Nondeterministically Selecting Positive Instances of Context-Free Languages*

Tomoyuki Yamakami

Faculty of Engineering, University of Fukui, 3-9-1 Bunkyo, Fukui 910-8507, Japan

Abstract. The notion of effectively selective languages was introduced into computational complexity theory in late 1970s and it has been discussed extensively in connection to the P=?NP question. Its scope has been extended to finite automata and formal languages in the past literature. By further extending the scope, this paper discusses the computational complexity of selective languages whose underlying selectors are computed in various ways by nondeterministic one-way pushdown automata equipped with write-once output tapes. We explore basic properties of those languages and demonstrate their relationships to the existing language families. We also seek a generalization of such selective languages by taking selectors from a higher functional hierarchy over multi-valued CFL functions.

Keywords: pushdown automaton, selective set, advice, multi-valued function, CFL hierarchy, CFLMV hierarchy

1 Complexity of Selecting Positive Instances

One-way pushdown automata are generally considered as simple-structured machines with limited access to (pushdown) memory devices and one may think that various questions on their "structural" properties could be easily answered. In reality, on the contrary, there still remain numerous unsolved questions concerning the family CFL of context-free languages recognized by *one-way nondeterministic pushdown automata* (or 1npda's, for short) as well as its deterministic variant DCFL. As a few quick examples, it is yet unknown whether every language in co-CFL is "dissectible" by appropriately-chosen regular languages [23] and whether there is a CFL/n-"pseudorandom" language in CFL(2) [21], where CFL(k) consists of intersections of k context-free languages and CFL/n is a Karp-Lipton advice variant of CFL [16]. Viewing DCFL and CFL as reasonable analogues of P and NP, many complexity-theoretic concepts have been adapted into automata theory. Along this line of research, we intend to study the notion of "selectivity" of Selman [9] strictly within the framework of formal languages and automata theory.

Given two arbitrary input instances to a pre-determined language, a *selector* chooses at least one of the two instances so that the chosen instance is most

likely to be a "positive" instance of the language (namely, to belong to the language). We are particularly concerned with the efficiency of such a sector. The computational complexity of languages whose selectors are efficiently computable has been discussed in connection to the P =?NP question. The importance of selectivity also comes from the fact that each selector can induce a certain type of partial pre-order on equivalence classes over strings (see [7, 9]).

The notion of P-*selective languages*, which was first introduced into computational complexity theory in 1979 by Selman [9] as a polynomial-time analogue of Jockusch's *semi-recursive sets* in recursion theory. The selective languages have been since then used as an important tool in studying various structural properties, such as completeness, self-reducibility, search-to-decision reductions, refinements, and promise problems (see [2] for references therein). Later, this notion was extended to NPSV- *and* NPMV-*selective languages* [2] and beyond to Σ_k^{P}SV- *and* Σ_k^{P}MV-*selective languages* [6], where two suffixes "SV" and "MV" indicate "single-valued function" and "multi-valued function", and Σ_k^{P} is the kth level of the polynomial-time hierarchy [8]. More generally, given any function class $\mathcal{F}$, $\mathcal{F}$-selective languages are defined simply by the choice of appropriate selectors from $\mathcal{F}$. Since the selectivity notion is heavily linked to languages and multi-valued partial functions, it is ideal to study the computational complexity of languages and functions together.

Taking the opposite research direction, we look into more restrictive computational models in order to witness how dramatically the complexity of selecting positive instances changes. A precursor to our investigation is the study of "DFA-selectivity" (which was originally called fa-selectivity) of Tantau [13]. Our main interest, on the contrary, lies on the behaviors of selectors when they are computed on 1npda's equipped with write-once* output tapes. Those machines are quite different in nature from powerful Turing machines and their behaviors are quite restrictive because of the limited usage of their memory device. Multi-valued partial functions computed by such machines are generally called CFL-*functions* and they have been recently discussed in a series of works [17, 20, 18, 21]. More specifically, we intend to study selective languages whose selectors are particularly chosen from function classes, such as CFLSV and CFLMV [20] (see Section 2.2 for their definitions), which are natural analogues of the aforementioned function classes NPSV and NPMV, respectively. In a later section, we then turn our attention to higher complexity classes Σ_k^{CFL}SV and Σ_k^{CFL}MV, which constitute a functional version of the CFL *hierarchy* [19] defined in parallel to the polynomial(-time) hierarchy.

2 Basic Notions and Notations

We will briefly explain existing notions and notations necessary to read through the subsequent sections.

* A tape is *write-once* if its tape head never moves to the left and, whenever it writes a nonempty symbol, it must move to the next blank cell.

2.1 Alphabets, Strings, and Languages

Given a finite set A, the notation $\|A\|$ denotes the number of elements in A. Let $\mathbb{N}$ indicate the set of all *natural numbers* (i.e., nonnegative integers) and set $\mathbb{N}^+ = \mathbb{N} - \{0\}$. For two integers m and n with $m \leq n$, the notation $[m,n]_{\mathbb{Z}}$ denotes the *integer interval* $\{m, m+1, m+2, \ldots, n\}$. In particular, we abbreviate $[1,n]_{\mathbb{Z}}$ as $[n]$ for any $n \in \mathbb{N}^+$. We use the term "polynomials" to mean polynomials with nonnegative integer coefficients. 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 Σ of "symbols" or "letters". A *string* x over Σ is a finite series of symbols chosen from Σ and its *length*, denoted by $|x|$, is the total occurrences of symbols in x. The *empty string* λ is a unique string of length 0. To express a pair of strings simultaneously, we use the *track notation* $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ and follow its convention, as discussed in [12]. For two symbols σ and τ, $\left[\begin{smallmatrix} \sigma \\ \tau \end{smallmatrix}\right]$ expresses a new symbol. For two strings $x = x_1 x_2 \cdots x_n$ and $y = y_1 y_2 \cdots y_n$ of length n, $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ denotes the concatenated string $\left[\begin{smallmatrix} x_1 \\ y_1 \end{smallmatrix}\right]\left[\begin{smallmatrix} x_2 \\ y_2 \end{smallmatrix}\right] \cdots \left[\begin{smallmatrix} x_n \\ y_n \end{smallmatrix}\right]$. We further expand this notation as follows. In the case of $|x| < |y|$, $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ expresses $\left[\begin{smallmatrix} x\#^m \\ y \end{smallmatrix}\right]$, where $m = |y| - |x|$ and $\#$ is a designated new symbol. Similarly, when $|x| > |y|$, the same succinct notation $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ expresses $\left[\begin{smallmatrix} x \\ y\#^m \end{smallmatrix}\right]$ with $m = |x| - |y|$.

A *language* over alphabet Σ is a collection of strings over Σ. Given an index $k \in \mathbb{N}$, Σ^k is composed of all strings of length k; in particular, Σ^0 equals $\{\lambda\}$. Let $\Sigma^* = \bigcup_{k \in \mathbb{N}} \Sigma^k$. For 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.

A function $h : \mathbb{N} \to \Sigma^*$ is called *length preserving* if $|h(n)| = n$ holds for all $n \in \mathbb{N}$. Recall from [19, 20] the notion of $\natural$-extension. For any string $x \in \Sigma^*$, a $\natural$-*extension* of x is a string $\tilde{x}$ over $\Sigma \cup \{\natural\}$ such that x is obtained from $\tilde{x}$ by deleting all occurrences of $\natural$. Given a language A, the *characteristic function* χ_A for A is a unique function defined as $\chi_A(x) = 1$ if $x \in A$, and $\chi_A(x) = 0$ otherwise.

Our basic computation models are *1-way (one-tape, one-head) nondeterministic finite automata* (or 1nfa's, for short) with λ-moves and *1-way (one-tape, one-head) nondeterministic pushdown automata* (or 1npda's) with λ-moves, Where a λ-*move* is a step at which an input-tape head stays still reading nothing. We also use their deterministic variants: 1dfa's and 1dpda's. Formally, a 1npda N with a write-once output tape computing $f : \Sigma^* \to \mathcal{P}(\Theta^*)$ has the following form: $N = (Q, \Sigma, \{\mathfrak{c}, \$\}, \Gamma, \Theta, \delta, q_0, \bot, Q_{acc}, Q_{rej})$, where Q is a finite set of inner states, Γ is a stack alphabet, Θ is an output alphabet, q_0 $(\in Q)$ is the initial state, $\bot$ $(\in \Gamma)$ is the bottom marker, $\delta : (Q - Q_{halt}) \times \check{\Sigma}_\lambda \times \Gamma \to \mathcal{P}(Q \times \Gamma^* \times \Theta_\lambda)$, where $Q_{halt} = Q_{acc} \cup Q_{rej}$, $\check{\Sigma}_\lambda = \Sigma \cup \{\lambda, \mathfrak{c}, \$\}$, $\mathfrak{c}$ and $\$$ are respectively the left and the right endmarkers, and $\Theta_\lambda = \Theta \cup \{\lambda\}$. Following [17, 19, 20, 18, 21], our 1npda's are assumed to reach halting states along all computation paths in $O(n)$ steps, where n refers to input size (see [19, 20] for reasoning). When M is in inner state q, scanning σ on the input tape, and a in the topmost stack cell, a transition $(p, z, \tau) \in \delta(q, \sigma, a)$ changes q to p, replaces a by z, and writes τ on the output tape. When $\sigma \neq \lambda$, the input-tape head must move to the

right. In contrast, when M makes a λ-move, its input-tape head reads λ and does not move. As for a 1nfa N, we use another transition function of the form $\delta : (Q - Q_{halt}) \times \check{\Sigma}_\lambda \to \mathcal{P}(Q \times \Theta_\lambda)$. It is important to remark that N is allowed to make λ-moves like 1npda's.

Along each computation path, a machine may produce a string on the output tape. We call such a string a *valid output* (or a *legitimate output*) if it is produced along an appropriate accepting computation path. Only valid outputs are considered as the machine's true "outputs" and we automatically disregard any string produced along any rejecting computation path. This convention is particularly important for one-way machines because this makes it possible for such a one-way machine to start producing all possible strings nondeterministically while reading an input string and then to *invalidate* any unwanted strings by simply entering rejecting states in the time of halting.

We say that a machine with two heads on input and output tapes is *synchronous* if, whenever the input-tape head moves to the right, the output-tape head also moves to the right, and vice visa.

To treat input instances of a given "2-ary" partial function f on a model of "1-way" machine M, we use the aforementioned track notation. For technical reason, unlike the case of Turing machines, we do not use two-tape 1nfa's and 1npda's to access two independent input strings x and y; instead, we split a single input tape into two tracks and write x and y on these tracks separately. More precisely, when a pair (x, y) of strings is given as an input instance to the machine M, we place x in the upper track and y in the lower track as $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ so that the tape head can read x and y simultaneously from left to right. More formally, if we wish to compute f on input (x, y), we start its underlying machine M whose input tape contains the string of the form $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$. From this formalism, we write $f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right])$, instead of $f(x, y)$, to describe the function f that takes an input of the form $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$.

The notations REG and CFL stand for the family of all regular languages and that of all context-free languages, respectively. For each number $k \in \mathbb{N}^+$, the *k-conjunctive closure* of CFL, denoted $\mathrm{CFL}(k)$, is defined recursively as $\mathrm{CFL}(1) = \mathrm{CFL}$ and $\mathrm{CFL}(k+1) = \{A \cap B \mid A \in \mathrm{CFL}(k), B \in \mathrm{CFL}\}$ (cf. [22]). The advised language family REG/n consists of all languages L for which there exist an advice alphabet Γ, a length-preserving total function (called an *advice function*) $h : \mathbb{N} \to \Gamma^*$, and a language $A \in \mathrm{REG}$ satisfying $L = \{x \mid \left[\begin{smallmatrix} x \\ h(|x|) \end{smallmatrix}\right] \in A\}$ [12]. Similarly, by replacing REG in REG/n with CFL, we obtain CFL/n [15, 16]. Moreover, P (resp., NP) is composed of all languages recognized by one-tape deterministic Turing machines or DTMs (resp., one-tape nondeterministic Turing machines or NTMs) in polynomial time.

2.2 Multi-Valued Partial Functions and Refinement

Throughout this paper, similarly to [20], 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[⋆⋆] Σ^* to Γ^* for certain alphabets Σ and Γ, not necessarily limited to $\{0,1\}$. Whenever f is single-valued, we often write $f(x) = y$ instead of $y \in f(x)$. The *domain* $\mathrm{dom}(f)$ of f is defined to be the set $\{x \in \Sigma^* \mid f(x) \neq \varnothing\}$. When $x \notin \mathrm{dom}(x)$, $f(x)$ is said to be *undefined*.

A multi-valued partial function $f : \Sigma^* \to \mathcal{P}(\Gamma^*)$ is called *polynomially bounded* if there exists a polynomial p such that, for any two strings $x \in \Sigma^*$ and $y \in \Gamma^*$, if $y \in f(x)$ then $|y| \leq p(|x|)$ holds. We understand that all function classes dealt with in this paper are assumed to be polynomially bounded. Given two alphabets Σ and Γ, 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 $|x| = |y|$. Similarly, f is *length decreasing* if $y \in f(x)$ implies $|y| < |x|$ for any $x, y \in \Sigma^*$.

Given two multi-valued functions f and g, we say that g is a *refinement* of f, denoted by $f \sqsubseteq_{ref} g$, if (1) $\mathrm{dom}(g) = \mathrm{dom}(f)$ and (2) for every input $x \in \mathrm{dom}(f)$, $g(x) \subseteq f(x)$ (as a set inclusion) holds (see [11]). When Condition (1) is replaced by (1′) $\mathrm{dom}(f) \subseteq \mathrm{dom}(g)$, we say that g is a *pseudo refinement* of f, denoted by $f \sqsubseteq^{(+)}_{ref} g$. For two sets $\mathcal{F}$ and $\mathcal{G}$ of functions, $\mathcal{F} \sqsubseteq_{ref} \mathcal{G}$ (resp., $\mathcal{F} \sqsubseteq^{(+)}_{ref} \mathcal{G}$) if, for every function $f \in \mathcal{F}$, there exists a function $g \in \mathcal{G}$ for which $f \sqsubseteq_{ref} g$ (resp., $f \sqsubseteq^{(+)}_{ref} g$). Clearly, $\mathcal{F} \sqsubseteq_{ref} \mathcal{G}$ implies $\mathcal{F} \sqsubseteq^{(+)}_{ref} \mathcal{G}$. When g is further single-valued, we often call f a *single-valued (pseudo) refinement* of f.

The following four basic function classes were introduced in [17]. The function class CFLMV is composed of all multi-valued functions f, each of which maps Σ^* to $\mathcal{P}(\Sigma^*)$ for a certain alphabet Σ and there exists a 1npda N with a 1-way read-once input tape together with a write-once 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. $\mathrm{CFLMV_t}$ is a natural restriction of CFLMV onto *total functions*. Two classes CFLSV and $\mathrm{CFLSV_t}$ are subclasses of CFLMV and $\mathrm{CFLMV_t}$ consisting only of single-valued functions. Another function class CFLMV(2) contains all multi-valued functions f such that there are two functions g_1 and g_2 in CFLMV satisfying $f(x) = g_1(x) \cap g_2(x)$ (set intersection) for any string x [21]. In a similar way, CFL2V(2) and $\mathrm{CFL2V_t}(2)$ are defined.

3 Efficiently Selective Languages and Their Basic Properties

In the polynomial-time setting, multi-valued functions have been used to define the notions of NPMV- and Σ^{P}_kMV-selectivity. Here, we introduce CFLMV-selectivity and its variants.

[⋆⋆] To describe a multi-valued function f, the expression "$f : \Sigma^* \to \Gamma^*$" is customarily used in the literature. This is equivalent to saying that f is a total mapping from Σ^* to $\mathcal{P}(\Gamma^*)$.

3.1 Definition of Selectors and Selective Languages

We begin with a general notion of *selectors*.

Definition 1. *Given a multi-valued function* $f : \Sigma^* \times \Sigma^* \to \mathcal{P}(\Sigma^*)$ *(called a* selector*), a language* A *is* f-selective *if, for any pair* $x, y \in \Sigma^*$*, (a)* $f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) \subseteq \{x, y\}$*, and (b)* $\{x, y\} \cap A \neq \varnothing$ *implies both* $f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) \neq \varnothing$ *and* $f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) \subseteq A$*.*

As an essential difference from two-way Turing machines, when a 1npda on input $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ produces a string, say, s on its output tape along each computation path, the 1npda cannot scan the passed tape cells again and thus there seems no general way of knowing that the produced string s is x or y unless the 1npda is synchronous. To circumvent the lack of such a post-production checkup of the output strings, we require the following extra technical setup. When a selector handles a string of the form $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$, we want to provide an underlying machine with $\left[\begin{smallmatrix} x\$_1 \\ y\$_2 \end{smallmatrix}\right]$, where $\$_1$ and $\$_2$ are designated endmarkers used for the first and the second tracks. To do so, we introduce the notion of "endmarked extensions." Given a selector f, another function g is called the *endmarked extension* of f if, for any distinct $x, y \in \Sigma^*$, (1) $f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) = \varnothing$ iff $g(\left[\begin{smallmatrix} x\$_1 \\ y\$_2 \end{smallmatrix}\right]) = \varnothing$, (2) $x \in f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right])$ iff $x\$_1 \in g(\left[\begin{smallmatrix} x\$_1 \\ y\$_2 \end{smallmatrix}\right])$, (3) $y \in f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right])$ iff $y\$_2 \in g(\left[\begin{smallmatrix} x\$_1 \\ y\$_2 \end{smallmatrix}\right])$, and (4) $x \in f(\left[\begin{smallmatrix} x \\ x \end{smallmatrix}\right])$ iff $g(\left[\begin{smallmatrix} x\$_1 \\ x\$_2 \end{smallmatrix}\right]) \in \{x\$_1, x\$_2\}$. For brevity, we call $x\$_1$ and $y\$_2$ *endmarked strings*.

Definition 2. *Given a class* $\mathcal{F}$ *of multi-valued partial functions, a language* A *is* $\mathcal{F}$-selective *if there exists a multi-valued function* f *and its endmarked extension* $\tilde{f}$ *such that (i)* A *is* f*-selective and (ii) both* f *and* $\tilde{f}$ *belong to* $\mathcal{F}$*. We write* $\mathcal{F}$-SEL *for the collection of all* $\mathcal{F}$*-selective languages.*

We stress that Definition 2 does not change the well-known selective language families, such as P-SEL and NPSV-SEL. For a further discussion on our introduction of endmarked extension, see Section 6.

Unlike P-SEL, however, we do not intend to discuss DFA-SEL and DCFL-SEL throughout this paper because they are too restrictive in practice to contain even their associated underlying language families REG and DCFL, respectively.

Let us make simple observations. Obviously, for any two function classes $\mathcal{F}$ and $\mathcal{G}$, if $\mathcal{F} \subseteq \mathcal{G}$, then $\mathcal{F}$-SEL $\subseteq \mathcal{G}$-SEL. Next, we remark that a 1nfa can decide which of two given strings x and y in the form of $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ is lexicographically smaller than the other. This fact will be used in later arguments.

Let $\leq$ denote the *lexicographic order* (i.e., sort firstly by length and sort secondly by a fixed alphabetical order).

Lemma 3. *The function* f *defined as* $f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) = \min\{x, y\}$*, where* min *is taken according to the lexicographic order, can be computed by a certain 1nfa.*

Proof. Given the function f, let us consider the following 1nfa M. On input $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$, first "guess" either x or y (i.e., nondeterministically choose either x or y), write down the guessed string, say, s. Assuming $|x| = |y|$, determine whether or not $s = \min\{x, y\}$ and then check whether $|x| = |y|$ by reading the entire input.

In the case of $|x| \neq |y|$, we immediately *invalidate* this computation path by entering a rejecting state. More precisely, we compare x and y symbol by symbol from left to right to check whether x precedes y or y precedes x according to the lexicographic order $\leq$. If we discover that x precedes y, then we check whether $s = x$ and $|x| = |y|$. If so, we accept; otherwise, we reject. When y precedes x, we do the same by exchanging x and y. □

A quick application of the above lemma is the existence of special selectors. We say that a selector f is *symmetric* if $f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) = f(\left[\begin{smallmatrix} y \\ x \end{smallmatrix}\right])$ for any pair $x, y \in \Sigma^*$. It is possible to make certain selectors symmetric.

Lemma 4. *For every language A in* $\mathrm{CFLSV_t}$-SEL*, there always exists a symmetric* $\mathrm{CFLSV_t}$*-selector f for A. This statement also holds for* CFLSV-SEL*,* CFL2V-SEL*, and* $\mathrm{CFL2V_t}$-SEL*.*

Lemma 5. $\mathrm{CFLSV_t}$-SEL $=$ co-($\mathrm{CFLSV_t}$-SEL) *and* co-(CFLSV-SEL) $\subseteq$ CFL2V-SEL*.*

The $\mathcal{F}$-selectivity is associated with a certain partial pre-ordering, which satisfies reflexivity and transitivity. We give a simple example of $\mathrm{CFLSV_t}$-selective language using the standard lexicographic ordering. Given a machine M and an input x, $M(x) = 1$ (resp., $M(x) = 0$) means that M accepts (resp., rejects) x.

Example 6 (1) All regular languages are $\mathrm{NFASV_t}$-selective; namely, REG $\subseteq$ $\mathrm{NFASV_t}$-SEL. To see this fact, for each regular language A, we take an arbitrary 1dfa M and construct its associated function f as follows. Define $f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) = x$ if (i) $M(x) = M(y) = 1$ and $x \leq y$, (ii) $M(x) = 1 \neq M(y)$, or (iii) $M(x) = M(y) = 0$. It is not difficult to see that f is indeed a selector and in $\mathrm{NFASV_t}$-SEL.

(2) Given a "single-valued" symmetric selector f for a language A over Σ, we write $x \preceq_f y$ if $f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) = x$. This binary relation $\preceq_f$ becomes a partial pre-order on Σ^* and it satisfies that, for every $y \in \Sigma^*$, $y \in A$ implies $\{x \mid x \preceq_f y\} \subseteq A$, and $y \notin A$ implies $A \subseteq \{x \mid x \preceq_f y\}$. This property will be used later.

3.2 Closure Properties

It is known that CFL is closed under union and inverse homomorphism but not intersection. Most $\mathcal{F}$-SEL's are naturally closed under union. By sharp contrast, intersections of even $\mathrm{NFASV_t}$-selective languages possess arbitrarily high complexities.

Lemma 7. *Let us consider an arbitrary language A over the binary alphabet $\Sigma = \{0, 1\}$ and assume that $\|A \cap \Sigma^n\| \leq 1$ holds for all $n \in \mathbb{N}^+$. There exist two languages $L_1, L_2 \in$* $\mathrm{NFASV_t}$-SEL *satisfying $A = L_1 \cap L_2$.*

Proof. Using the standard lexicographic ordering $\leq$ on Σ^*, we define $L_1 = \{x \in \Sigma^* \mid \exists y \in A \cap \Sigma^{|x|}[x \leq y]\}$. By Lemma 3, the function $f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) = \min\{x, y\}$, where "min" is according to $\leq$, belongs to $\mathrm{NFASV_t}$. Thus, f is an $\mathrm{NFASV_t}$-selector for

L_1. In a similar way, we define $L_2 = \{x \in \Sigma^* \mid \exists y \in A \cap \Sigma^{|x|}[y \leq x]\}$. It then follows that $A = L_1 \cap L_2$. □

Since we can freely choose A in Lemma 7, we instantly obtain the following consequence. This is similar to the non-closure property of the class P-SEL of all P-selective sets under intersection [5]. Let TALLY be the set of all *tally languages* over arbitrary alphabets Σ (i.e., languages consisting only of strings of the form a^i for a certain fixed symbol $a \in \Sigma$).

Proposition 8. NFASV$_{\mathrm{t}}$-SEL *is not closed under intersection. The same holds for* CFLSV$_{\mathrm{t}}$-SEL *and* CFLSV-SEL.

Proof. Take a tally language A not in NFASV$_{\mathrm{t}}$-SEL. By Lemma 7, there are two languages $L_1, L_2 \in$ NFASV$_{\mathrm{t}}$-SEL for which $A = L_1 \cap L_2$. If NFASV$_{\mathrm{t}}$-SEL is closed under intersection, then A must be in NFASV$_{\mathrm{t}}$-SEL, a contradiction. □

Note that CFL $\cap$ TALLY $\subseteq$ NFASV$_{\mathrm{t}}$-SEL because all "unary" context-free languages are regular. However, it is not known whether or not CFL $\subseteq$ CFLSV$_{\mathrm{t}}$-SEL.

Next, we argue various closure properties of CFLSV$_{\mathrm{t}}$-SEL. The first property is under a certain weak form of many-one reduction. Given two languages A and B, we say that A is NFASV$_{\mathrm{t}}$-*translatable*, denoted by $A \leq_{trans}^{\mathrm{NFASV_t}} B$, if there exists a synchronous 1nfa N (called a translator) such that, on input x, N produces a single *valid* string, say, w on its output tape such that $x \in A$ iff $w \in B$. If we further require N to satisfy the length decreasing property (i.e., the length of any string produced on the output tape must be strictly shorter than the input length), then we instead use the term of *length-decreasing* NFASV$_{\mathrm{t}}$-*translatability*. We claim the following lemma. In comparison, P-SEL is known to be closed downward under polynomial-time truth-table reductions [10].

Lemma 9. CFLSV$_{\mathrm{t}}$-SEL *is closed downward under* NFASV$_{\mathrm{t}}$-*translations; that is, for any two languages A and B, if $A \leq_{trans}^{\mathrm{NFASV_t}} B$ and $B \in$* CFLSV$_{\mathrm{t}}$-SEL*, then $A \in$* CFLSV$_{\mathrm{t}}$-SEL.

Proof. For any two languages A and B, assume that $A \leq_{trans}^{\mathrm{NFASV_t}} B$ via a translator N with synchronized tape heads and that $B \in$ CFLSV$_{\mathrm{t}}$-SEL via a selector f. Let N_f denote a 1npda computing f. For simplicity, we assume that A and B are languages over the same alphabet, say, Σ. We want to show that $A \in$ CFLSV$_{\mathrm{t}}$-SEL. Given two strings $x, y \in \Sigma^*$, assume that $N(x)$ produces a single valid string, say, z and similarly $N(y)$ produces w. Next, we define $g(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) = x$ if $f(\left[\begin{smallmatrix} z \\ w \end{smallmatrix}\right]) = z$, and $g(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) = y$ if $f(\left[\begin{smallmatrix} z \\ w \end{smallmatrix}\right]) = w$. We claim that g is a selector for A. Let us consider the case where $\{x, y\} \cap A \neq \varnothing$. If $x \in A$ and $y \notin A$, then $z \in A$ and $w \notin A$; thus, we obtain $g(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) = x \in A$. Assume that $x, y \in A$. Since $z, w \in A$ by the definition of N, we conclude that $\{x, y\} \cap g(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) \neq \varnothing$.

Next, we show that g is in CFLSV$_{\mathrm{t}}$ by constructing an appropriate 1npda M computing g. On input $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$, M simulates both $N(x)$ and $N(y)$ simultaneously by reading $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ symbol by symbol since N's tape heads are synchronized, and

it produces $[\begin{smallmatrix} z \\ w \end{smallmatrix}]$ on an "imaginary" output tape. During this process, as each symbol σ of the string $[\begin{smallmatrix} z \\ w \end{smallmatrix}]$ is produced on the imaginary tape, M simulates one step (together with a series of λ-moves) of N_f on σ. It is not difficult to see that M computes g correctly. □

As for P-selectivity, Buhrman and Torenvliet [1] demonstrated that a language A is in P iff A is polynomial-time self-reducible and P-selective. In a similar spirit, we show the following theorem concerning $\mathrm{NFASV_t}$-translatability.

Theorem 10. *Given any language A, if A is $\mathrm{CFLSV_t}$-selective and A is length-decreasing $\mathrm{NFASV_t}$-translatable to $\overline{A}$, then A belongs to $\mathrm{CFL} \cap \text{co-CFL}$.*

Proof. Given any language A, let M denote a 1nfa that translates A to $\overline{A}$. Let f be a $\mathrm{CFLSV_t}$-selector for A and take an appropriate 1npda M_f computing f. To show that $A \in \mathrm{CFL}$, we define N as follows. On input x, N runs M on x. While M produces z along a certain computation path, N produces $[\begin{smallmatrix} x \\ z \end{smallmatrix}]$ on an imaginary tape of N. Note that $x \neq z$ since $x \in A$ iff $z \in \overline{A}$. Whenever a single symbol σ is produced on the imaginary tape, N simulates one step (together with a series of λ-moves, if necessary) of M_f on σ using a stack and the imaginary tape. Since $|z| < |x|$, by simulating M_f, N easily notices which of x and z is a valid outcome of M_f. If $f([\begin{smallmatrix} x \\ z \end{smallmatrix}]) = x$, then N accepts; otherwise, N rejects.

Next, we show that N correctly recognizes A. Assume that $x \in A$. If $f([\begin{smallmatrix} x \\ z \end{smallmatrix}]) = z$, then we conclude that $z \in A$ because $\{x, z\} \cap A \neq \varnothing$. However, this implies $x \notin A$, a contradiction. Thus, we obtain $f([\begin{smallmatrix} x \\ z \end{smallmatrix}]) = x$; in other words, N accepts. Next, assume that $x \notin A$. If $f([\begin{smallmatrix} x \\ z \end{smallmatrix}]) = x$, then $z \notin A$ holds because, otherwise, $f([\begin{smallmatrix} x \\ z \end{smallmatrix}]) = z$ must hold. Since $z \notin A$, we obtain $x \in A$, a contradiction. This implies $f([\begin{smallmatrix} x \\ z \end{smallmatrix}]) = z$, and thus N rejects. Therefore, N correctly recognizes A.

By Lemma 5, $\overline{A}$ is also $\mathrm{CFLSV_t}$-SEL. Moreover, $\overline{A}$ is also length-preserving $\mathrm{NFASV_t}$-translatable to A. Since the above argument for A also works for $\overline{A}$, we thus conclude that $\overline{A}$ is in CFL. Therefore, A belongs to $\mathrm{CFL} \cap \text{co-CFL}$. □

Lemma 11. $\mathrm{CFLSV_t}$-SEL *is closed under inverse homomorphism.*

Proof. Let A denote any language over alphabet Σ in $\mathrm{CFLSV_t}$-SEL and let h be any homomorphism from another alphabet Γ to Σ. Our goal is to show that the language $A_{h^{-1}} = \{x \in \Gamma^* \mid h(x) \in A\}$ belongs to $\mathrm{CFLSV_t}$-SEL. Take a selector f in $\mathrm{CFLSV_t}$ for A and consider a 1npda M computing its endmarked extension $\tilde{f}$. Consider the following algorithm. On input $[\begin{smallmatrix} x \\ y \end{smallmatrix}]$, we guess $z \in \{x, y\}$ and produce it on an output tape. At the same time, we apply h to each symbol of $[\begin{smallmatrix} x \\ y \end{smallmatrix}]$. By adjusting the size of obtained strings by h symbol by symbol, we run M using an "imaginary" output tape. When M ends its computation, M produces either $\$_1$ or $\$_2$, which indicates which of x and y is actually produced. We accept the input if the string produced by M matches the guessed string; otherwise, we reject the input.

4 Power and Limitation of Selective Languages

We will continue exploring basic properties of $\mathcal{F}$-selective languages. In Example 6(1), we have already shown that REG $\subseteq$ NFASV$_{\mathrm{t}}$-SEL. Hereafter, we intend to establish a further connection between CFL variants and their selectivity.

Proposition 12. *(1)* CFL $\subseteq$ CFL2V-SEL. *(2)* CFL $\cap$ co-CFL $\subseteq$ CFLSV$_{\mathrm{t}}$-SEL. *(3)* CFL(2) $\subseteq$ CFL2V(2)-SEL.

Proof. (1) For each language $A \in$ CFL, take a 1npda M that recognizes A. To show that $A \in$ CFL2V-SEL, it suffices to construct an appropriate selector for A. In what follows, we will describe a 1npda N equipped with a write-once output tape computing this selector. Let (x, y) be any input pair. Assume that $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ is given to an input tape of N. On this input $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$, N guesses (i.e., nondeterministically chooses) either x or y and runs M on this guessed string. Assume that N has guessed a string $s \in \{x, y\}$. While reading s, N copies it onto the output tape symbol by symbol. Along each computation path, when M halts in an accepting state or a rejecting state, N enters the same inner state. Remember that, if N enters a rejecting state along a computation path, the string s produced on the output tape is automatically *invalidated*. Let $f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right])$ be the valid outcome of N on the input $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$. It is obvious that f is a two-valued partial function. Since $f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) \subseteq \{x, y\}$ for any pair (x, y), f belongs to CFL2V. It is easy to show that $f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) \subseteq A$ when $\{x, y\} \cap A \neq \varnothing$. Thus, A is in CFL2V-SEL.

(2) Assume that $A \in$ CFL $\cap$ co-CFL. Recall from [21, Lemma 2.1] that $A \in$ CFL $\cap$ co-CFL iff $\chi_A \in$ CFLSV$_{\mathrm{t}}$. By this equivalence, we obtain $\chi_A \in$ CFLSV$_{\mathrm{t}}$. Take a 1npda M computing χ_A. We then define another 1npda N as follows. On input $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$, guess two strings $s, t \in \{x, y\}$ (nondeterministically). In the case of $s = x$, we run M on s, and simultaneously writes t on the output tape. Along a computation path, assuming $t = x$, if M outputs 1, then N rejects; otherwise, N accepts. In contrast, assuming $t = y$, if M outputs 1, then N rejects; otherwise, N accepts. In the case of $s = y$, N runs M on s and writes y. If M outputs 1, then N accepts; otherwise, N rejects. Let g be a multi-valued partial function computed by N. It is easy to show that f is total and single-valued. Moreover, it is not difficult to show that g is a selector for A and in CFLSV$_{\mathrm{t}}$.

(3) Let L be any language in CFL(2) and take two languages $A_1, A_2 \in$ CFL satisfying $L = A_1 \cap A_2$. By (1), there are CFL2V-selectors f_1 and f_2 for A_1 and A_2, respectively. We define $g(z) = f_1(z) \cap f_2(z)$ for any z. Clearly, g belongs to CFL2V(2). Since g is a selector for L, L belongs to CFL2V(2)-SEL. □

The family NFASV$_{\mathrm{t}}$-SEL contains REG by Example 6(1). However, it is not powerful enough to include DCFL.

Proposition 13. DCFL $\not\subseteq$ NFASV$_{\mathrm{t}}$-SEL.

Proof. Consider the non-regular language $L_{ab} = \{a^n b^n \mid n \in \mathbb{N}\}$ over the binary alphabet $\Sigma = \{a, b\}$. Assume that $L_{ab} \in$ NFASV$_{\mathrm{t}}$-SEL and take a selector f for L_{ab} in NFASV$_{\mathrm{t}}$. This f induces a partial pre-order $\preceq_f$ on Σ^* as in Example

6(2). Note that $\{x \in \Sigma^n \mid x \preceq_f a^n b^n\} \subseteq L_{ab}$ and $L_{ab} \subseteq \{x \in \Sigma^n \mid i, j \in \mathbb{N}, x \preceq_f a^i b^j, i + j = n, i \neq j\}$. Thus, L_{ab} equals $\{x \in \Sigma^* \mid n \in \mathbb{N}, x \preceq_f a^n b^n\}$. It then follows that $a^n b^n \leq_f a^i b^j$ for any i, j with $i + j = n$. We define co-1nfa N as follows. On input $a^i b^j$ (with $i + j = n$), guess $a^k b^l$ (with $k + l = n$), run N on $\left[\begin{smallmatrix} a^i b^j \\ a^k b^l \end{smallmatrix}\right]$ to see $f(\left[\begin{smallmatrix} a^i b^j \\ a^k b^l \end{smallmatrix}\right]) = a^i b^j$ (that is, $a^i b^j \preceq_f a^k b^l$). If so, we accept, or else we reject. We then conclude that $x \in L_{ab}$ iff all computation paths of N are accepting. Since N is a co-1nfa, this characterization of L_{ab} implies that L_{ab} is in REG, a contradiction. □

In Proposition 12(1), we have already seen that CFL $\subseteq$ CFL2V-SEL. We further relate CFL $\subseteq$ CFLSV-SEL to a pseudo refinement of CFL2V$_{\mathrm{t}}$ by CFLSV$_{\mathrm{t}}$.

Lemma 14. *If* CFL $\subseteq$ CFLSV-SEL*, then* CFL2V$_{\mathrm{t}} \sqsubseteq^{(+)}_{ref}$ CFLSV$_{\mathrm{t}}$.

Proof. Assume that CFL $\subseteq$ CFLSV-SEL. Take any function $f \in$ CFL2V$_{\mathrm{t}}$ having its range D of size exactly 2. Let $D = \{y_1, y_2\}$ and let $d = \max\{|y| : y \in D\}$. Let M denote a 1npda computing f. Define $L = \{\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] \mid |x| \geq d, y \in f(x)\}$, which belongs to CFL since D is finite. Since $L \in$ CFLSV-SEL by our assumption, take a CFLSV-selector g for L and let N be its underlying 1npda that computes g with a write-once output tape. We then define $\tilde{g}$ as $\hat{g}(x) = g(\left[\begin{smallmatrix} u \\ v \end{smallmatrix}\right])$ for $u = \left[\begin{smallmatrix} x \\ y_1 \end{smallmatrix}\right]$ and $v = \left[\begin{smallmatrix} x \\ y_2 \end{smallmatrix}\right]$. Note that $\hat{g}$ is a single-valued function.

Consider the following 1npda, say, N. On input x, compute $\hat{g}$ on x and produce its outcome on an imaginary tape. Instead of producing an outcome of the form $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ with $y \in D$, write down y on the output tape. Let h be a function computed by this machine N. Note that h is in CFLSV$_t$ since f is a total function.

Finally, we want to show that $f \sqsubseteq^{(+)}_{ref} h$. Assume that $f(x) \neq \varnothing$. Consider the case where $f(x) = \{y_1, y_2\}$. Since $\left[\begin{smallmatrix} x \\ y_1 \end{smallmatrix}\right], \left[\begin{smallmatrix} x \\ y_2 \end{smallmatrix}\right] \in L$, $\hat{g}(x)$ must output a single string of the form $\left[\begin{smallmatrix} x \\ w \end{smallmatrix}\right]$ in L, where $w \in \{y_1, y_2\}$. By the definition of h, we obtain $h(x) = w$. The case of $f(x) = \{y\} \subseteq D$ is similarly treated. □

Lemma 14 helps us prove the following assertion.

Proposition 15. CFL2V-SEL = CFLSV-SEL $\Rightarrow$ CFL $\subseteq$ CFLSV-SEL $\Rightarrow$ CFL2V$_{\mathrm{t}}$-SEL = CFLSV$_{\mathrm{t}}$-SEL.

Proof. Assume that CFLSV-SEL = CFL2V-SEL. Since CFL $\subseteq$ CFL2V-SEL by Proposition 12(1), it follows that CFL $\subseteq$ CFLSV-SEL. Assuming CFL $\subseteq$ CFLSV-SEL, we show that CFL2V$_{\mathrm{t}}$-SEL = CFLSV$_{\mathrm{t}}$-SEL. For any language $L \in$ CFL2V$_{\mathrm{t}}$-SEL, take a CFL2V$_{\mathrm{t}}$-selector f for L. By Lemma 14, there exists a $g \in$ CFLSV$_{\mathrm{t}}$ satisfying $f \sqsubseteq^{(+)}_{ref} g$. Since $dom(f) \subseteq dom(g)$ and $g(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) \subseteq f(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]) \subseteq \{x, y\}$, we conclude that g is a selector for L as well. Hence, L belongs to CFLSV$_{\mathrm{t}}$-SEL. □

We further discuss the limitations of CFLSV$_{\mathrm{t}}$-SEL and CFLMV(2)-SEL. It is known that DCFL $\not\subseteq$ REG/n [12] and CFL(2) $\not\subseteq$ CFL/n [21, Corollary 3.10]. These separation results together with Proposition 12 yields the following immediate consequence.

Proposition 16. *(1)* $\mathrm{CFLSV_t\text{-}SEL} \not\subseteq \mathrm{REG}/n$. *(2)* $\mathrm{CFL2V(2)\text{-}SEL} \not\subseteq \mathrm{CFL}/n$.

Proof. (1) Assume that $\mathrm{CFLSV_t\text{-}SEL} \subseteq \mathrm{REG}/n$. By Proposition 12(2), we conclude that $\mathrm{CFL} \cap \text{co-CFL} \subseteq \mathrm{REG}/n$. Since $\mathrm{DCFL} \subseteq \mathrm{CFL} \cap \text{co-CFL}$, we obtain $\mathrm{DCFL} \subseteq \mathrm{REG}/n$. This contradicts the separation $\mathrm{DCFL} \not\subseteq \mathrm{REG}/n$ [12].

(2) Similarly to (1), assume that $\mathrm{CFL2V(2)\text{-}SEL} \subseteq \mathrm{CFL}/n$. Since $\mathrm{CFL}(2) \subseteq \mathrm{CFL2V(2)\text{-}SEL}$ by Proposition 12(3), we conclude that $\mathrm{CFL}(2) \subseteq \mathrm{CFL}/n$, a contradiction against $\mathrm{CFL}(2) \not\subseteq \mathrm{CFL}/n$ [21]. □

5 Extensions to Higher Complexity Classes over CFL

Analogously to the polynomial(-time) hierarchy over P and NP, the CFL *hierarchy* was introduced in [19] over DCFL and CFL by way of respectively viewing DCFL and CFL as P and NP. We quickly review the definition of this intriguing hierarchy. An *oracle 1npda* is a 1npda equipped with a write-once query tape by which the 1npda makes queries (adaptively) to a given oracle. Such a query tape is marked by two endmarkers $\{¢, \$\}$, where $\$$ is placed on a tape cell whose index is bounded by $O(n)$ [19]. Every time when a query is made, the query tape is automatically initialized with the endmarkers. The role of the oracle is to reset the oracle 1npda's inner state to a "yes" state (q_{yes}) or a "no" state (q_{no}) according to the case where a query word belongs to the oracle or it does not, respectively. Given a language A, the notation CFL_T^A (or $\mathrm{CFL}_T(A)$) expresses the collection of all languages recognized by those oracle 1npda's with an access to the oracle A. For a language family $\mathcal{C}$, we use the notation $\mathrm{CFL}_T^{\mathcal{C}}$ (or $\mathrm{CFL}_T(\mathcal{C})$) for the union $\bigcup_{A\in\mathcal{C}} \mathrm{CFL}_T^A$. The *CFL hierarchy* is $\{\Delta_k^{\mathrm{CFL}}, \Sigma_k^{\mathrm{CFL}}, \Pi_k^{\mathrm{CFL}} \mid k \in \mathbb{N}^+\}$ [19], where $\Delta_1^{\mathrm{CFL}} = \mathrm{DCFL}$, $\Sigma_1^{\mathrm{CFL}} = \mathrm{CFL}$, $\Pi_k^{\mathrm{CFL}} = \text{co-}\Sigma_k^{\mathrm{CFL}}$, $\Sigma_{k+1}^{\mathrm{CFL}} = \mathrm{CFL}_T(\Pi_k^{\mathrm{CFL}})$, and $\Delta_{k+1}^{\mathrm{CFL}} = \mathrm{DCFL}_T(\Pi_k^{\mathrm{CFL}})$ for each $k \geq 1$.

We have seen a lower bound of the complexity of $\mathrm{CFLSV_t\text{-}SEL}$ in Proposition 12. In what follows, we show an upper bound of the complexity of $\mathrm{CFLSV_t\text{-}SEL}$. In comparison, we note that $\text{P-SEL} \subseteq \mathrm{NP/lin} \cap \text{co-NP/lin}$ [4]. It is important to note that our advice can use an arbitrary alphabet, not necessarily limited to $\{0, 1\}$ in the case of NP/lin.

Proposition 17. $\mathrm{CFLSV_t\text{-}SEL} \subseteq \mathrm{CFL}_T(\mathrm{CFL}(2))/n \cap \text{co-}\mathrm{CFL}_T(\mathrm{CFL}(2))/n$.

Proof. Since co-($\mathrm{CFLSV_t\text{-}SEL}$) = $\mathrm{CFLSV_t\text{-}SEL}$ by Lemma 5, it suffices to prove that $\mathrm{CFLSV_t\text{-}SEL} \subseteq \mathrm{CFL}_T(\mathrm{CFL}(2))/n$. Let L be any language with a symmetric $\mathrm{CFLSV_t}$-selector f. Let $\tilde{f}$ denote the endmarked extension of f. We consider a tournament (graph) $G = (V, E)$ induced by f as $V = L \cap \Sigma^n$ and $E = \{(a, b) \mid a \neq b, f(\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]) = b\}$. We use the following result by Hohn, Landau, and Vaughan (cited in [14] and also in [3]): given a k-tournament $G = (V, E)$, there is a special node from which all nodes can be reached via paths of length at most 2.

We use the advice alphabet $\{0, 1, a\}$ and define $h(n) = a^n$ if $L \cap \Sigma^n = \varnothing$, and w_n otherwise, where w_n ($\in L$) is the special player (i.e., node) of the tournament

from which all the other players (nodes) can be reached along paths of length at most 2. We define $A = \{[\begin{smallmatrix} u \\ w \end{smallmatrix}] \mid u = [\begin{smallmatrix} x \\ z \end{smallmatrix}], (w, x) \in E \vee [(w, z) \in E \wedge (z, x) \in E]\}$. We claim that A is in CFL(2). We define B_1 as follows: on input $[\begin{smallmatrix} u \\ w \end{smallmatrix}]$ with $u = [\begin{smallmatrix} x \\ z \end{smallmatrix}]$, nondeterministically check one of the following two conditions: (i) $\tilde{f}([\begin{smallmatrix} w\$_1 \\ x\$_1 \end{smallmatrix}]) = x\$_2$ and (ii) $w \neq z$ and $\tilde{f}([\begin{smallmatrix} w\$_1 \\ z\$_2 \end{smallmatrix}]) = z\$_2$. Let B_2 be defined similarly by replacing (ii) with the following condition: (ii′) $z \neq x$ and $\tilde{f}([\begin{smallmatrix} z\$_1 \\ x\$_2 \end{smallmatrix}]) = x\$_2$. It follows that $B_1, B_2 \in$ CFL. Since $A = B_1 \cap B_2$, we obtain $A \in$ CFL(2). Next, we define an oracle 1npda N as follows: on input $[\begin{smallmatrix} u \\ w \end{smallmatrix}]$, guess z and query $[\begin{smallmatrix} u \\ w \end{smallmatrix}]$ with $u = [\begin{smallmatrix} x \\ z \end{smallmatrix}]$ to A. If A answers affirmatively, then N accepts; otherwise, N rejects. Since $L = \{x \mid [\begin{smallmatrix} x \\ h(|x|) \end{smallmatrix}] \in A\}$, L belongs to $\mathrm{CFL}_T(A)/n \subseteq \mathrm{CFL}_T(\mathrm{CFL}(2))/n$. The second part CFLSV$_{\mathrm{t}}$-SEL $\subseteq$ co-CFL$_T$(CFL(2))$/n$ follows from the fact that CFLSV$_{\mathrm{t}}$-SEL is closed under complementation. □

Multi-valued functional versions of CFL_T^A and $\mathrm{CFL}_T^{\mathcal{C}}$ are denoted respectively by CFLMV_T^A (or $\mathrm{CFLMV}_T(A)$) and $\mathrm{CFLMV}_T^{\mathcal{C}}$ (or $\mathrm{CFLMV}_T(\mathcal{C})$) [18]. The *CFLMV hierarchy* then consists of language families: $\Sigma_1^{\mathrm{CFL}}\mathrm{MV} = \mathrm{CFLMV}$, and $\Sigma_{k+1}^{\mathrm{CFL}}\mathrm{MV} = \mathrm{CFLMV}_T(\Pi_k^{\mathrm{CFL}})$, and $\Pi_k^{\mathrm{CFL}}\mathrm{MV} =$ co-$\Sigma_k^{\mathrm{CFL}}\mathrm{MV}$ for every level $k \geq 1$ [20]. Similarly, $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}$ is defined for every $k \geq 1$.

Here, we only remark that Lemmas 4–5, Theorem 10, and Proposition 12(1) are easily extended to an arbitrary level $k \geq 1$. For instance, a generalization of Proposition 12(1) is stated as follows.

Proposition 18. *Let $k \geq 1$. (1) $\Sigma_k^{\mathrm{CFL}} \subseteq \Sigma_k^{\mathrm{CFL}}$2V-SEL. (2) $\Pi_k^{\mathrm{CFL}} \subseteq \Sigma_{k+1}^{\mathrm{CFL}}$2V-SEL.*

Corollary 19. *For any level $k \geq 1$, if $\Pi_k^{\mathrm{CFL}} \not\subseteq \Sigma_k^{\mathrm{CFL}}$2V-SEL, then Σ_k^{CFL}2V-SEL $\neq \Sigma_{k+1}^{\mathrm{CFL}}$2V-SEL.*

Proof. Since $\Pi_k^{\mathrm{CFL}} \subseteq \Sigma_{k+1}^{\mathrm{CFL}}$2V-SEL holds by Proposition 18, the assumption $\Pi_k^{\mathrm{CFL}} \not\subseteq \Sigma_k^{\mathrm{CFL}}$2V-SEL implies Σ_k^{CFL}2V-SEL $\neq \Sigma_{k+1}^{\mathrm{CFL}}$2V-SEL. □

Proposition 12(2) can be also extended into a higher level; however, its proof is more involved than that of the proposition.

Theorem 20. *For any level $k \geq 1$, $\Sigma_k^{\mathrm{CFL}} \cap \Pi_k^{\mathrm{CFL}} \subseteq \Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$-SEL.*

Corollary 21. *Let $k \geq 2$. If $\Sigma_k^{\mathrm{CFL}} = \Pi_k^{\mathrm{CFL}}$, then $\Sigma_k^{\mathrm{CFL}} \subseteq \Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$-SEL.*

Proof of Theorem 20. We first note the following claim.

Claim **22** *For any index $k \geq 1$. Given a language A, $A \in \Sigma_k^{\mathrm{CFL}} \cap \Pi_k^{\mathrm{CFL}}$ iff $\chi_A \in \Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$.*

The case of $k = 1$ of Claim 22 was already shown as [21, Lemma 2.1] and its generalization for $k \geq 2$ can be similarly proven.

Let $A \in \Sigma_k^{\mathrm{CFL}} \cap \Pi_k^{\mathrm{CFL}}$. Claim 22 implies that χ_A is in $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$. Define $f([\begin{smallmatrix} x \\ y \end{smallmatrix}]) = [\begin{smallmatrix} u \\ v \end{smallmatrix}]$, where $u = \chi_A(x)$ and $v = \chi_A(y)$.

Next, we claim the following.

Claim **23** *Let $k \geq 2$ and let p be a function in $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$. The function g defined as $g([\begin{smallmatrix} x \\ y \end{smallmatrix}]) = [\begin{smallmatrix} p(x) \\ p(y) \end{smallmatrix}]$ for all $x, y \in \Sigma^*$ belongs to $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$.*

The proof of this claim is quite technical. One may prove it by first establishing that underlying oracle 1npda's can be replaced by oracle 1nfa's having access to Dyck languages, as shown for Σ_k^{CFL} in [19, arXiv version].

Proof of Claim 23. This proof relies on a result of [19, arXiv version]. Let $k \geq 2$ and let $p \in \Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}} \cap \mathcal{F}_{fin}$. Recall that, in [19], for any language A, two language families $\Sigma_{m,k}^{\mathrm{NFA},A}$ and $\Pi_{m,k}^{\mathrm{NFA},A}$ were introduced. In a similar way, we can define $\Sigma_{m,k}^{\mathrm{NFA}}\mathrm{SV}_{\mathrm{t}}{}^{A}$ and $\Pi_{m,k}^{\mathrm{NFA}}\mathrm{SV}_{\mathrm{t}}{}^{A}$ as follows. Let $\Sigma_{m,1}^{\mathrm{NFA}}\mathrm{SV}_{\mathrm{t}}{}^{A} = (\mathrm{NFASV}_{\mathrm{t}})_m^A$, $\Pi_{m,k}^{\mathrm{NFA}}\mathrm{SV}_{\mathrm{t}}{}^{A} = \text{co-}\Sigma_{m,k}^{\mathrm{NFA}}\mathrm{SV}_{\mathrm{t}}{}^{A}$, and $\Sigma_{m,k+1}^{\mathrm{NFA}}\mathrm{SV}_{\mathrm{t}}{}^{A} = (\mathrm{NFASV}_{\mathrm{t}})_m^{\Pi_{m,k-1}^{\mathrm{NFA},A}}$ for any $k \geq 1$. Following an argument given in [19, arXiv version], we obtain the following claim.

Claim **24** *For any $k \geq 1$, $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}} = \Sigma_{m,k}^{\mathrm{NFA}}\mathrm{SV}_{\mathrm{t}}{}^{DYCK}$, where $DYCK$ denotes the Dyck language.*

Since $f \in \Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$, take an oracle 1nfa M that computes f with access to oracle B in $\Pi_{m,k-1}^{\mathrm{NFA},DYCK}$. Consider the following machine N. On input $[\begin{smallmatrix} x \\ y \end{smallmatrix}]$, N simulates in parallel both $M(x)$ and $M(y)$ without using any stack. If $M(x)$ produces u and $M(y)$ produces w on their query tapes, then N produces $[\begin{smallmatrix} \tilde{u} \\ \tilde{w} \end{smallmatrix}]$ on its query tape, where $\tilde{u}$ and $\tilde{w}$ are $\natural$-extensions of u and w. Moreover, N outputs $[\begin{smallmatrix} a \\ b \end{smallmatrix}]$, where a and b are outcomes of M on x and y, respectively. Finally, define $C = \{[\begin{smallmatrix} \tilde{u} \\ \tilde{w} \end{smallmatrix}] \mid u, w \in B\}$. It was shown in [19] that $\Pi_{k-1}^{\mathrm{CFL}} = \Pi_{m,k-1}^{\mathrm{NFA},DYCK}$. Since $C \in \Pi_{k-1}^{\mathrm{CFL}}$, we obtain $C \in \Pi_{m,k-1}^{\mathrm{NFA},DYCK}$. By the definition of N and C, the function defined by $g([\begin{smallmatrix} x \\ y \end{smallmatrix}]) = [\begin{smallmatrix} f(x) \\ f(y) \end{smallmatrix}]$ is computed by N with the oracle C. Therefore, g belongs to $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$. □

By Claim 24, it follows that the function f belongs to $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$. Next, consider the following oracle 1npda, say, N. On input $[\begin{smallmatrix} x \\ y \end{smallmatrix}]$, guess $b \in \{0, 1\}$ and write x (resp., y) on an output tape if $b = 0$ (resp., $b = 1$). At the same time, calculate $f([\begin{smallmatrix} x \\ y \end{smallmatrix}])$ and let z be its valid outcome (along a certain accepting computation path). If $b = 0$ and z is either $[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}]$ or $[\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}]$, then N accepts. In the case where z is $[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}]$ (resp., $[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}]$), N accepts if $b = 1$ (resp., $b = 0$). In all other cases, N rejects. Notice that N runs with synchronized tape heads. Let h denote the function computed by N. It is not difficult to show that h is indeed a selector for A. □

Let us consider a basic relationship between "refinement" and "selectivity."

Proposition 25. *For each level $k \geq 2$, $\Sigma_k^{\mathrm{CFL}}\mathrm{2V} \sqsubseteq_{ref}^{(+)} \Sigma_k^{\mathrm{CFL}}\mathrm{SV}$ implies $\Sigma_k^{\mathrm{CFL}} \subseteq \Sigma_k^{\mathrm{CFL}}\text{SV-SEL}$. The same statement holds when $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}$ is replaced by $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$.*

Proposition 26. *Let $k \geq 2$. If $\Sigma_k^{\mathrm{CFL}} = \Pi_k^{\mathrm{CFL}}$, then $\Sigma_k^{\mathrm{CFL}}\mathrm{2V} \sqsubseteq_{ref}^{(+)} \Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$.*

Proof. It is shown in [19, arXiv version] that $\Sigma_k^{\mathrm{CFL}} = \Pi_k^{\mathrm{CFL}}$ implies $\Sigma_k^{\mathrm{CFL}} = \Sigma_{k+1}^{\mathrm{CFL}}$. From $\Sigma_k^{\mathrm{CFL}} = \Pi_k^{\mathrm{CFL}}$, it follows that $\Sigma_{k+1}^{\mathrm{CFL}}\mathrm{MV} \sqsubseteq_{ref} \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV}$ [20]; actually, we can improve this to (*) $\Sigma_k^{\mathrm{CFL}}\mathrm{MV} \sqsubseteq_{ref} \Sigma_k^{\mathrm{CFL}}\mathrm{SV}$.

Claim **27** *For every $k \geq 2$, $\Sigma_k^{\mathrm{CFL}}\mathrm{SV} \sqsubseteq_{ref}^{(+)} \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$.*

Proof of Claim 27. Let $f \in \Sigma_k^{\mathrm{CFL}}\mathrm{SV}$. Assume that $k \geq 2$. Since $\Sigma_k^{\mathrm{CFL}}\mathrm{SV} = \mathrm{CFLSV}_T(\Pi_{k-1}^{\mathrm{CFL}})$ by the definition, take an oracle 1npda M and an oracle B in Π_{k-1}^{CFL} computing f. Let $C = \{1y \mid y \in B\} \cup \{0x \mid f(x) \neq \varnothing\}$. Since $\{1y \mid y \in B\}$ is in Π_{k-1}^{CFL} and $\{0x \mid f(x) \neq \varnothing\}$ is in Σ_k^{CFL}, it follows that C belongs to Σ_k^{CFL}. Consider the following machine. On input x, guess $b \in \{0, 1\}$. If $b = 1$, then simulate M with C as an oracle. If $b = 0$, then query $0x$ to C. If C's answer is "yes," then reject; if the answer is "no," then accept. Moreover, whenever M queries y, N queries $1y$.

Let g denote the function computed by N with C. Note that g belongs to $\mathrm{CFLSV}_{\mathrm{t}}{}^{C} \subseteq \mathrm{CFLSV}_{\mathrm{t}}{}^{\Sigma_k^{\mathrm{CFL}}} \subseteq \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$. Moreover, it is not difficult to show that g is a total function and that, if $x \in \mathrm{dom}(f)$, $f(x) = g(x)$. Therefore, g is a pseudo refinement of f. □

By Claim 27 together with (*), we obtain $\Sigma_k^{\mathrm{CFL}}2\mathrm{V} \sqsubseteq_{ref}^{(+)} \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$. Since $\Sigma_k^{\mathrm{CFL}} = \Sigma_{k+1}^{\mathrm{CFL}}$, we obtain $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}} = \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$. Therefore, we can conclude that $\Sigma_k^{\mathrm{CFL}}2\mathrm{V} \sqsubseteq_{ref}^{(+)} \Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$.

Notice that combining Propositions 25 and 26 also leads to Corollary 21.

6 A Closing Discussion

The study of efficient selectivity was initiated in 1979 by Selman [9] using polynomial-time computable selectors and it has been since then expanded to other types of selectors. An introduction of selectivity to formal languages and automata theory was found in [13]. To further broaden the scope of study on the selectivity issues in formal languages and automata theory, we have considered in this paper one-way nondeterministic pushdown automata as an underlying machine model to compute selectors.

It is unfortunate that we have left numerous selectivity issues unsettled throughout this paper. For example, it is unclear at this moment that the introduction of the notion of "endmarked extension" of selectors in Definition 2 is truly necessary. We hope to prove that the use of this endmarked extension is actually redundant and we can simplify Definition 2 without endmarked extension.

References

1. Buhrman, H., Torenvliet, L.: P-selective self-reducible sets: a new characterization of P. J. Comput. System Sci. **53**, 210–217 (1996)

2. Hemachandra, L.A., Hoene, A., Naik, A.V., Ogiwara, M., Selman, A.L., Thierauf, T., Wang, J.: Nondeterministically selective sets. Int. J. Found. Comput. Sci. **6**, 403–416 (1995)
3. Hemaspaandra, L., Nasipak, C., Parkins, K.: A note on linear-nondeterminism, linear-sized, Karp-Lipton advice for the P-selective sets. J. Universal Computer Science **4**, 670–674 (1998)
4. Hemaspaandra, L., Torenvliet, L.: Optimal advice. Theor. Comput. Sci. **154**, 367–377 (1996)
5. Hemaspaandra, L.A., Jiang, Z.: P-selectivity: intersection and indices. Theor. Comput. Sci. **145**, 371–380 (1995)
6. Hemaspaandra, L.A., Naik, A.V., Ogihara, M., Selman, A.L.: Computing solutions uniquely collaspses the polynomial hierarchy. SIAM J. Comput. **25**, 697–708 (1996)
7. Ko, K.: On self-reducibility and weak P-selectivity. J. Comput. System Sci. **26**, 209–221 (1983)
8. Meyer, A.R., Stockmeyer, L.J.: 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)
9. Selman, A.L.: P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. Math. Systems Theory **13**, 55–65 (1979)
10. Selman, A.L.: Analogues of semirecursive sets and effective reducibilities to the study of NP complexity. Inform. Control **52**, 36–51 (1982)
11. Selman, A.L.: A taxonomy of complexity classes of functions. J. Comput. System Sci. **48**, 357–381 (1994)
12. Tadaki, K., Yamakami, T., Lin, J.C.H.: Theory of one-tape linear-time Turing machines. Theor. Comput. Sci. **411**, 22–43 (2010)
13. Tantau, T.: On the structural similarities of finite automata and Turing machine enumerability classes. Doctoral Dissertation, Elektrotechnik und Informatik der Technischen Universität Berlin (2003)
14. West, D.: Introduction to Graph Theory. Prentice Hall (1996)
15. Yamakami, T.: Swapping lemmas for regular and context-free languages (2008), manuscript. Available at arXiv:0808.4122, 2008.
16. 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)
17. Yamakami, T.: Immunity and pseudorandomness of context-free languages. Theor. Comput. Sci. **412**, 6432–6450 (2011)
18. Yamakami, T.: Not all multi-valued partial CFL functions are refined by single-valued functions (extended abstract). In: Proc. of IFIP TCS 2014. Lecture Notes in Computer Science, vol. 8705, pp. 136–150. Springer (2014), There is an error in the main proof.
19. Yamakami, T.: Oracle pushdown automata, nondeterministic reducibilities, and the hierarchy over the family of context-free languages. In: Proc. of SOFSEM 2014. CEUR Workshop Proceedings, vol. 8327, pp. 514–525. Springer (2014), A complete and corrected version appears at arXiv:1303.1717.
20. Yamakami, T.: Structural complexity of multi-valued partial functions computed by nondeterminsitic pushdown automata (extended abstract). In: Proc. of the 15th Italian Conference of Theoretical Computer Science. CEUR Workshop Proceedings, vol. 1231, pp. 225–236 (2014)
21. Yamakami, T.: Pseudorandom generators against advised context-free languages. Theor. Comput. Sci. **613**, 1–27 (2016)

22. Yamakami, T.: Intersection and union hierarchies of deterministic context-free languages and pumping lemmas. In: Proc. of LATA 2020. Lecture Notes in Computer Science, vol. 12038, pp. 341–353. Springer (2020)
23. Yamakami, T., Kato, Y.: The dissecting power of regular languages. Inf. Process. Lett. **113**, 116–122 (2013)