Oracle Pushdown Automata, Nondeterministic Reducibilities, and the Hierarchy over the Family of Context-Free Languages

Tomoyuki Yamakami

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

Abstract. We implement various oracle mechanisms on nondeterministic pushdown automata, which naturally induce nondeterministic reducibilities among formal languages in a theory of context-free languages. In particular, we examine a notion of nondeterministic many-one CFL-reducibility and carry out ground work of formulating a coherent framework for further expositions. Another more powerful reducibility—Turing CFL-reducibility—is also discussed in comparison. The Turing CFL-reducibility, in particular, makes it possible to induce a useful hierarchy (the CFL hierarchy) built over the family CFL of context-free languages. For each level of this hierarchy, basic structural properties are proven and three alternative characterizations are presented. We also show that the CFL hierarchy enjoys an upward collapse property. The first and second levels of the hierarchy are proven to be different. We argue that the CFL hierarchy coincides with a hierarchy over CFL built by applications of many-one CFL-reductions. Our goal is to provide a solid foundation for structural-complexity analyses in automata theory.

Keywords: regular language, context-free language, pushdown automaton, oracle, many-one reducibility, Turing reducibility, CFL hierarchy, polynomial hierarchy, Dyck language.

1 Backgrounds and Main Themes

A fundamental notion of *reducibility* has long played an essential role in the development of a theory of NP-completeness. In the 1970s, various forms of polynomial-time reducibility emerged, most of which were based on models of multi-tape oracle Turing machine, and they provided a technical means to study *relativizations* of associated families of languages. Most typical reducibilities in use today in computational complexity theory include many-one, truth-table, and Turing reducibilities obtained by imposing appropriate restrictions on the functionality of oracle mechanism of underlying Turing machines. Away from standard complexity-theoretical subjects, we will shift our attention to a theory of formal languages and automata. Within this theory, we wish to lay out a framework for a future extensive study on structural complexity issues by providing a solid foundation for various notions of reducibility and their associated relativizations.

Among many languages, we are particularly interested in *context-free languages*, which are characterized by context-free grammars or one-way nondeterministic pushdown automata (or npda's, hereafter). The context-free languages are inherently nondeterministic. In light of the fact that the notion of nondeterminism appears naturally in real life, this notion has become a key to many fields of computer science. The family CFL of context-free languages has proven to be a fascinating subject, simply because the languages in CFL behave quite differently from the languages in the corresponding nondeterministic polynomial-time class NP. For instance, whereas NP is closed under any Boolean operation (possibly) except for complementation, CFL is not even closed under intersection. This non-closure property is caused by the lack of flexibility in a use of its memory storage on an underlying model of npda. On the contrary, a restricted use of memory helps us prove a separation between the first and the second levels of the Boolean hierarchy $\{\mathrm{CFL}_k \mid k \geq 1\}$ built over CFL by applying alternatingly two operations of intersection and union to CFL [13]. Moreover, we can prove that a family of languages $\mathrm{CFL}(k)$ composed of intersections of k context-free languages truly forms an infinite hierarchy [7]. Such an architectural constraint sometimes becomes a crucial issue in certain applications of pushdown automata.

A most simple type of well-known reduction is probably *many-one reduction* and, by adopting the existing formulation of this reducibility, we intend to bring a notion of nondeterministic many-one reducibility into context-free languages under the name of *many-one CFL-reducibility*. We write CFL_m^A to denote the family of languages that are many-one CFL-reducible to oracle A. Notice that Reinhardt [8] earlier considered many-one reductions that are induced by nondeterministic finite automata (or nfa's), which use no memory space. We wish to build a hierarchy of language families over CFL using our new reducibility by immediate analogy with constructing the *polynomial(-time) hierarchy* over NP. For this purpose, we choose npda's rather than nfa's. Owing mostly to a unique architecture of npda's, our reducibility exhibits quite distinctive features; for instance, this reducibility in general does not admit a transitivity property. (For this reason, our reducibility might have been called a "quasi-reducibility" if the transitive property is a prerequisite for a reducibility notion.) As a consequence, the family CFL is not closed under the many-one CFL-reducibility (namely, $\mathrm{CFL}_m^{\mathrm{CFL}} \neq \mathrm{CFL}$). This non-closure property allures us to study the family $\mathrm{CFL}_{m[k]}^{\mathrm{CFL}}$ whose elements are obtained by the k-fold application of many-one CFL-reductions to languages in CFL. As shown in Section 3.1, the language family $\mathrm{CFL}_{m[k]}^{\mathrm{CFL}}$ turns out to coincide with $\mathrm{CFL}_m^{\mathrm{CFL}(k)}$.

We further discuss another more powerful reducibility in use—Turing CFL-reducibility based on npda's. This reducibility introduces a hierarchy analogous to the polynomial hierarchy: the hierarchy $\{\Delta_k^{\mathrm{CFL}}, \Sigma_k^{\mathrm{CFL}}, \Pi_k^{\mathrm{CFL}} \mid k \geq 1\}$ built over CFL, which we succinctly call the *CFL hierarchy*, and this hierarchy turns out to be quite useful in classifying the computational complexity of formal languages. As quick examples, two languages $Dup_2 = \{xx \mid x \in \{0,1\}^*\}$ and $Dup_3 = \{xxx \mid x \in \{0,1\}^*\}$, which are known to be outside of CFL, fall into the second level Σ_2^{CFL} of the CFL hierarchy. A simple matching language $Match = \{x\#w \mid \exists u, v\,[w = uxv]\}$ is also in Σ_2^{CFL}. Two more languages $Sq = \{0^n1^{n^2} \mid n \geq 1\}$ and $Prim = \{0^n \mid n \text{ is a prime number}\}$ belong to Σ_2^{CFL} and Π_2^{CFL}, respectively. A slightly more complex language $MulPrim = \{0^{mn} \mid m \text{ and } n \text{ are prime numbers}\}$ is a member of Σ_3^{CFL}. The first and second levels of the CFL hierarchy are easily proven to be different. Regarding the aforementioned language families $\mathrm{CFL}(k)$ and CFL_k, we can show later that the families $\mathrm{CFL}(\omega) = \bigcup_{k\geq 1} \mathrm{CFL}(k)$ and $\mathrm{BHCFL} = \bigcup_{k\geq 1} \mathrm{CFL}_k$ belong to $\Sigma_2^{\mathrm{CFL}} \cap \Pi_2^{\mathrm{CFL}}$ of the CFL hierarchy. In Section 4.1, we show that $\mathrm{CFL}_m^{\mathrm{CFL}(\omega)}$ is located within Σ_3^{CFL}. Despite obvious similarities between their definitions, the CFL hierarchy and the polynomial hierarchy are quite different in nature. Because of npda's architectural restrictions, "standard" techniques of simulating a two-way Turing machine, in general, do not apply; hence, we need to develop new simulation techniques for npda's.

In this paper, we employ three simulation techniques to obtain some of the aforementioned results. The first technique is of guessing and verifying a *stack history* to eliminate a use of stack, where a stack history roughly means a series of consecutive stack operations made by an underlying npda. The second technique is applied to the case of simulating two or more tape heads by a single tape head. To adjust the different head speeds, we intentionally insert extra dummy symbols to generate a single query word so that an oracle can eliminate them when it accesses the query word. The last technique is to generate a string that encodes a computation path generated by a nondeterministic machine. All the techniques are explained in details in Sections 3.1–3.2. Those simulation techniques actually make it possible to obtain three alternative characterizations of the CFL hierarchy in Section 4.2.

Topics excluded from this extended abstract are found in its full version available at arXiv:1303.1717.

2 Preparation

Given a finite set A, the notation $|A|$ expresses the number of elements in A. Let $\mathbb{N}$ be the set of all *natural numbers* (i.e., nonnegative integers) and set $\mathbb{N}^+ = \mathbb{N} - \{0\}$. For any number $n \in \mathbb{N}^+$, $[n]$ denotes the integer set $\{1, 2, \ldots, n\}$. The term "polynomial" always means a polynomial on $\mathbb{N}$ with coefficients of nonnegative integers. In particular, a *linear polynomial* is of the form $ax + b$ with $a, b \in \mathbb{N}$. The notation $A - B$ for two sets A and B indicates the *difference* $\{x \mid x \in A, x \notin B\}$ and $\mathcal{P}(A)$ denotes the *power set* of A. The *Kleene closure* Σ^* of Σ is the infinite union $\bigcup_{k\in\mathbb{N}} \Sigma^k$. Similarly, the notation $\Sigma^{\leq k}$ is used to mean $\bigcup_{i=1}^{k} \Sigma^i$. Given a language A over Σ, its *complement* is $\Sigma^* - A$, which is also denoted by $\overline{A}$. We use the following three class operations between two language families $\mathcal{C}_1$ and $\mathcal{C}_2$: $\mathcal{C}_1 \wedge \mathcal{C}_2 = \{A \cap B \mid A \in \mathcal{C}_1, B \in \mathcal{C}_2\}$, $\mathcal{C}_1 \vee \mathcal{C}_2 = \{A \cup B \mid A \in \mathcal{C}_1, B \in \mathcal{C}_2\}$, and $\mathcal{C}_1 - \mathcal{C}_2 = \{A - B \mid A \in \mathcal{C}_1, B \in \mathcal{C}_2\}$, where A and B must be defined over the same alphabet. For a use of *track notation* $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$, see [9].

As our basic computation models, we use the following types of finite-state machines: *one-way deterministic finite automaton* (or dfa, in short) with λ-moves and *one-way nondeterministic pushdown automaton* (or npda) with λ-moves, where a *λ-move* (or a *λ-transition*) is a transition of the machine's configurations in which a target tape head stays still. 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. Therefore, all cells through which the tape head passes during a computation must contain no blank symbols. An *output* (or *outcome*) along a computation path is a string produced on the output tape after the computation path is terminated. We call an output string *valid* (or *legitimate*) if it is produced along a certain accepting computation path. When we refer to the machine's outputs, we normally disregard any invalid strings left on the output tape on a rejecting computation path. REG, CFL, and DCFL stand for the families of all regular languages, of all context-free languages, and of all deterministic context-free languages, respectively.

3 Natural Reducibilities

A typical way of comparing the computational complexity of two formal languages is various forms of *resource-bounded reducibility*. Such reducibility is also regarded as a *relativization* of its underlying language family. We refer the reader to [2] for basics of computational complexity theory.

3.1 Many-One Reductions by Npda's

Our exposition begins with an introduction of an appropriate form of nondeterministic many-one reducibility whose reductions are operated by npda's.

Our "reduction machine" is essentially a restricted version of "pushdown transducer" or "algebraic transduction" (see, e.g., [1]). Here, we define this notion in a style of "oracle machine." An *m-reduction npda M* is a standard npda equipped with an extra *query tape* on which the machine writes a string surrounded by blank cells starting at the designated *start cell* for the purpose of making a query to a given *oracle*. We treat this query tape as an output tape, and thus the query-tape head must move to a next blank cell whenever it writes a non-blank symbol. Formally, an m-reduction npda is a tuple $(Q, \Sigma, \{¢, \$\}, \Theta, \Gamma, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$, where Θ is a query alphabet and δ is of the form: $\delta : (Q - Q_{halt}) \times (\check{\Sigma} \cup \{\lambda\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^* \times (\Theta \cup \{\lambda\}))$, where $Q_{halt} = Q_{acc} \cup Q_{rej}$ and $\check{\Sigma} = \Sigma \cup \{¢, \$\}$. There are two types of λ-moves. Assuming $(p, \tau, \xi) \in \delta(q, \sigma, \gamma)$, if $\sigma = \lambda$, then the input-tape head stays still (or makes a λ-move); in contrast, if $\tau = \lambda$, then the query-tape head stays still (or makes a λ-move). Since repetitions of λ-moves potentially produce extremely long output strings, we should require the following *termination condition* for M. Recall that, as a consequence of Greibach's normal form theorem, all context-free languages can be recognized by *λ-free* npda's (i.e., npda's with no λ-moves) whose computation paths have length $O(n)$, always ending in certain halting states, where n is its input size. The runtime of $O(n)$ is truly significant for languages in CFL. Likewise, we assume that, for any m-reduction npda, *all* computation paths should terminate (reaching halting inner states) within $O(n)$ time.

A language L over alphabet Σ is *many-one CFL-reducible* to another language A over alphabet Θ if there exists an m-reduction npda $M = (Q, \Sigma, \{\cent, \$\}, \Theta, \Gamma, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$ such that, for every input $x \in \Sigma^*$, (1) along each computation path $p \in ACC_M(x)$, M produces a valid query string $y_p \in \Theta^*$ on the query tape and (2) x is a member of L iff there is a computation path $p \in ACC_M(x)$ satisfying $y_p \in A$. For simplicity, we also say that M *reduces* (or *m-reduces*) L to A. With the use of this new reducibility, we make the notation CFL_m^A (or $\mathrm{CFL}_m(A)$) express the family of all languages L that are many-one CFL-reducible to A, where the language A is customarily called an *oracle*. Given an oracle npda M and an oracle A, the notation $L(M, A)$ (or $L(M^A)$) denotes the set of strings accepted by M relative to A. For a class $\mathcal{C}$ of oracles, $\mathrm{CFL}_m^{\mathcal{C}}$ (or $\mathrm{CFL}_m(\mathcal{C})$) denotes the union $\bigcup_{A \in \mathcal{C}} \mathrm{CFL}_m^A$.

Likewise, we define the relativized language family NFA_m^A (or $\mathrm{NFA}_m(A)$) using "nfa's" as m-reduction machines instead of "npda's." To be more precise, an *m-reduction nfa M* for NFA_m^A is a tuple $(Q, \Sigma, \{\cent, \$\}, \Theta, \delta, q_0, Q_{acc}, Q_{rej})$, where δ is a map from $(Q - Q_{halt}) \times (\check{\Sigma} \cup \{\lambda\})$ to $\mathcal{P}(Q \times (\Theta \cup \{\lambda\}))$. We also impose an $O(n)$ time-bound on all computation paths of M.

Making an analogy with "oracle Turing machine" that functions as a mechanism of reducing a language to another given target language A, we want to use the term "oracle npda" to mean an npda that is equipped with an extra write-only output tape (called a *query tape*) besides a read-only input tape. As noted before, we explicitly demand every oracle npda to terminate on all computation paths within $O(n)$ steps.

We will use an informal term of "guessing" when we refer to a nondeterministic choice (or a series of nondeterministic choices). For example, when we say that an npda M guesses a string z, we actually mean that M makes a series of nondeterministic choices that cause to produce z.

Example 1. As the first concrete example, setting $\Sigma = \{0, 1\}$, let us consider the language $Dup_2 = \{xx \mid x \in \Sigma^*\}$. This language is known to be non-context-free; however, it can be many-one CFL-reducible to CFL by the following M and A. An m-reduction (or oracle) npda M nondeterministically produces a query word $x^R\natural y$ (with a special symbol $\natural$) from each input of the form xy using a stack appropriately More formally, a transition function δ of this oracle npda M is given as follows: $\delta(q_0, \cent, Z_0) = \{(q_0, Z_0, \lambda)\}$, $\delta(q_0, \$, Z_0) = \{(q_{acc}, Z_0, \natural)\}$, $\delta(q_0, \sigma, Z_0) = \{(q_1, \sigma Z_0, \lambda)\}$, $\delta(q_1, \sigma, \tau) = \{(q_1, \sigma\tau, \lambda), (q_2, \sigma\tau, \lambda)\}$, $\delta(q_2, \lambda, \tau) = \{(q_2, \lambda, \tau)\}$, $\delta(q_2, \lambda, Z_0) = \{(q_3, Z_0, \natural)\}$, $\delta(q_3, \lambda, Z_0) = \{(q_3, Z_0, \sigma)\}$, and $\delta(q_3, \$, Z_0) = \{(q_{acc}, Z_0, \lambda)\}$, where $\sigma, \tau \in \Sigma$. A CFL-oracle A is defined as $\{x^R\natural x \mid x \in \Sigma^*\}$; that is, the oracle A checks whether $x = y$ from the input $x^R\natural y$ using its own stack. In other words, Dup_2 belongs to CFL_m^A, which is included in $\mathrm{CFL}_m^{\mathrm{CFL}}$. Similarly, the non-context-free language $Dup_3 = \{xxx \mid x \in \Sigma^*\}$ also falls into $\mathrm{CFL}_m^{\mathrm{CFL}}$. For this case, we design an m-reduction npda to produce $x^R\natural y\natural y^R\natural z$ from each input xyz and make a CFL-oracle check whether $x = y = z$

by using its stack twice. Another language $Match = \{x\#w \mid \exists u, v\,[w = uxv]\,\}$, where $\#$ is a separator not in x and w, also belongs to $\mathrm{CFL}_m^{\mathrm{CFL}}$. These examples prove that $\mathrm{CFL}_m^{\mathrm{CFL}} \neq \mathrm{CFL}$.

Example 2. The language $Sq = \{0^n 1^{n^2} \mid n \geq 1\}$ belongs to $\mathrm{CFL}_m^{\mathrm{CFL}}$. To see this fact, let us consider the following oracle npda N and oracle A. Given any input w, N first checks if w is of the form $0^i 1^j$. Simultaneously, N nondeterministically selects $(j_1, j_2, \ldots, j_k)$ satisfying (i) $j = j_1 + j_2 + \cdots + j_k$ and (ii) $j_1 = j_2$, $j_3 = j_4$, …, and N produces on its query tape a string w' of the form $0^i \natural 1^{j_1} \natural 1^{j_2} \natural \cdots \natural 1^{j_k}$. The desired oracle A receives w' and checks if the following two conditions are all met: (i') $j_2 = j_3$, $j_4 = j_5$, … and (ii') $i = k$ by first pushing 0^i into a stack and then counting the number of $\natural$. Clearly, A belongs to CFL. Therefore, Sq is in CFL_m^A, which is included in $\mathrm{CFL}_m^{\mathrm{CFL}}$. A similar idea proves that the language $Comp = \{0^n \mid n \text{ is a composite number}\}$ belongs to $\mathrm{CFL}_m^{\mathrm{CFL}}$. In symmetry, $Prim = \{0^n \mid n \text{ is a prime number}\}$ is a member of co-$(\mathrm{CFL}_m^{\mathrm{CFL}})$, where co-$\mathcal{C}$ denotes the *complement* of language family $\mathcal{C}$, namely, co-$\mathcal{C} = \{\overline{A} \mid A \in \mathcal{C}\}$.

A *Dyck language* L over alphabet $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_d\} \cup \{\sigma_1', \sigma_2', \ldots, \sigma_d'\}$ is a language generated by a deterministic context-free grammar whose production set is $\{S \to \lambda | SS | \sigma_i S \sigma_i' : i \in [d]\}$, where S is a start symbol. For convenience, denote by $DYCK$ the family of all Dyck languages.

Lemma 1. $\mathrm{CFL}_m^{\mathrm{CFL}} = \mathrm{CFL}_m^{\mathrm{DCFL}} = \mathrm{CFL}_m^{DYCK}$.

Proof Sketch. We first claim that (1) $\mathrm{CFL} = \mathrm{NFA}_m^{DYCK}$ and (2) $\mathrm{CFL}_m^A = \mathrm{CFL}_m(\mathrm{NFA}_m^A)$ for any oracle A. The first claim (1) can be seen as a different form of Chomsky-Schützenberger theorem. To show (1), we employ a simple but useful technique of guessing a correct *stack history* (namely, a series of popped and pushed symbols along a halting computation path) and verifying its correctness. With an appropriate encoding method, we can claim that a stack history is correct iff its encoding belongs to a ceratin fixed Dyck language. Whenever an oracle npda tries to either push down symbols into its stack or pop up a symbol from the stack, instead of using an actual stack, we write down an encoded series of those symbols on a write-only query tape and then ask an oracle to verify that the series indeed encodes a correct stack history. We skip (2) due to the page limit. By combining the claims (1)–(2), it follows that $\mathrm{CFL}_m^{\mathrm{CFL}} = \mathrm{CFL}_m(\mathrm{NFA}_m^{DYCK}) \subseteq \mathrm{CFL}_m^{DYCK}$. □

Given each number $k \in \mathbb{N}^+$, the *k-conjunctive closure of CFL*, denoted $\mathrm{CFL}(k)$ in [12], is defined recursively as follows: $\mathrm{CFL}(1) = \mathrm{CFL}$ and $\mathrm{CFL}(k{+}1) = \mathrm{CFL}(k) \wedge \mathrm{CFL}$. These language families truly form an infinite hierarchy [7]. For convenience, we set $\mathrm{CFL}(\omega) = \bigcup_{k \in \mathbb{N}^+} \mathrm{CFL}(k)$. Hereafter, we will explore basic properties of $\mathrm{CFL}_m^{\mathrm{CFL}(k)}$.

The lack of the transitivity property of the many-one CFL-reducibility necessitates an introduction of a helpful abbreviation of a *k-fold application of the reductions*. For any given oracle A, we recursively set $\mathrm{CFL}_{m[1]}^A = \mathrm{CFL}_m^A$ and $\mathrm{CFL}_{m[k+1]}^A = \mathrm{CFL}_m(\mathrm{CFL}_{m[k]}^A)$ for each index $k \in \mathbb{N}^+$. Given any language family $\mathcal{C}$, the notation $\mathrm{CFL}_{m[k]}^{\mathcal{C}}$ denotes the union $\bigcup_{A \in \mathcal{C}} \mathrm{CFL}_{m[k]}^A$.

Theorem 1. *For every index $k \in \mathbb{N}^+$, $\mathrm{CFL}_m^{\mathrm{CFL}(k)} = \mathrm{CFL}_{m[k]}^{\mathrm{CFL}}$.*

Proof Sketch. When $k = 1$, it holds that $\mathrm{CFL}_{m[1]}^{\mathrm{CFL}} = \mathrm{CFL}_m^{\mathrm{CFL}} = \mathrm{CFL}_m^{\mathrm{CFL}(1)}$. Next, we will show that, for every index $k \geq 2$, $\mathrm{CFL}_{m[k]}^{\mathrm{CFL}} \subseteq \mathrm{CFL}_m^{\mathrm{CFL}(k)}$ holds. We are focused on the most important case of $k = 2$. This case follows from the claim that $\mathrm{CFL}_{m[2]}^{\mathrm{CFL}(r)} \subseteq \mathrm{CFL}_m^{\mathrm{CFL}(r)\wedge\mathrm{CFL}}$ for every index $r \in \mathbb{N}^+$. Let $L \in \mathrm{CFL}_m^B$ and $B \in \mathrm{CFL}_m^A$ for a certain set $A \in \mathrm{CFL}(r)$. Let M_1 and M_2 be two oracle npda's witnessing $L \in \mathrm{CFL}_m^B$ and $B \in \mathrm{CFL}_m^A$, respectively. Consider the following oracle npda N. Given input x, N simulates M_1 on x in the following way. Whenever M_1 tries to write a symbol, say, b on a query tape, N simulates, using an actual stack, several steps (including all consecutive λ-moves) of M_2 that can be made during reading b. By simulating M_2, N aims at producing an encoded stack history y of M_2 (on the upper track of a tape) and a query word z (on the lower track). Since the tape heads of M_2 on both input and query tapes may move in different speeds, we need to adjust their speeds by inserting a series of fresh symbol, say, $\natural$ between symbols of the stack history and the query word. For this purpose, it is useful to introduce a terminology to describe strings obtained by inserting $\natural$. Assuming that $\natural \notin \Sigma$, a *$\natural$-extension* of a given string x over Σ is a string $\tilde{x}$ over $\Sigma \cup \{\natural\}$ satisfying that x is obtained directly from $\tilde{x}$ simply by removing all occurrences of $\natural$ in $\tilde{x}$. For instance, if $x = 01101$, then $\tilde{x}$ may be $01\natural1\natural01$ or $011\natural\natural01\natural$. N actually produces $\left[\begin{smallmatrix} y \\ z \end{smallmatrix}\right]$ on the query tape. An appropriate oracle in $\mathrm{CFL}(r) \wedge \mathrm{CFL}$ can check its correctness. Thus, $L \in \mathrm{CFL}_m^{\mathrm{CFL}(r)\wedge\mathrm{CFL}}$. □

An immediate consequence is that $\mathrm{CFL}_m^{\mathrm{CFL}(\omega)} = \bigcup_{k\in\mathbb{N}^+} \mathrm{CFL}_{m[k]}^{\mathrm{CFL}}$.

3.2 Turing Reducibility by Npda's

We define a notion of *Turing CFL-reducibility* using a model of npda with a write-only query tape and three extra inner states q_{query}, q_{no}, and q_{yes} that represent a query signal and two possible oracle answers, respectively. More specifically, when an oracle npda enters q_{query}, it triggers a query, by which a query word is automatically transferred to an oracle, a query tape becomes blank, and its tape head instantly returns to the start cell. When the oracle returns its answer, either 0 (no) or 1 (yes), it automatically sets the oracle npda's inner state to q_{no} or q_{yes}, respectively. Such a machine is called a *T-reduction npda* (or just an *oracle npda* as before) and it is used to reduce a language to another language. To be more precise, an oracle npda is a tuple $(Q, \Sigma, \{\cent, \$\}, \Theta, \Gamma, \delta, q_0, Z_0, Q_{oracle}, Q_{acc}, Q_{rej})$, where $Q_{oracle} = \{q_{query}, q_{yes}, q_{no}\}$, Θ is a query alphabet and δ has the form: $\delta : (Q - Q_{halt} \cup \{q_{query}\}) \times (\check{\Sigma} \cup \{\lambda\}) \times \Gamma \to \mathcal{P}((Q - \{q_{yes}, q_{no}\}) \times \Gamma^* \times (\Theta \cup \{\lambda\}))$.

Unlike many-one CFL-reductions, a T-reduction npda's computation depends on a series of oracle answers. Since such an oracle npda, in general, cannot implement an internal clock to control its running time, certain oracle answers may lead to an extremely long computation, and thus the machine may recognize even "infeasible" languages. To avoid such a pitfall, we need to demand that, *no matter what oracle is provided*, its underlying oracle npda M must halt on *all* computation paths within $O(n)$ time, where n refers to input size.

Similarly to CFL_m^A and $\mathrm{CFL}_m^{\mathcal{C}}$, we introduce two new notations CFL_T^A and $\mathrm{CFL}_T^{\mathcal{C}}$. An associated deterministic version is denoted $\mathrm{DCFL}_T^{\mathcal{C}}$. A simple relationship between the Turing and many-one CFL-reducibilities is exemplified in Proposition 1. To describe the proposition, we need a notion of the *Boolean hierarchy over CFL*, which was introduced in [13] by setting $\mathrm{CFL}_1 = \mathrm{CFL}$, $\mathrm{CFL}_{2k} = \mathrm{CFL}_{2k-1} \wedge \text{co-CFL}$, and $\mathrm{CFL}_{2k+1} = \mathrm{CFL}_{2k} \vee \mathrm{CFL}$. For simplicity, we denote by BHCFL the union $\bigcup_{k\in\mathbb{N}^+} \mathrm{CFL}_k$. Notice that $\mathrm{CFL} \neq \mathrm{CFL}_2$ holds because $\text{co-CFL} \subseteq \mathrm{CFL}_2$ and $\text{co-CFL} \nsubseteq \mathrm{CFL}$.

Proposition 1. $\mathrm{CFL}_T^{\mathrm{CFL}} = \mathrm{CFL}_m^{\mathrm{CFL}_2} = \mathrm{NFA}_m^{\mathrm{CFL}_2}$.

Proof Sketch. We wish to demonstrate that (1) $\mathrm{CFL}_T^{\mathrm{CFL}} \subseteq \mathrm{CFL}_m^{\mathrm{CFL}_2}$, (2) $\mathrm{CFL}_m^{\mathrm{CFL}_2} \subseteq \mathrm{NFA}_m^{\mathrm{CFL}_2}$, and (3) $\mathrm{NFA}_m^{\mathrm{CFL}_2} \subseteq \mathrm{CFL}_T^{\mathrm{CFL}}$. If all are proven, then the proposition immediately follows. We will show only (1) and (3).

(1) We start with an arbitrary language L in CFL_T^A relative to a certain language A in CFL. Take a T-reduction npda M reducing L to A, and let M_A be an npda recognizing A. Hereafter, we will build a new m-reduction npda N_1 to show that $L \in \mathrm{CFL}_m^{\mathrm{CFL}_2}$. On input x, the machine N_1 tries to simulate M on x by running the following procedure. Along each computation path, before M begins producing the ith query word on a query tape, N_1 guesses its oracle answer b_i (either 0 or 1) and writes it down onto its query tape. While M writes the ith query word y_i, N_1 appends $y_i\natural$ to b_i. When M halts, N_1 produces a query word w of the form $b_1y_1\natural b_2y_2\natural\cdots\natural b_ky_k\natural$, where $k \in \mathbb{N}$. Let L_2 be a collection of those w's such that, for every index $i \in [k]$, if $b_i = 1$ then $y_i \in A$. Similarly, let L_3 be a collection of those w's such that, for every index $i \in [k]$, if $b_i = 0$ then $y_i \in \overline{A}$. It is not difficult to verify that N_1 m-reduces L to $L_2 \cap L_3$.

Next, we want to claim that L_2 and $\overline{L_3}$ are in CFL. This claim leads to a conclusion that L is included in $\mathrm{CFL}_m^{L_2\cap L_3} \subseteq \mathrm{CFL}_m(\mathrm{CFL}\wedge\text{co-CFL}) = \mathrm{CFL}_m^{\mathrm{CFL}_2}$. Obviously, L_2 is in CFL. To see that $\overline{L_3} \in \mathrm{CFL}$, let $w = b_1y_1\natural b_2y_2\natural\cdots\natural b_ky_k\natural$. If $w \in \overline{L_3}$, then there exists an index $i \in [k]$ such that $b_i = 0$ and $y_i \in A$. This last property can be checked by running M_A sequentially on each y_i and emptying its stack after each run of M_A. Thus, $\overline{L_3}$ is in CFL.

(3) Choose an oracle A in CFL_2 and consider an arbitrary language L in CFL_m^A. Furthermore, take two languages $A_1, A_2 \in \mathrm{CFL}$ for which $A = A_1 \cap \overline{A}_2$. Let M be an oracle nfa that recognizes L relative to A. Notice that M has no stack. We will define another oracle npda N as follows. On input x, N first marks 0 on its query tape and start simulating M on x. Whenever M tries to write a symbol σ on its query tape, N writes it down on a query tape and simultaneously copies it into a stack. After M halts with a query word, say, w, N makes the first query with the query word $0w$. If its oracle answer is 0, then N rejects the input. Subsequently, N writes 1 on the query tape (provided that the tape automatically becomes blank), pops the stored string w^R from the stack, and copies it to the query tape. After making the second query with $1w^R$, if its oracle answer equals 1, then N rejects the input. When N has not entered any rejecting state, then N finally accepts the input. The corresponding oracle B is defined as $\{0w \mid w \in A_1\} \cup \{1w^R \mid w \in A_2\}$. It is easy to see that $x \in L$ if

and only if N accepts x relative to B. Since CFL is known to be closed under reversal, $\{1w^R \mid w \in A_2\}$ is context-free, and thus B is a member of CFL. We then conclude that $L \in \mathrm{CFL}_T^B \subseteq \mathrm{CFL}_T^{\mathrm{CFL}}$. $\square$

4 The CFL Hierarchy

4.1 Reducibility and a Hierarchy

Applying Turing CFL-reductions to CFL level by level, we can build a useful hierarchy, called the *CFL hierarchy*, whose kth level consists of three language families Δ_k^{CFL}, Σ_k^{CFL}, and Π_k^{CFL}. To be more precise, for each level $k \geq 1$, we set $\Delta_1^{\mathrm{CFL}} = \mathrm{DCFL}$, $\Sigma_1^{\mathrm{CFL}} = \mathrm{CFL}$, $\Delta_{k+1}^{\mathrm{CFL}} = \mathrm{DCFL}_T(\Sigma_k^{\mathrm{CFL}})$, $\Pi_k^{\mathrm{CFL}} = \text{co-}\Sigma_k^{\mathrm{CFL}}$, and $\Sigma_{k+1}^{\mathrm{CFL}} = \mathrm{CFL}_T(\Sigma_k^{\mathrm{CFL}})$. Additionally, we set $\mathrm{CFLH} = \bigcup_{k\in\mathbb{N}^+} \Sigma_k^{\mathrm{CFL}}$. The CFL hierarchy can be used to categorize the complexity of typical non-context-free languages discussed in most introductory textbooks. We will review a few typical examples that fall into the CFL hierarchy.

Example 3. In Example 1, we have seen the languages $Dup_2 = \{xx \mid x \in \{0,1\}^*\}$ and $Dup_3 = \{xxx \mid x \in \{0,1\}\}$, which are both in $\mathrm{CFL}_m^{\mathrm{CFL}}$. Note that, since $\mathrm{CFL}_m^A \subseteq \mathrm{CFL}_T^A$ for any oracle A, every language in $\mathrm{CFL}_m^{\mathrm{CFL}}$ belongs to $\mathrm{CFL}_T^{\mathrm{CFL}} = \Sigma_2^{\mathrm{CFL}}$. Therefore, Dup_2 and Dup_3 are in Σ_2^{CFL}. In addition, as shown in Example 2, the language $Sq = \{0^n1^{n^2} \mid n \geq 1\}$ is in $\mathrm{CFL}_m^{\mathrm{CFL}}$ while $Prim = \{0^n \mid n \text{ is a prime number }\}$ is in co-$(\mathrm{CFL}_m^{\mathrm{CFL}})$. Therefore, we conclude that Sq is in Σ_2^{CFL} and $Prim$ is in Π_2^{CFL}. A similar but more involved example is the language $MulPrim = \{0^{mn} \mid m \text{ and } n \text{ are prime numbers}\}$. It is possible to show that $MulPrim$ belongs to $\mathrm{CFL}_m(\text{co-}(\mathrm{CFL}_m^{\text{co-CFL}}))$, which equals Σ_3^{CFL}.

Lemma 2. *Let k be any integer satisfying $k \geq 1$.*

1. $\mathrm{CFL}_T(\Sigma_k^{\mathrm{CFL}}) = \mathrm{CFL}_T(\Pi_k^{\mathrm{CFL}})$ *and* $\mathrm{DCFL}_T(\Sigma_k^{\mathrm{CFL}}) = \mathrm{DCFL}_T(\Pi_k^{\mathrm{CFL}})$.
2. $\Sigma_k^{\mathrm{CFL}} \cup \Pi_k^{\mathrm{CFL}} \subseteq \Delta_{k+1}^{\mathrm{CFL}} \subseteq \Sigma_{k+1}^{\mathrm{CFL}} \cap \Pi_{k+1}^{\mathrm{CFL}}$.
3. $\mathrm{CFLH} \subseteq \mathrm{DSPACE}(O(n))$.

Hereafter, we will explore fundamental properties of our new hierarchy. Our starting point is a closure property under length-nondecreasing substitution, where a substitution $s : \Sigma \to \mathcal{P}(\Theta^*)$ is called *length nondecreasing* if $s(\sigma) \neq \emptyset$ and $\lambda \notin s(\sigma)$ for every symbol $\sigma \in \Sigma$. We expand s as follows. Define $s(\sigma_1\sigma_2\cdots\sigma_n) = \{x_1x_2\cdots x_n \mid \forall i \in [n](x_i \in s(\sigma_i))\}$ for $\sigma_1, \sigma_2, \ldots, \sigma_n \in \Sigma$ and let $s(L) = \bigcup_{x\in L} s(x)$ for language $L \subseteq \Sigma^*$. A homomorphism $h : \Sigma \to \Theta^*$ is called *λ-free* if $h(\sigma) \neq \lambda$ for every $\sigma \in \Sigma$. Note that the condition of length nondecreasing is necessary because every recursively enumerable language can be a homomorphic image of a certain language in CFL_2 ($\subseteq \Sigma_2^{\mathrm{CFL}}$) [3].

Lemma 3. *1. (substitution property) Let $k \in \mathbb{N}^+$ and let s be any length-nondecreasing substitution on alphabet Σ satisfying $s(\sigma) \in \Sigma_k^{\mathrm{CFL}}$ for each symbol $\sigma \in \Sigma$. For any language A over Σ, if L is in Σ_k^{CFL}, then $s(L)$ is also in Σ_k^{CFL}.*

2. *For each index $k \in \mathbb{N}^+$, the family Σ_k^{CFL} is closed under the following operations: concatenation, union, reversal, Kleene closure, λ-free homomorphism, and inverse homomorphism.*

We will show that the second level of the CFL hierarchy contains BHCFL.

Proposition 2. BHCFL $\subseteq \Sigma_2^{\mathrm{CFL}} \cap \Pi_2^{\mathrm{CFL}}$.

Proof Sketch. We will show that BHCFL $\subseteq \Sigma_2^{\mathrm{CFL}}$. Obviously, $\mathrm{CFL}_1 \subseteq \Sigma_2^{\mathrm{CFL}}$ holds. It is therefore enough to show that $\mathrm{CFL}_k \subseteq \Sigma_2^{\mathrm{CFL}}$ for every index $k \geq 2$.

We first claim that, for every index $k \geq 1$, $\mathrm{CFL}_{2k} = \bigvee_{i\in[k]} \mathrm{CFL}_2$ ($= \mathrm{CFL}_2 \vee \mathrm{CFL}_2 \vee \cdots \vee \mathrm{CFL}_2$ with k repetitions of CFL_2) and $\mathrm{CFL}_{2k+1} = (\bigvee_{i\in[k]} \mathrm{CFL}_2) \vee \mathrm{CFL}$. This can be shown using an idea of [13, Claim 4]. Next, we claim that $\mathrm{CFL}_{2k}, \mathrm{CFL}_{2k+1} \subseteq \Sigma_2^{\mathrm{CFL}}$ for all indices $k \geq 1$. The proof of this claim proceeds by induction on $k \geq 1$. Furthermore, we will prove that BHCFL $\subseteq \Pi_2^{\mathrm{CFL}}$. It is possible to prove by induction on $k \in \mathbb{N}^+$ that co-$\mathrm{CFL}_k \subseteq \mathrm{CFL}_{k+1}$. From this inclusion, we obtain co-BHCFL $\subseteq$ BHCFL. By symmetry, BHCFL $\subseteq$ co-BHCFL holds. Thus, we conclude that BHCFL = co-BHCFL. □

Let us turn our attention to CFL(ω). A direct analysis of each language family CFL(k) shows that CFL(ω) is included in BHCFL.

Proposition 3. *1.* CFL(ω) $\subseteq$ BHCFL *(thus,* CFL(ω) $\subseteq \Sigma_2^{\mathrm{CFL}} \cap \Pi_2^{\mathrm{CFL}}$*).*
2. $\mathrm{CFL}_m^{\mathrm{CFL}(\omega)} \subseteq \Sigma_3^{\mathrm{CFL}}$.

Proof Sketch. A key to the proof of the first part of this proposition is the following claim: for every index $k \geq 1$, $\mathrm{CFL}(k) \subseteq \mathrm{CFL}_{2k+1}$ holds. The first part then implies that $\mathrm{CFL}_m^{\mathrm{CFL}(\omega)}$ is included in $\mathrm{CFL}_m^{\mathrm{BHCFL}}$. Since BHCFL $\subseteq \Sigma_2^{\mathrm{CFL}} \cap \Pi_2^{\mathrm{CFL}}$ by Proposition 2, it follows that $\mathrm{CFL}_m^{\mathrm{BHCFL}}$ is included in $\mathrm{CFL}_m(\Pi_2^{\mathrm{CFL}})$, which is obviously a subclass of $\mathrm{CFL}_T(\Pi_2^{\mathrm{CFL}}) = \Sigma_3^{\mathrm{CFL}}$. □

4.2 Structural Properties

We will further explore structural properties that characterize the CFL hierarchy. Moreover, we will present three alternative characterizations (Theorem 2 and Proposition 4) of the hierarchy. Let us consider a situation in which Boolean operations are applied to languages in the CFL hierarchy. In the following lemma, the third statement needs an extra attention. As we have seen, it holds that CFL $\wedge$ CFL = CFL(2) $\neq$ CFL. Therefore, the equality $\Sigma_k^{\mathrm{CFL}} \wedge \Sigma_k^{\mathrm{CFL}} = \Sigma_k^{\mathrm{CFL}}$ does not hold in the first level (i.e., $k = 1$). Surprisingly, it is possible to prove that this equality actually holds for any level *more than* 1.

Lemma 4. *Let $k \geq 1$.*

1. $\Sigma_k^{\mathrm{CFL}} \vee \Sigma_k^{\mathrm{CFL}} = \Sigma_k^{\mathrm{CFL}}$ *and* $\Pi_k^{\mathrm{CFL}} \wedge \Pi_k^{\mathrm{CFL}} = \Pi_k^{\mathrm{CFL}}$.
2. $\Sigma_k^{\mathrm{CFL}} \wedge \Pi_k^{\mathrm{CFL}} \subseteq \Sigma_{k+1}^{\mathrm{CFL}} \cap \Pi_{k+1}^{\mathrm{CFL}}$ *and* $\Sigma_k^{\mathrm{CFL}} \vee \Pi_k^{\mathrm{CFL}} \subseteq \Sigma_{k+1}^{\mathrm{CFL}} \cap \Pi_{k+1}^{\mathrm{CFL}}$.
3. $\Sigma_k^{\mathrm{CFL}} \wedge \Sigma_k^{\mathrm{CFL}} = \Sigma_k^{\mathrm{CFL}}$ *and* $\Pi_k^{\mathrm{CFL}} \vee \Pi_k^{\mathrm{CFL}} = \Pi_k^{\mathrm{CFL}}$ *for all levels $k \geq 2$.*

Lemma 4(3) is not quite trivial and its proof follows from Theorem 2, in which we give two new characterizations of Σ_k^{CFL} in terms of many-one reducibilities. For our purpose, we introduce two extra many-one hierarchies. The *many-one CFL hierarchy* consists of language families $\Sigma_{m,k}^{\mathrm{CFL}}$ and $\Pi_{m,k}^{\mathrm{CFL}}$ ($k \in \mathbb{N}^+$) defined as follows: $\Sigma_{m,1}^{\mathrm{CFL}} = \mathrm{CFL}$, $\Pi_{m,k}^{\mathrm{CFL}} = \text{co-}\Sigma_{m,k}^{\mathrm{CFL}}$, and $\Sigma_{m,k+1}^{\mathrm{CFL}} = \mathrm{CFL}_m(\Pi_{m,k}^{\mathrm{CFL}})$ for any $k \geq 1$, where the subscript "m" stands for "many-one." A *relativized many-one NFA hierarchy*, which was essentially formulated in [8], is defined as follows relative to oracle A: $\Sigma_{m,1}^{\mathrm{NFA},A} = \mathrm{NFA}_m^A$, $\Pi_{m,k}^{\mathrm{NFA},A} = \text{co-}\Sigma_{m,k}^{\mathrm{NFA},A}$, and $\Sigma_{m,k+1}^{\mathrm{NFA},A} = \mathrm{NFA}_m(\Pi_{m,k}^{\mathrm{NFA},A})$ for every index $k \geq 1$. Given a language family $\mathcal{C}$, $\Sigma_{m,k}^{\mathrm{NFA},\mathcal{C}}$ (or $\Sigma_{m,k}^{\mathrm{NFA}}(\mathcal{C})$) denotes the union $\bigcup_{A\in\mathcal{C}} \Sigma_{m,k}^{\mathrm{NFA},A}$.

Theorem 2. *$\Sigma_k^{\mathrm{CFL}} = \Sigma_{m,k}^{\mathrm{CFL}} = \Sigma_{m,k}^{\mathrm{NFA}}(DYCK)$ for every index $k \geq 1$.*

Proof Sketch. The first step toward the proof is to prove two key claims. (1) For every index $k \geq 1$, it holds that $\Sigma_{k+1}^{\mathrm{CFL}} \subseteq \mathrm{CFL}_m(\Sigma_k^{\mathrm{CFL}} \wedge \Pi_k^{\mathrm{CFL}}) \subseteq \mathrm{NFA}_m(\Sigma_k^{\mathrm{CFL}} \wedge \Pi_k^{\mathrm{CFL}})$. (2) For any two indices $k \geq 1$ and $e \geq k-1$, it holds that $\mathrm{NFA}_m(\Sigma_{m,k}^{\mathrm{CFL}} \wedge \Pi_{m,e}^{\mathrm{CFL}}) \subseteq \mathrm{CFL}_m(\Pi_{m,e}^{\mathrm{CFL}})$.

In the second step, we use induction on $k \geq 1$ to prove the theorem. Since Lemma 1 handles the base case $k = 1$, it is sufficient to assume that $k \geq 2$. The second equality of the theorem is shown as follows. If $k = 1$, then the claim is exactly the same as $\mathrm{CFL} = \mathrm{NFA}_m^{DYCK}$. In the case of $k \geq 2$, assume that $L \in \mathrm{CFL}_m^A$ for a certain language A in $\Pi_{m,k-1}^{\mathrm{CFL}}$. A proof similar to that of $\mathrm{CFL} = \mathrm{NFA}_m^{DYCK}$ demonstrates the existence of a certain Dyck language D satisfying that $\mathrm{CFL}_m^A = \mathrm{NFA}_m^B$, where B is of the form $\{[\,\frac{\tilde{y}}{\tilde{z}}\,] \mid y \in D, z \in A\}$ and $\tilde{y}$ and $\tilde{z}$ are $\natural$-extensions of y and z, respectively. The definition places B into the language family $\mathrm{DCFL} \wedge \Pi_{m,k-1}^{\mathrm{CFL}}$, which equals $\Pi_{m,k-1}^{\mathrm{CFL}}$ because of $k \geq 2$. By our induction hypothesis, $\Pi_{m,k-1}^{\mathrm{CFL}} = \Pi_{m,k-1}^{\mathrm{NFA}}(DYCK)$ holds. It thus follows that $\mathrm{NFA}_m^B \subseteq \mathrm{NFA}_m(\Pi_{m,k-1}^{\mathrm{NFA}}(DYCK)) = \Sigma_{m,k}^{\mathrm{NFA}}(DYCK)$, and therefore we obtain $L \in \mathrm{CFL}_m^A \subseteq \mathrm{NFA}_m^A \subseteq \Sigma_{m,k}^{\mathrm{NFA}}(DYCK)$.

Next, we will establish the first equality given in the theorem. Clearly, $\Sigma_{m,k}^{\mathrm{CFL}} \subseteq \Sigma_k^{\mathrm{CFL}}$ holds since $\mathrm{CFL}_m^A \subseteq \mathrm{CFL}_T^A$ for any oracle A. Now, we target the opposite containment. By (1), it follows that $\Sigma_k^{\mathrm{CFL}} \subseteq \mathrm{NFA}_m(\Sigma_{k-1}^{\mathrm{CFL}} \wedge \Pi_{k-1}^{\mathrm{CFL}})$. Since $\Sigma_{k-1}^{\mathrm{CFL}} = \Sigma_{m,k-1}^{\mathrm{CFL}}$, we obtain $\Sigma_k^{\mathrm{CFL}} \subseteq \mathrm{NFA}_m(\Sigma_{m,k-1}^{\mathrm{CFL}} \wedge \Pi_{m,k-1}^{\mathrm{CFL}})$. Note that (2) implies the inclusion $\mathrm{NFA}_m(\Sigma_{m,k-1}^{\mathrm{CFL}} \wedge \Pi_{m,k-1}^{\mathrm{CFL}}) \subseteq \mathrm{CFL}_m(\Pi_{m,k-1}^{\mathrm{CFL}}) = \Sigma_{m,k}^{\mathrm{CFL}}$. In conclusion, $\Sigma_k^{\mathrm{CFL}} \subseteq \Sigma_{m,k}^{\mathrm{CFL}}$ holds. □

An *upward collapse property* holds for the CFL hierarchy except for the first level. Similar to the notation CFL_e expressing the eth level of the Boolean hierarchy over CFL, a new notation $\Sigma_{k,e}^{\mathrm{CFL}}$ is introduced to denote the eth level of the Boolean hierarchy over Σ_k^{CFL}. Additionally, we set $\mathrm{BH}\Sigma_k^{\mathrm{CFL}} = \bigcup_{e\in\mathbb{N}^+} \Sigma_{k,e}^{\mathrm{CFL}}$.

Lemma 5. *(upward collapse properties) Let k be any integer at least 2.*

1. $\Sigma_k^{\mathrm{CFL}} = \Sigma_{k+1}^{\mathrm{CFL}}$ *iff* $\mathrm{CFLH} = \Sigma_k^{\mathrm{CFL}}$.
2. $\Sigma_k^{\mathrm{CFL}} = \Pi_k^{\mathrm{CFL}}$ *iff* $\mathrm{BH}\Sigma_k^{\mathrm{CFL}} = \Sigma_k^{\mathrm{CFL}}$.

3. $\Sigma_k^{\mathrm{CFL}} = \Pi_k^{\mathrm{CFL}}$ *implies* $\Sigma_k^{\mathrm{CFL}} = \Sigma_{k+1}^{\mathrm{CFL}}$.

From Lemma 5, if the Boolean hierarchy over Σ_k^{CFL} collapses to Σ_k^{CFL}, then the entire CFL hierarchy collapses. It is not clear, however, that a much weaker assumption like $\Sigma_{k,e}^{\mathrm{CFL}} = \Sigma_{k,e+1}^{\mathrm{CFL}}$ suffices to draw the collapse of the CFL hierarchy (for instance, $\Sigma_{k+1}^{\mathrm{CFL}} = \Sigma_{k+2}^{\mathrm{CFL}}$).

Theorem 2 also gives a logical characterization of Σ_k^{CFL}. For convenience, we define a function Ext as $Ext(\tilde{x}) = x$ for any $\natural$-extension $\tilde{x}$ of string x.

Proposition 4. *Let $k \geq 1$. For any language $L \in \Sigma_k^{\mathrm{CFL}}$ over alphabet Σ, there exists another language $A \in$ DCFL and a linear polynomial p with $p(n) \geq n$ for all $n \in \mathbb{N}$ that satisfy the following equivalence relation: for any number $n \in \mathbb{N}$ and any string $x \in \Sigma^n$, $x \in L$ if and only if*

$$\exists \tilde{x}(|\tilde{x}| \leq p(n))\, \exists y_1(|y_1| \leq p(n))\, \forall y_2(|y_2| \leq p(n)) \cdots Q_k y_k(|y_k| \leq p(n))\, [x = Ext(\tilde{x}) \wedge [\tilde{x}, y_1, y_2, \ldots, y_k]^T \in A],$$

where Q_k is $\exists$ ($\forall$, resp.) if k is odd (even, resp.) and $\tilde{x}$ is a $\natural$-extension of x.

Recall that the first and second levels of the CFL hierarchy are different. It is possible to prove that the rest of the hierarchy is infinite unless the polynomial hierarchy over NP collapses.

References

1. Berstel, J.: Transductions and Context-Free Languages. B. G. Teubner, Stuttgart (1979)
2. Du, D., Ko., K.: Theory of Computational Complexity. John Willey & Sons (2000)
3. Ginsburg, S., Greibach, S.A., Harrison, M.A.: One-way stack languages. J. ACM 14, 389–418 (1967)
4. Greibach, S.A.: The hardest context-free language. SIAM J. Comput. 2, 304–310 (1973)
5. Hromkovič, J., Schnitger, G.: On probabilistic pushdown automata. Inf. Comput. 208, 982–995 (2010)
6. Ladner, R., Lynch, N., Selman, A.: A comparison of polynomial-time reducibilities. Theor. Comput. Sci. 1, 103–123 (1975)
7. Liu, L.Y., Weiner, P.: An infinite hierarchy of intersections of context-free languages. Math. Systems Theory 7, 185–192 (1973)
8. Reinhardt, K.: Hierarchies over the context-free languages. In: Dassow, J., Kelemen, J. (eds.) IMYCS 1990. LNCS, vol. 464, pp. 214–224. Springer, Heidelberg (1990)
9. Tadaki, K., Yamakami, T., Lin, J.C.H.: Theory of one-tape linear-time Turing machines. Theor. Comput. Sci. 411, 22–43 (2010)
10. Yamakami, T.: Swapping lemmas for regular and context-free languages. Available at arXiv:0808.4122 (2008)
11. 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)
12. Yamakami, T.: Immunity and pseudorandomness of context-free languages. Theor. Comput. Sci. 412, 6432–6450 (2011)
13. Yamakami, T., Kato, Y.: The dissecting power of regular languages. Inf. Pross. Lett. 113, 116–122 (2013)
14. Younger, D.H.: Recognition and parsing of context-free languages in time n^3. Inf. Control 10, 189–208 (1967)