# Enumeration of Context-Free Languages and Related Structures

Michael Domaratzki*
Jodrey School of Computer Science,
Acadia University
Wolfville, NS B4P 2R6 Canada
mike.domaratzki@acadiau.ca

Alexander Okhotin†
Department of Mathematics,
University of Turku
FIN–20014 Turku, Finland
okhotin@cs.utu.fi

Jeffrey Shallit
School of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1 Canada
shallit@graceland.uwaterloo.ca

**Abstract**

In this paper, we consider the enumeration of context-free languages. In particular, for any reasonable descriptional complexity measure for context-free grammars, we demonstrate that the exact quantity of context-free languages of size $n$ is uncomputable. Nevertheless, we are able to give upper and lower bounds on the number of such languages. We also generalize our results to enumerate predicates that are recursively enumerable or co-recursively enumerable.

## 1 Introduction

Enumeration is the study of the number of distinct objects from an infinite set that are of a fixed, finite size. Enumeration has a long history of interaction with formal language theory, dating to the 1950s, where researchers were primarily interested in the enumeration of deterministic finite automata (DFAs). We refer the reader to Domaratzki *et al.* [2] for a bibliography of research on enumeration of finite automata. Recently, enumeration of other objects that characterize regular languages—particularly non-deterministic finite automata (NFAs) [2] and regular expressions [12]—has also been examined. These problems are often complicated by the fact that it is difficult to enumerate the number of NFAs, DFAs and regular expressions that generate distinct languages. Experimental results can also be obtained by exhaustively computing the number of objects generating distinct regular languages. However, we note that in each of these cases, the equivalence of two objects (DFAs, NFAs or regular expressions) is computable, a basic necessity in algorithms for explicitly computing these quantities.

In this paper, we examine the problem of enumerating context-free languages (CFLs). It is well known that given two context-free grammars (CFGs), it is undecidable whether they generate

*Research supported in part by NSERC.
†Supported by the Academy of Finland under grant 206039.

the same language. This eliminates the natural method for computing experimental results for the number of CFLs of a given size. However, it does not immediately preclude that enumerating the number of CFLs of size $n$ is computable given $n$. To our knowledge, despite the simplicity of this problem, it has not been studied before. We note, however, that several authors have considered the unrelated problem of, for a fixed CFG $G$, enumerating (exactly or approximately) the number of, or listing the, words of length $n$ generated by $G$. For example, see Dömösi [4], Bertoni *et al.* [1], Gore *et al.* [6] and Martinez [13].

We show that the enumeration of CFLs is uncomputable in general. Thus, it is impossible to compute the exact values algorithmically. However, we also give asymptotic bounds on the number of CFLs of size $n$. We examine this problem with respect to different notions of the size of a CFG, including number of symbols and number of variables in CNF.

## 2 Preliminary Definitions

Let $\Sigma$ be a finite set of symbols, called *letters*. Then $\Sigma^*$ is the set of all finite sequences of letters from $\Sigma$, which are called *words*. The empty word $\varepsilon$ is the empty sequence of letters. The *length* of a word $w = w_1 w_2 \cdots w_n \in \Sigma^*$, where $w_i \in \Sigma$, is $n$, and is denoted $|w|$. For any $w \in \Sigma^*$ and $a \in \Sigma$, we denote by $|w|_a$ the number of occurrences of $a$ in $w$. A *language* $L$ is any subset of $\Sigma^*$. For additional background in formal languages and automata theory, please see Yu [14] or Hopcroft and Ullman [11]. In particular, we assume that the reader is familiar with the concepts of DFAs and NFAs.

A context-free grammar (CFG) is a 4-tuple $G = (V, \Sigma, P, S)$ where $\Sigma$ is an alphabet, $V$ is a finite set of *variables* (or *non-terminals*), $S \in V$ is the unique start variable and $P$ is the set of productions. Each production in $P$ has the form $A \to \alpha$ where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$. Let $\alpha, \beta \in (V \cup \Sigma)^*$. We denote by $\alpha \Rightarrow \beta$ the fact that there exists $A \to \gamma$ in $P$ with $\alpha = \alpha_1 A \alpha_2$ and $\beta = \alpha_1 \gamma \alpha_2$ for some $\alpha_1, \alpha_2 \in (V \cup \Sigma)^*$. Let $\Rightarrow^*$ denote the reflexive, transitive closure of $\Rightarrow$. The language generated by a CFG $G$ is denoted by

$$L(G) = \{w \in \Sigma^* \ : \ S \Rightarrow^* w\}.$$

A language $L$ is a context-free language (CFL) if there exists a CFG $G$ such that $L = L(G)$.

In this paper, $\lg$ denotes logarithm to the base 2.

## 3 Descriptional Complexity of Context-Free Languages

Our first task is to consider appropriate descriptional complexity measures for context-free grammars and context-free languages. Unlike the case of deterministic finite automata and regular languages—where *state complexity* is a widely accepted descriptional measure—there is no agreed upon complexity measure for context-free languages. For instance, Gruska [9, 10] defines the following descriptional complexity measures for a CFG $G = (V, \Sigma, P, S)$:

(a) The number of variables $|V|$, denoted $\text{var}(G)$;

(b) The number of grammatical levels of $G$, denoted $\text{lev}(G)$. A *level* of $G$ is a subset $P_0 \subseteq P$ such that $A \to \alpha \in P_0$ implies

$$B \to \beta \in P_0 \iff A \Rightarrow^* \gamma_1 B \gamma_2 \text{ and } B \Rightarrow^* \eta_1 A \eta_2 \text{ for some } \gamma_1, \gamma_2, \eta_1, \eta_2 \in (V \cup \Sigma)^*$$

(c) The depth of $G$, denoted depth$(G)$. The depth of $G$ is the maximum depth of any grammatical level: depth$(G) = \max\{depth(P_0) \ : \ P_0$ is a level of $G\}$. The depth of a grammatical level is given by

$$\text{depth}(P_0) = |\{A \ : \ \exists \alpha \text{ such that } A \to \alpha \in P_0\}|.$$

Other measures examined in the literature are

(d) The number of productions in $G$.

(e) The sum of the lengths of the productions:

$$\sum_{A \to \alpha \in P} |\alpha| + 2$$

This measure is known as the *number of symbols* in $G$ [5]. It is often denoted symb$(G)$.

For each of these measures, the complexity of a CFL $L$ is the minimum complexity over all CFGs $G$ with $L(G) = L$. We do not discuss measures (b), (c) or (d) further in this paper.

We discuss measure (a), the total number of variables, in some detail. In general, this is not an accurate measure of the total size of a CFG if, for example, we are encoding and storing the CFG. This is because the lengths of the right-hand sides of the productions are not bounded in any way. Further, it suffers from the problem that for a fixed number $n$, there are infinitely many CFLs generated by CFGs with $n$ variables (in fact, any finite language can be generated by a CFG with a single non-terminal). We would like to prevent any descriptional complexity measure we are dealing with from possessing the property that there are infinitely many distinct languages of a fixed size. Let $d$ be a descriptional complexity measure on CFGs over a fixed alphabet. Let us further assume that all variable names are chosen from a fixed countable set. Then we say that $d$ is *well-behaved* if

$$\forall n \geq 0 \ |\{G \ : \ d(G) \leq n\}| < \infty.$$

We note that measure (a) is not well-behaved.

However, there are also some compelling reasons for examining measure (a) as a descriptional complexity measure for CFGs. The most obvious is the link to the state complexity of regular languages. In the traditional conversion of a DFA to a (right-linear) CFG, a DFA with state complexity $n$ is converted to a grammar with $n$ variables. With this in mind, we can salvage the measure of "number of variables" as a descriptional complexity measure by insisting that our grammars be in Chomsky normal form (CNF). (A grammar is in CNF if all productions are of the form $A \to BC$ or $A \to a$, where $A, B, C$ are variables and $a$ is a terminal.) If $L$ is a CFL, we define

$$\text{vcnf}(L) = \min\{|V| \ : \ G = (V, \Sigma, P, S) \text{ is a CFG in CNF accepting } L\}.$$

This measure has been employed by Domaratzki *et al.*[3] for measuring the descriptional complexity tradeoffs between regular and context-free languages. We adopt this descriptional complexity measure in Section 5.2 for enumerating unary CFLs.

## 4 Uncomputability of Enumeration of CFLs

We begin with a proof that it is impossible to exactly compute the number of CFLs of a given size. We note that the undecidability result does not depend on the descriptional complexity measure, as long as it is itself computable and well-defined.

**Theorem 1.** *Let $f(n)$ denote the number of distinct CFLs of size $n$ with respect to a given, computable well-behaved descriptional complexity measure. Then the function $f$ is uncomputable.*

*Proof.* Let us denote our well-behaved, computable descriptional complexity measure by $d$. Let $G', G''$ be CFGs. Without loss of generality, we can assume that $d(G') = d(G'')$, since we can always add useless non-terminals or productions to pad the smaller grammar. Suppose that $f(n)$ can be effectively computed for any given $n$, and consider the following algorithm:

Input: $G', G''$, such that $d(G') = d(G'') = n$
Let $\mathcal{G}_n$ be the set of all CFGs of measure $n$.
Data structure: $\mathcal{C}$, a partition of $\mathcal{G}_n$.
1. Let $\mathcal{C} = \big\{\{\mathcal{G}_n\}\big\}$.
2. Compute $m = f(n)$.
3. For all $w \in \Sigma^*$
4. For each group $C \in \mathcal{C}$
5. Determine whether $w \in L(G_i)$ for all $G_i \in C$.
6. If $\exists i, j$: $w \in L(G_i)$ and $w \notin L(G_j)$
7. Split $C$ into
$C_0 = \{G \in C \mid w \notin L(G)\}$,
$C_1 = \{G \in C \mid w \in L(G)\}$
8. If $|\mathcal{C}| = m$
9. Break from *for* loop
10. Accept if $G'$ and $G''$ are in the same class in $\mathcal{C}$, reject otherwise.

The general strategy is to create all $m$ equivalence classes of context-free languages, based on being able to compute $m = f(n)$. Then, once these classes have been created, we test if two grammars are equivalent by testing if they are in the same class. Since this is undecidable, computing $f(n)$ must also be undecidable.

To see that the algorithm works, consider that the last line (10) of the algorithm is eventually reached: since there are $m = f(n)$ distinct CFLs of size $n$, by testing each grammar on words of successive lengths, eventually all $m$ equivalence classes will be separated from each other.

Listing $\mathcal{G}_n$ is trivial: we can simply list all possibilities for a CFG with size $n$ exhaustively. Line (5) can be implemented by any general CFL parsing strategy: CYK or Earley's Algorithm, for example. Line (10) can be tested since we only need to test for grammar equality, not equivalence, in order to conclude that $G'$ and $G''$ are or are not in the same class. □

We can also consider the undecidability of other enumerative problems relating to context-free grammars.

**Theorem 2.** *Let $f(n)$ denote the number of CFGs of size $n$ (with respect to a well-behaved, computable descriptional complexity measure) that generate $\Sigma^*$. Then $f(n)$ is not computable.*

*Proof.* Supposing the computability of $f$, let us construct an algorithm for solving the universality problem for context-free grammars ("determine, whether a given $G$ over $\Sigma$ generates $\Sigma^*$"), which is known to be undecidable, or, to be precise, $\Pi_1$-complete.

Input: $G$, such that $d(G) = n$
Let $\mathcal{G}_n = \{\mathcal{G}_1, \ldots \mathcal{G}_N\}$ be all CFGs $G'$ with $d(G') = n$.
Data structure: $\mathcal{C}$, a subset of $\mathcal{G}_n$.

1. Let $\mathcal{C} = \mathcal{G}_n$
2. Compute $m = f(n)$.
3. For all $w \in \Sigma^*$
4. For all $G' \in \mathcal{C}$
5. If $w \notin L(G)$
6. Remove $G'$ from $\mathcal{C}$
7. if $|\mathcal{C}| = m$
8. Accept if $G \in \mathcal{C}$, reject otherwise.

The basic idea is the same as in Theorem 1, and it is more obvious in this case. There are $m = f(n)$ context-free grammars of size $n$ that generate $\Sigma^*$, while each of the rest of the grammars cannot generate some string $w \in \Sigma^*$. All these strings will eventually be found, and all grammars that generate languages other than $\Sigma^*$ will be removed from $\mathcal{C}$. Then lines (7–8) report the result. □

We note that, in contrast, the exact number of distinct CFGs of size $n$ that generate $\varnothing$ or a finite language is computable.

# 5 Enumerating CFLs

We now turn to the growth of some enumerative functions related to CFLs. Due to the results in the previous section, these functions are uncomputable. However, we are able to give estimates of their growth.

## 5.1 Enumerating CFLs by Number of Symbols

We first consider estimating the number of CFLs with $n$ symbols (i.e., the total sum of production lengths is $n$) over a $k$ letter alphabet. Let $\lambda_k(n)$ denote the number of CFLs generated by CFGs with $n$ symbols over a $k$-letter alphabet. We begin with an upper bound:

**Theorem 3.** *For all $n, k \geq 1$, the following inequality holds:*

$$\lg(\lambda_k(n)) \leq (n-1)\lg(n/2 + k + 1).$$

*Proof.* For any word $w = w_1 \# w_2 \# \cdots \# w_m \#$ over the alphabet $\Sigma \cup V \cup \{\#\}$, $w$ defines a CFG-like form

$$\begin{array}{ccc} \alpha_1 & \rightarrow & w'_1 \\ \alpha_2 & \rightarrow & w'_2 \\ \vdots & \vdots & \vdots \\ \alpha_n & \rightarrow & w'_m \end{array}$$

where $w_i = \alpha_i w_i'$ and $\alpha_i \in V \cup \Sigma$. Note that if $\alpha_i \notin V$, then this does not define a valid CFG. However, this will suffice for our upper bound.

Therefore, we see that for all words $w \in (|V| + k + 1)^{n-1}$, we get at most one valid grammar with $n$ symbols. If $w$ is of length $n-1$, there can be at most $n/2$ productions (at most every other symbol of $w$ can be #). Thus, we get $|V| \leq n/2$ and our upper bound is $(n/2 + k + 1)^{n-1}$. □

We now give a lower bound. We require the following result due to Domaratzki *et al.* [2]:

**Theorem 4.** *The number of distinct regular languages generated by a DFA with $n$ states over a $k \geq 2$ letter alphabet is at least $n2^n n^{(k-1)n}$.*

**Theorem 5.** *For all $n \geq 1$ and $k \geq 2$, we have*

$$\lg(\lambda_k(n)) \geq \frac{k-1}{4k+2} n \lg n + \lg n - \frac{k-1}{4k+2} n \lg(4k+2) - \lg(4k+2) + \frac{n}{4k+2}$$

*Proof.* We consider the impact of converting a DFA with $n$ states over a $k$-letter alphabet to a CFG. Let $M = (Q, \Sigma, \delta, q_0, F)$. Define a CFG $G = (Q, \Sigma, P, q_0)$. For each $\delta(q_i, a_j) = q_\ell$ we define the production $q_i \to a_j q_\ell$. Thus, for each of the $nk$ transitions in $M$, we have a production of length $4$. Furthermore, for each final state $q_f \in F$, we have a production $q_f \to \varepsilon$, of length $2$. Thus, the total number of symbols in $G$ is $4nk + 2|F| \leq (4k+2)n$. Using Theorem 4, we obtain our result. □

## 5.2 Enumerating CFLs by Number of variables

We now discuss enumeration of CFLs by the number of variables in a CNF CFG generating the language. We require the following theorem due to Domaratzki *et al.* [3]:

**Theorem 6.** *Let $n \geq 1$ and let $L$ be any subset of $\{\varepsilon, a, \ldots, a^{n-1}\}$. Then there exists a CFG $G$ in CNF with at most $4\lceil n^{1/3} \rceil + 2$ variables generating $L$.*

Let $F_k(n)$ give the number of distinct CFLs generated by a CNF CFG with at most $n$ variables over a $k$-letter alphabet. In the following theorem, $k$ is considered a constant.

**Theorem 7.** $F_k(n) \in 2^{\Theta(n^3)}$.

*Proof.* By Theorem 6, for some $f(n) \in \Omega(n)$, each finite language $L \subseteq \{\varepsilon, a, \ldots, a^{f(n)}\}$ can be generated by a CNF CFG with $n$ variables. This establishes the lower bound.

For the upper bound, consider that there are $n^3 + kn$ possible productions in a grammar with $n$ variables in CNF: $n^3$ of the form $A_i \to A_j A_k$ for any choice of $1 \leq i, j, k \leq n$ and $kn$ of the form $A_i \to a_j$ for any $1 \leq i \leq n$ and $1 \leq j \leq k$. Each subset of the $n^3 + kn$ possible productions defines a grammar, which yields an upper bound of $2^{n^3+kn}$. □

We can make the discussion more precise for finite unary context-free languages.

**Theorem 8.** *Let $U(n)$ measure the number of distinct unary finite languages generated by a CFG in CNF with $n$ variables. Then $\lg(U(n))$ satisfies*

$$\frac{n^3}{64} - 1 \leq \lg(U(n)) \leq \frac{n^3}{3} - n^2 + \frac{4n}{3} - \Theta(n \lg n)$$

*for $n$ sufficiently large.*

*Proof.* Let us first give the upper bound: Consider an arbitrary grammar in CNF with $n$ variables. Without loss of generality, let the variables be $A_1, A_2, \ldots, A_n$ with $A_1$ the start symbol. There are $\sum_{i=1}^{n}(n-i)(n-i-1)$ different productions of the form $A_i \to A_j A_k$ with $j \leq k$—the order of variables on the left-hand side is irrelevant, since unary languages commute, and the productions may be assumed to be of the form $A_i \to A_j A_k$ with $j, k > i$, since the language generated is finite. Finally, there are $n$ productions of the form $A_i \to a$. Therefore, simplifying there are $\frac{n^3}{3} - n^2 + \frac{4n}{3}$ possible productions, and $2^{\frac{n^3}{3}-n^2+\frac{4n}{3}}$ possible sets of productions. Noting that the names of the non-start symbols are irrelevant, we can divide this quantity by $(n-1)!$. Thus, by Stirling's Approximation, $\lg(U(n)) \leq \frac{n^3}{3} - n^2 + \frac{4n}{3} - \Theta(n \lg n)$.

For the lower bound, we make the lower bound of the proof of Theorem 7 more precise. In particular, if $f(n) = \frac{n^3}{64} - 1$, then all subsets of $\{\varepsilon, a, \ldots, a^{f(n)-1}\}$ can be generated by a CNF CFG with $4\lceil(\frac{n^3}{64} - 1)\rceil^{1/3} + 2 \leq n$ variables. This gives the result. □

### 5.3 Number of Universal Grammars

Motivated by Theorem 2, we can also give estimates on the number of CFGs generating $\Sigma^*$. Note that we are not concerned with enumerating CFLs, as we were in the previous sections, but grammars. We first consider the measure of the number of variables in CNF. Note that CNF grammars cannot generate $\Sigma^*$, so the theorem discusses grammars that generate $\Sigma^+$ (this quantity is also uncomputable, as is easily seen).

**Theorem 9.** *Let $n \geq 1$ and $k \geq 2$. Denote by $\mu_k(n)$ the number of CNF CFGs with $n$ variables generating $\Sigma^+$ where $|\Sigma| = k$. The following inequalities hold:*

$$(n-1)^3 + (n-1)^2 + (n-1)k \leq \lg(\mu_k(n)) \leq n^3 + nk$$

*Proof.* The upper bound is the trivial bound from Theorem 7. For the lower bound, consider a grammar in CNF with variables $\{A_1, \ldots, A_n\}$ with $A_1$ the start symbol. We set the productions of $A_1$ to include the following productions:

$$\begin{aligned} A_1 &\to A_1 A_1 \\ A_1 &\to a \quad \forall a \in \Sigma. \end{aligned}$$

Note that we now guarantee that the grammar generates $\Sigma^+$. Now, every grammar including these productions also generates $\Sigma^+$. Thus, the lower bound comes from choosing the remaining $(n-1)^2$ productions of the form $A_1 \to A_i A_j$ with $1 < i, j \leq n$, the $(n-1)^3$ productions of the form $A_i \to A_j A_k$ with $1 < i, j, k \leq n$, and the $(n-1)k$ productions of the form $A_i \to a$ with $1 < i \leq n$ and $a \in \Sigma$, in all possible ways. □

We can also construct a similar result when measuring CFGs by the number of symbols.

**Theorem 10.** *Let $n \geq 1$ and $k \geq 2$. Denote by $\nu_k(n)$ the number of CNF CFGs with $n$ symbols generating $\Sigma^*$ where $|\Sigma| = k$. The following inequalities hold:*

$$(n - 3k - 8)\lg(k+1) \leq \lg(\nu_k(n)) \leq (n-1)\lg(n/2 + k + 1).$$

*For the lower bound, we additionally require $n > 3k + 10$.*

*Proof.* The upper bound is from Theorem 3. The lower bound is similar to the proof of Theorem 9. Consider again the grammar with fixed start symbol $A$ and productions

$$\begin{aligned} A &\rightarrow AA \\ A &\rightarrow a \quad \forall a \in \Sigma. \\ A &\rightarrow \varepsilon \end{aligned}$$

Then the total length of these productions is $3k+6$. Consider the remaining length of $n-3k-6$ symbols. Any word $w$ over $\Sigma \cup \{A\}$ of length $n-3k-8$ can be extended to a production $A \rightarrow w$. For any such $w$, adjoining it to the $k+2$ productions above, we get a distinct grammar (provided $n-3k-8 > 2$, to prevent us from duplicating the above productions). This gives a lower bound on $\nu_k(n)$ of $(k+1)^{n-3k-8}$. □

# 6 General results on Uncomputability of Enumerative Functions

Theorems 1 and 2 share a single method of argument: it is shown that knowing the values of an enumerative function allows one to solve a certain related problem, and the undecidability of that problem implies the uncomputability of the enumerative function.

In this section we prove general theorems that can be used to establish results of this kind.

## 6.1 The number of objects satisfying a predicate

Consider the following abstract generalization of Theorem 1: instead of the set of context-free grammars, let us take any recursive set $X$ with any computable and well-behaved descriptional complexity measure $d$ defined on it; instead of the universality condition, let us take an arbitrary predicate $P$ on $X$. We denote by $f_P(n)$ the following quantity

$$f_P(n) = |\{x \in X \ : \ d(x) = n, \text{and } P(x) \text{ holds. }\}|.$$

That is, $f_P(n)$ denotes the number of words of measure $n$ in $X$ that satisfy the predicate $P$.

**Theorem 11.** *Let $X$ be a recursive language, let $\mathcal{O}$ be an oracle. Let $P$ be a predicate on $X$ that is recursively enumerable in $\mathcal{O}$ or co-recursively enumerable in $\mathcal{O}$. Then, if $f_P(n)$ is a computable function, then $P$ is recursive in $\mathcal{O}$.*

*Proof.* Assume $P$ is recursively enumerable (in $\mathcal{O}$), let $f_P$ be computable.

Input: $x \in X$, such that $d(G) = n$
Let $X_n = \{x_1, \ldots x_N\}$ be all elements of $X$ of measure $n$
Data structure: $\mathcal{C}$, a subset of $X_n$.

1. Let $\mathcal{C} = \varnothing$
2. Compute $m = f(n)$.
3. Start simulating the r.e. procedure for $P$ simultaneously on all $y \in X_n$
4. For every $y$ on which one of the procedures halts
5. Add $y$ to $C$
6. If $|C| = m$
7. End the simulation
8. Accept if $x \in \mathcal{C}$, reject otherwise.

All $f(n)$ yes-instances of the predicate will eventually be found, as the $m$ corresponding r.e. procedures terminate. At this point the lines (6–7) will be invoked, and the acceptance condition in line (8) determines the result: $x$ is accepted if and only if $P(x)$ is true. Since the constructed algorithm always halts, we have proved that $P$ is recursive in $\mathcal{O}$.

The case of co-recursively enumerable $P$ (as in the special case treated in Theorem 1) follows by observing that $\neg P$ is recursively enumerable in $\mathcal{O}$, and $f_{\neg P}(n) = |X_n| - f_P(n)$ is computable. Therefore, by the first part of the theorem, $\neg P$ is recursive in $\mathcal{O}$, and hence so is $P$. □

If the oracle is not used, the theorem infers recursiveness out of (co-)recursive enumerability of the predicate, provided that the enumerative function is computable. If we let $\mathcal{O}$ be the oracle for the TM halting problem, then Theorem 11 states that if $P$ is in $\Sigma_2$ or $\Pi_2$ in the arithmetical hierarchy and $f_P$ is computable, then $P \in \Sigma_2 \cap \Pi_2$. More generally, we have the following corollary:

**Corollary 1.** *Let $X$ be a recursive set, let $P$ be a predicate on $X$ that is in $\Sigma_k \cup \Pi_k$ for some $k \geqslant 1$. If $f_P$ is computable, then $P$ is in $\Sigma_k \cap \Pi_k$.*

In the following, $\Delta$ denotes the symmetric difference of two sets: $S_1 \,\Delta\, S_2 = (S_1 \setminus S_2) \cup (S_2 \setminus S_1)$.

**Corollary 2.** *For any predicate $P$ that is in $\Sigma_k \,\Delta\, \Pi_k$, the corresponding function $f_P$ is not computable.*

**Corollary 3.** *For any predicate $P$ that is complete for $\Sigma_k$ or for $\Pi_k$, the corresponding function $f_P$ is not computable.*

This allows us to obtain results of the following kind:

(a) The number of Turing machines of size $n$ that halt on $\varepsilon$ is uncomputable (because the decision problem is $\Sigma_1$-complete).

(b) The number of unambiguous context-free grammars of size $n$ is uncomputable (the decision problem is $\Pi_1$-complete).

(c) The number of context-free grammars of size $n$ that generate languages with a context-free complement is not computable (the decision problem is $\Sigma_2$-complete).

## 6.2 The number of equivalence classes

Let us similarly generalize Theorem 1, which states the uncomputability of the number of distinct CFLs generated by CFGs of a given size. In the general context, we consider a recursive set $X$ with a computable and well-behaved descriptional complexity measure $d$, and an equivalence relation $R$ on $X$, and we denote the number of equivalence classes on the elements of measure $n$ in $X$ by $g_R(n)$.

**Theorem 12.** *Let $X$ be a recursive language, let $R$ be an equivalence relation on $X$ that is recursively enumerable using an oracle $\mathcal{O}$ or co-recursively enumerable using $\mathcal{O}$. Then, if $g_R(n)$ is a computable function, then $R$ is recursive in $\mathcal{O}$.*

*Proof.* Let us start from the case of a recursively enumerable $R$. Using the computability of $g_R$, construct the following decision procedure for $R$:

Input: $x, y \in X$, such that $d(x) = d(y) = n$
Let $X_n = \{z_1, \ldots z_N\} \subseteq X$ be all elements of $X$ of measure $n$.
Data structure: $\mathcal{C}$, a partition of $X_n$.

1. Let $\mathcal{C} = \big\{\{z_1\}, \ldots, \{z_N\}\big\}$.
2. Compute $m = g_R(n)$.
3. Start simulating the r.e. procedure for $P$ simultaneously on all pairs $(x', x'') \in X_n \times X_n$
4. For every pair $(x', x'')$, on which one of the procedures halts
5. If $x'$ and $x''$ belong to different classes $\mathcal{C}_1, \mathcal{C}_2$ in $\mathcal{C}$,
6. Merge $\mathcal{C}_1, \mathcal{C}_2$ into one class
7. If $|\mathcal{C}| = m$,
8. End the simulation.
9. Accept if $x$ and $y$ are in the same class in $\mathcal{C}$, reject otherwise.

Once the equivalence of any members is established, they are moved to a single class in the partition. Since the relation is recursively enumerable, the equivalence of all equivalent members will eventually be determined, at which moment $|\mathcal{C}|$ will reach the value $g_R(n)$. Since the algorithm knows this number, it will stop and correctly determine whether $x$ and $y$ are equivalent or not.

The second case, where $R$ is co-recursively enumerable, is handled symmetrically.

Input: $x, y \in X$, such that $d(x) = d(y) = n$
Let $X_n \subseteq X$ contain all elements of $X$ of measure $n$.
Data structures: (i) $\mathcal{C}$, a partition of $X_n$; (ii) $S$, a set of pairs from $X_n$.

1. Let $\mathcal{C} = \big\{X_n\big\}$.
2. Compute $m = g_R(n)$.
3. Start simulating the r.e. procedure for $\neg P$ simultaneously on all $(x', x'') \in X_n \times X_n$
4. For every pair $(x', x'')$, on which one of the procedures halts
5. Add $(x', x'')$ to $S$.
6. If $x'$ and $x''$ belong to the same class $C \in \mathcal{C}$
7. If there exists a partition $C = C_1 \cup C_2$, such that $C_1 \times C_2 \subseteq S$,
8. Split $C$ into $C_1$ and $C_2$.
9. If $|\mathcal{C}| = m$,
10. End the simulation.
11. Accept if $x$ and $y$ are in the same class in $\mathcal{C}$, reject otherwise.

Note that, unlike Theorem 1, an equivalence class cannot be split in two immediately upon finding any inconsistency: in order to determine the exact partition, the algorithm has to wait until all the negative information is accumulated in $S$, when the condition in line (7) becomes true. This eventually happens, because $\neg P$ is recursively enumerable, and hence $\mathcal{C}$ is eventually split into $m$ classes. Knowing $g_R(n)$, the algorithm can stop and make the correct decision. □

**Corollary 4.** *Let $R$ be an equivalence relation on $X$ that is in $\Sigma_k \cup \Pi_k$ for some $k \geqslant 1$. If $g_R$ is computable, then $R$ is in $\Sigma_k \cap \Pi_k$.*

**Corollary 5.** *For any equivalence relation $R$ that is in $\Sigma_k \,\Delta\, \Pi_k$, the corresponding function $g_R$ is not computable.*

**Corollary 6.** *For any equivalence relation $R$ that is complete for $\Sigma_k$ or for $\Pi_k$, the corresponding function $g_R$ is not computable.*

In particular, Corollary 6 implies the earlier Theorem 1. Let us note some further instances of these results:

(a) The number of distinct rational relations defined by nondeterministic finite transducers with $n$ states is uncomputable (their equivalence problem is known to be undecidable [8] and can be easily seen to be $\Pi_1$-complete).

(b) The number of distinct recursively enumerable languages recognized by Turing machines of size $n$ is uncomputable (because the TM equivalence problem is $\Pi_2$-complete).

We note in passing that example (a) is related to an open problem posed by Harrison. Let $M = (Q, \Sigma, \delta, \tau)$ be a sequential transducer and $R_M$ be defined by $R_M = \cup_{q \in Q} \, _{x \in \Sigma^*} \{(x, \tau(q, x))\}$. In other words, for any pair of strings $x \in \Sigma^*$ and $y \in \Delta^*$, $x \, R_M \, y$ holds if and only if there exists a state $q \in Q$ such that $y = \tau(q, x)$ (see Gray and Harrison [7] for the definitions and more background on sequential transducers). Harrison asks for the number of distinct binary relations $R$ defined by sequential transducers of size $n$ for fixed alphabets $\Sigma, \Delta$.

A limitation common to both types of our general negative results is that Corollary 2 and Corollary 5 are not applicable to undecidable predicates and equivalence relations located between the levels of the arithmetical hierarchy. Nothing can be inferred about the computability of associated enumerative functions in those cases. Sometimes they turn out to be computable, and sometimes their computability status cannot be determined by our methods.

For instance, consider the following relation on the set of Turing machines: $M_1 \sim M_2$ if and only if either both of them halt on $\varepsilon$, or neither of them does. It is easy to see that this is an equivalence relation, and it is between the levels of the arithmetical hierarchy–to be precise, is in $(\Sigma_2 \cap \Pi_2) \setminus (\Sigma_1 \cup \Pi_1)$. For this relation, the number of classes of equivalence is trivially computable, because there are exactly two such classes: those machines that halt on $\varepsilon$ and those that do not.

Let us consider another equivalence relation, this time on context-free grammars. Let us say that $G_1 \sim G_2$ if and only if either (a) $L(G_1) = L(G_2)$ and this language is regular, or (b) both $L(G_1)$ and $L(G_2)$ are not regular. It is easy to see that the number of equivalence classes on grammars of measure $n$ is exactly the number of regular languages generated by such grammars plus 1. The relation is both $\Sigma_2$-hard and $\Pi_2$-hard by a reduction from the CFG regularity and non-regularity problems, respectively, and it is decidable using an oracle in $\Sigma_2$. This puts it into $(\Sigma_3 \cap \Pi_3) \setminus (\Sigma_2 \cup \Pi_2)$, so Corollary 5 is again not applicable, and it remains unknown whether this number can be effectively computed.

# 7 Conclusions

We have shown that the number of context-free languages of a fixed size, over an arbitrary fixed alphabet is uncomputable. However, we have also given estimates on the growth of these uncomputable functions.

Several interesting uncomputability results remain open. In particular, we leave open the problem of the uncomputability of functions related to enumeration of predicates and equivalence relations which are not complete for any level of the arithmetic hierarchy. We also leave open the

following interesting particular instance of this problem: Is the number of regular languages generated by CFGs of a fixed size computable? We conjecture it is not.

## References


[1] BERTONI, A., GOLDWURM, M., AND SABADINI, N. The complexity of counting the number of strings of given length in context-free languages. *Theoretical Computer Science 86* (1991) 325–342.

[2] DOMARATZKI, M., KISMAN, D., AND SHALLIT, J. On the number of distinct languages accepted by finite automata with $n$ states. *Journal of Automata, Languages and Combinatorics 7*, 4 (2002), 469–486.

[3] DOMARATZKI, M., PIGHIZZINI, G., AND SHALLIT, J. Simulating finite automata with context-free grammars. *Information Processing Letters 84* (2002), 339–344.

[4] DÖMÖSI, P. Unusual algorithms for lexicographical enumeration. *Acta Cybernetica 14* (2000), 461–468.

[5] CSUHAJ-VARJÚ, E., DASSOW, J., KELEMEN, J., AND PĂUN, G. *Grammar Systems*, Gordon and Breach, 1994.

[6] GORE, V., JERRUM, M., KANNAN, S., SWEEDYK, Z., AND MAHANEY, S. A quasi-polynomial time algorithm for sampling words from a context-free language. *Information and Computation 134* (1997), 59–74.

[7] GRAY, J., AND HARRISON, M. The theory of sequential relations. *Information and Control 9* (1966), 435–468.

[8] GRIFFITHS, T., The Unsolvability of the Equivalence Problem for $\lambda$-Free Nondeterministic Generalized Machines. *Journal of the ACM 15* (1968), 409–413.

[9] GRUSKA, J. Some classifications of context-free languages. *Information and Control 14*, 2 (1969), 152–179.

[10] GRUSKA, J. Complexity and unambiguity of context-free grammars and languages. *Information and Control 18*, 5 (1971), 502–519.

[11] HOPCROFT, J. E., AND ULLMAN, J. D. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[12] LEE, J., AND SHALLIT, J. Enumerating regular expressions and their languages. In *Implementation and Application of Automata: 9th International Conference, CIAA 2004* (2005), M. Domaratzki, A. Okhotin, K. Salomaa, and S. Yu, Eds., vol. 3317 of *LNCS*, Springer-Verlag, pp. 2–22.

[13] MARTINEZ, A. Topics in Formal Languages: String Enumeration, Unary NFAs and State Complexity. M.Math Thesis, University of Waterloo, 2002.

[14] YU, S. Regular languages. In *Handbook of Formal Languages, Vol. I*, G. Rozenberg and A. Salomaa, Eds. Springer-Verlag, 1997, pp. 41–110.