0,1,\natural\}$ be our alphabet and define $f_{dup\natural}(x) = \{x\natural x\}$ for any input $x \in \{0,1\}^*$ and $f_{dup\natural}(x) = \{\lambda\}$ for any other inputs x. It is not difficult to show that $f_{dup\natural} \in \mathrm{CFLSV_t}^{(2)}$. To show that $f_{dup\natural} \notin \mathrm{CFLSV_t}$, we assume that $f_{dup\natural} \in \mathrm{CFLSV_t}$. Let $DUP_\natural$ be a "marked" version of DUP (duplication), defined as $DUP_\natural = \{x\natural x \mid x \in \{0,1\}^*\}$. It holds that, for every $w \neq \lambda$, $w \in \mathrm{ran}(f_{dup\natural})$ iff there is a string x such that $w \in f_{dup\natural}(x)$ iff $w \in DUP_\natural$. Thus, $DUP_\natural \cup \{\lambda\} = \mathrm{ran}(f_{dup\natural})$. Note that $DUP_\natural \in \mathrm{CFL}$ since $f_{dup\natural} \in \mathrm{CFLSV_t}$. However, it is well-known that $DUP_\natural \notin \mathrm{CFL}$. This leads to a contradiction. Therefore, we conclude that $f_{dup\natural} \notin \mathrm{CFLSV_t}$. □

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

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

Proposition 6. *1.* $\mathrm{REG}//\mathrm{NFASV_t} \nsubseteq \mathrm{CFL}$ *and* $\mathrm{CFL} \nsubseteq \mathrm{REG}//\mathrm{NFAMV}$.
2. $\mathrm{REG}//\mathrm{NFASV_t}$ *is closed under complement but* $\mathrm{REG}//\mathrm{NFAMV}$ *is not.*
3. $\mathrm{CFL} \cap \text{co-CFL} \subsetneq \mathrm{REG}//\mathrm{CFLSV_t}$.

Proof Sketch. (1) Note that the language $DUP_\# = \{x\#x \mid x \in \{0,1\}^*\}$ (duplication) falls into $\mathrm{REG}//\mathrm{NFASV_t}$ by setting $h(x\#y) = y$ and $B = \{[\,{x \atop x}\,] \mid x \in \{0,1\}^*\}$. The key idea is the following claim. (*) A language L is in REG//NFAMV iff L is recognized by a certain *one-way two-head (non-sensing) nfa* (or an nfa(2), in short) with λ-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 nfa(2), it follows that $L_{pal} \notin \mathrm{REG}//\mathrm{NFAMV}$.

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

Bwfore 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 CFL belong to REG. On the contrary, there is a function $f : \{1\}^* \to \{0,1\}^*$ such that f is in CFLMV but not in NFAMV.

4 Oracle Computation and Two Relativizations

Oracle computation is a natural extension of ordinary stand-alone computation by providing external information by way of *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., [14]), let us consider many-one and Turing relativizations of CFL functions. The first notion of *many-one relativization* was discussed for languages in automata theory [8, 21] and we intend to extend it to CFL functions. Given any language A over alphabet Γ, a function $f : \Sigma^* \to \mathcal{P}(\Gamma^*)$ is in CFLMV_m^A (or $\mathrm{CFLMV}_m(A)$) 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, $p \in ACC_M(x)$), 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) $f(x)$ equals the set $\{z_p \mid y_p \in A, p \in ACC_M(x)\}$. Such an M is referred to as an *oracle npda*. Given any language family $\mathcal{C}$, we further set $\mathrm{CFLMV}_m^{\mathcal{C}}$ (or $\mathrm{CFLMV}_m(\mathcal{C})$) to be $\bigcup_{A\in\mathcal{C}} \mathrm{CFLMV}_m^A$. Similarly, we define CFLSV_m^A and $\mathrm{CFLSV}_m^{\mathcal{C}}$ by additionally demanding the size of each output string set is at most 1. Using CFLSV_m^A, a relativized language family CFL_m^A defined in [21] can be expressed as $\mathrm{CFL}_m^A = \{L \mid \eta_L \in \mathrm{CFLSV}_m^A\}$.

Lemma 7. *1.* $\mathrm{CFLMV}_m^{\mathrm{REG}} = \mathrm{CFLMV}$ *and* $\mathrm{CFLSV}_m^{\mathrm{REG}} = \mathrm{CFLSV}$*.*
2. $\mathrm{CFLSV}^{(k+1)} = \mathrm{CFLSV}_m^{\mathrm{CFL}(k)}$*.*

Proposition 8. *1.* $\mathrm{REG}//\mathrm{CFLSV_t} \subseteq \mathrm{CFL}_m^{\mathrm{CFL}(2)} \cap \text{co-}\mathrm{CFL}_m^{\mathrm{CFL}(2)}$*.*
2. $\mathrm{CFL}//\mathrm{CFLSV_t} \subseteq \mathrm{CFL}_m^{\mathrm{CFL}(3)}$*.*

Proof Sketch. We will prove only (2). For convenience, we write $[x,y]^T$ for $[\,{}^x_y\,]$ in this proof. Let $\mathrm{CFL}_{m[1]}^A = \mathrm{CFL}_m^A$ and $\mathrm{CFL}_{m[k+1]}^A = \mathrm{CFL}_m(\mathrm{CFL}_{m[k]}^A)$ for every index $k \geq 1$ [21]. Let $L \in \mathrm{CFL}//\mathrm{CFLSV_t}$ and take an npda M and a function $h \in \mathrm{CFLSV_t}$ 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 $\natural$-extensions of x and y, respectively. An oracle A receives a $\natural$-extension $[\tilde{x},\tilde{y}]^T$ and decides whether M accepts $[x,y]^T$ by removing all $\natural$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 $[x,z]^T$ 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 $\mathrm{CFL}_m^{\mathrm{CFL}}$. Hence, A is in $\mathrm{CFL}_m^B \subseteq \mathrm{CFL}_{m[2]}^{\mathrm{CFL}}$, and thus L is in $\mathrm{CFL}_{m[3]}^{\mathrm{CFL}}$. Since $\mathrm{CFL}_{m[3]}^{\mathrm{CFL}} = \mathrm{CFL}_m^{\mathrm{CFL}(3)}$ [21], the desired conclusion follows. □

The second relativization is *Turing relativization.* A multi-valued partial function f belongs to CFLMV_T^A (or $\mathrm{CFLMV}_T(A)$) if there exists an oracle npda M having *three extra inner 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, say, s written on the query tape is sent to A and, automatically, *the content of the query tape becomes blank* and *the tape head returns to the start 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 $O(n)$ *no matter what oracle is used.*

Obviously, $\mathrm{CFLMV}_T^A = \mathrm{CFLMV}_T^{\overline{A}}$ holds for any oracle A. Define $\mathrm{CFLMV}_T^{\mathcal{C}}$ (or $\mathrm{CFLMV}_T(\mathcal{C})$) to be the union $\bigcup_{A\in\mathcal{C}} \mathrm{CFLMV}_T^A$ for a given language family $\mathcal{C}$.

Analogously to the well-known *NPMV hierarchy*, composed of $\Sigma_k^{\mathrm{p}}\mathrm{MV}$ and $\Pi_k^{\mathrm{p}}\mathrm{MV}$ for $k \in \mathbb{N}^+$ [14], we inductively define $\Sigma_1^{\mathrm{CFL}}\mathrm{MV} = \mathrm{CFLMV}$, $\Pi_k^{\mathrm{CFL}}\mathrm{MV} = \text{co-}\Sigma_k^{\mathrm{CFL}}\mathrm{MV}$, and $\Sigma_{k+1}^{\mathrm{CFL}}\mathrm{MV} = \mathrm{CFLMV}_T(\Pi_k^{\mathrm{CFL}})$ for every index $k \geq 1$. In a similar fashion, we define $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}$ using CFLSV_T^A in place of CFLMV_T^A. The above *CFLMV hierarchy* is useful to scaling the computational complexity of given functions. For example, the function $f(w) = \{x \in \{0,1\}^* \mid \exists u, v\,[w = uxxv]\}$ for every $w \in \{0,1\}^*$ belongs to $\Sigma_2^{\mathrm{CFL}}\mathrm{MV}$. Moreover, it is possible to show that $\mathrm{CFLMV} \cup \text{co-CFLMV} \subseteq \mathrm{CFLMV} \ominus \mathrm{CFLMV} \subseteq \Sigma_4^{\mathrm{CFL}}\mathrm{MV}$.

Proposition 9. *Each function in* $\bigcup_{k\in\mathbb{N}^+} \Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$ *can be computed by an appropriate* $O(n)$ *space-bounded multi-tape deterministic Turing machine.*

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

For $k \geq 3$, it is possible to give the exact characterization of $\mathrm{REG}/\!/\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$. This makes a sharp contrast with Proposition 6(3).

Proposition 10. *For every index* $k \geq 3$, $\Sigma_k^{\mathrm{CFL}} \cap \Pi_k^{\mathrm{CFL}} = \mathrm{REG}/\!/\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$.

Proof Sketch. By extending the proof of Proposition 6(3), it is possible to show that $\Sigma_k^{\mathrm{CFL}} \cap \Pi_k^{\mathrm{CFL}} \subseteq \mathrm{REG}/\!/\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$. Similarly to Proposition 6(2), it holds that $\mathrm{REG}/\!/\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$ is closed under complement. It thus suffices to show that $\mathrm{REG}/\!/\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}} \subseteq \Sigma_k^{\mathrm{CFL}}$. This can be done by a direct simulation. □

Lemma 11. *Let* $e \geq 2$ *and* $k \geq 1$. $\Sigma_k^{\mathrm{CFL}}\mathrm{SV} = \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV} \Rightarrow \Sigma_k^{\mathrm{CFL}} = \Sigma_{k+1}^{\mathrm{CFL}} \Rightarrow \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV} = \Sigma_{k+e}^{\mathrm{CFL}}\mathrm{SV}$.

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

Regarding refinement, from the proof of [8, Theorem 3] follows $\mathrm{NFAMV} \sqsubseteq_{ref} \mathrm{NFASV}$. This result leads to $\mathrm{NFAMV} \circ \mathrm{CFLSV} \sqsubseteq_{ref} \mathrm{CFLSV}$ in [10]. By a direct simulation, nevertheless, it is possible to show that $\Sigma_k^{\mathrm{CFL}}\mathrm{MV} \sqsubseteq_{ref} \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV}$ for every $k \geq 1$. Lemma 11 together with this fact leads to the following consequence.

Proposition 12. *Let* $k \geq 2$. *If* $\Sigma_k^{\mathrm{CFL}} = \Sigma_{k+1}^{\mathrm{CFL}}$, *then* $\Sigma_{k+1}^{\mathrm{CFL}}\mathrm{MV} \sqsubseteq_{ref} \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV}$.

Recently, it was shown in [22] that $\mathrm{CFLMV} \not\sqsubseteq_{ref} \mathrm{CFLSV}$ 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* $f : \Sigma^* \to \Gamma^*$ such that there exists an npda M and an *opt* $\in$ {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* $<$ over Γ^* (e.g., *abbe* $<$ *abc* and *ab* $<$ *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 $f(w) = \max\{PAL_{sub}(w)\}$ for $w \in \{0,1\}^*$, where PAL_{sub} is defined in Section 1, is a member of OptCFL. It holds that $\mathrm{CFLMV_t} \sqsubseteq_{ref} \mathrm{OptCFL}$.

Proposition 13. $\mathrm{CFLSV_t} \subsetneqq \mathrm{OptCFL} \subseteq \Sigma_4^{\mathrm{CFL}}\mathrm{SV_t}$.

The first part of Proposition 13 comes from a fact that the function $f(w) = \max\{g(w)\}$, where $g(w) = \{\lambda\} \cup \{x_i y_i \mid w = x_1\natural x_2\natural x_3\#y_1\natural y_2\natural y_3, x_i = y_i^R, i \in \{1,2,3\}\}$, is in OptCFL but not in CFLSV_t. This latter part is proven by applying the functional pumping lemma.

Note that, in the polynomial-time setting, a much sharper upper-bound of $\mathrm{OptP} \subseteq \Sigma_2^{\mathrm{p}}\mathrm{SV_t}$ 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 $\Sigma_2^{\mathrm{CFL}}\mathrm{SV_t}$.

Proposition 14. $\mathrm{OptNFA}_{EL} \subseteq \Sigma_2^{\mathrm{CFL}}\mathrm{SV_t}$.

Proof Sketch. Let $f \in \mathrm{OptNFA}_{EL}$ 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 oracle answer is 1, enter an accepting state; otherwise, reject. Make another npda M_2 receive $y\#x^R$, 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. $\square$

Proposition 15. *1.* $\mathrm{CFL} \cup \text{co-CFL} \subseteq \mathrm{REG}//\mathrm{OptCFL} \subseteq \Sigma_4^{\mathrm{CFL}} \cap \Pi_4^{\mathrm{CFL}}$.
2. $\mathrm{CFL}//\mathrm{OptCFL} \subseteq \Sigma_5^{\mathrm{CFL}}$.

Proof Sketch. We will show only (1). Note that REG//OptCFL is closed under complementation. Let $L \in \mathrm{CFL}$ 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 $h(x)$ be $\max\{N_1(x)\}$ for all x's. It follows that $L = \{[\,{}^{x}_{h(x)}\,] \mid h(x) \neq \emptyset\}$. This proves that $L \in \mathrm{REG}//\mathrm{OptCFL}$. Next, let $L \in \mathrm{REG}//\mathrm{OptCFL}$. Since $\mathrm{OptCFL} \subseteq \Sigma_4^{\mathrm{CFL}}\mathrm{SV_t}$, we obtain $L \in \mathrm{REG}//\Sigma_4^{\mathrm{CFL}}\mathrm{SV_t}$. By Proposition 10, this implies that L belongs to $\Sigma_4^{\mathrm{CFL}} \cap \Pi_4^{\mathrm{CFL}}$. $\square$

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 langauges by functional binary transductions. Inform. Control, 15, 95–109 (1969)
9. Köbler, 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)