

# Cocke–Younger–Kasami–Schwartz–Zippel algorithm and relatives

Vladislav Makarov\*

December 8, 2022

## Abstract

The equivalence problem for unambiguous grammars is an important, but very difficult open question in formal language theory. Consider the *limited* equivalence problem for unambiguous grammars — for two unambiguous grammars  $G_1$  and  $G_2$ , *and  $n$*  tell whether or not they describe the same set of words of length  $n$ . Obviously, the naive approach requires exponential time with respect to  $n$ . By combining two classic algorithmic ideas, I introduce a  $O(\text{poly}(n, |G_1|, |G_2|))$  algorithm for this problem. Moreover, the ideas behind the algorithm prove useful in various other scenarios.

## 1 Preface

This Section contains some details about the technical structure of this paper and the reasons for its existence. Therefore, feel free to skip it. Just remember that this paper has a “prequel”: “Why the equivalence problem for unambiguous grammars has not been solved back in 1966?”.

This paper and the companion paper “Why the equivalence problem for unambiguous grammars has not been solved back in 1966?” are based on the Chapters 4 and 3 of my Master’s thesis [22] respectively. While the thesis is published openly in the SPbSU system, it has not been published in a peer-reviewed journal (or via any other scholarly accepted publication method) yet.

These two papers are designed to amend the issue. As of the current date, I have not submitted them to a refereed venue yet. Therefore, they are only published as arXiv preprints for now.

Considering the above, it should not be surprising that huge parts of the original text are copied almost verbatim. However, the text is not totally the same as the Chapter 4 of my thesis. Some things are altered for better clarity of exposition and there are even some completely new parts.

---

\*Saint-Petersburg State University. Supported by Russian Science Foundation, project 18-11-00100.

Why did I decide to split the results into two papers? There are two main reasons.

Firstly, both papers are complete works by themselves. From the idea standpoint, some of the methods and results of this paper were motivated by the careful observation of results of the companion paper. However, the main result of this paper is stated and proven without any explicit references to the content of the companion paper.

The second reason is explained in the “prequel” paper in great detail. For simplicity, let us say that this paper contains specific “positive” results, while the previous focuses on more vague “negative” results.

With technical details out of the way, let us move on to the mathematical part.

## 2 Cocke–Younger–Kasami–Schwartz–Zippel algorithm

Recall the statement of the equivalence problem for unambiguous grammars.

**Problem 1** (The equivalence problem for unambiguous grammars.). *You are given two ordinary grammars  $G_1$  and  $G_2$ . Moreover, you know that they are both unambiguous from a 100% trustworthy source. Is there an algorithm to tell whether  $L(G_1)$  and  $L(G_2)$  are equal? Because the grammars are guaranteed to be unambiguous, the algorithm may behave arbitrarily if either of  $G_1$  and  $G_2$  is ambiguous, including not terminating at all.*

In the “prequel” paper we studied the extents and limitations of matrix substitution. Now, let us focus on another approach.

**Definition 1.** For a language  $L$ , its  $n$ -slice is the language  $\{w \mid w \in L, |w| = n\}$  of all words from  $L$  of length exactly  $n$ .

As we have seen in the “prequel”, matrix substitution is in some way “independent” over the slices of  $L(G_1)$  and  $L(G_2)$ :  $L(G_1)$  and  $L(G_2)$  are  $d$ -similar if and only if  $n$ -slices of  $L(G_1)$  and  $L(G_2)$  are  $d$ -similar for all  $n$ .

The main advantage of the matrix substitution approach is its *uniformity*: it checks all  $n$ -slices for similarity at the same time. However, we can do better if the uniformity is not required. Specifically, consider the following problem:

**Problem 2.** *Given two unambiguous grammars  $G_1$  and  $G_2$ , tell whether  $n$ -slices of  $L(G_1)$  and  $L(G_2)$  are equal.*

The naive solution for Problem 2 takes  $\Theta(|\Sigma|^n \cdot \text{poly}(n, |G_1|, |G_2|))$  time. Fortunately, it is unnecessary to iterate over all strings:

**Theorem 1** (Cocke–Younger–Kasami–Schwartz–Zippel algorithm). *If both  $G_1$  and  $G_2$  are in Chomsky normal form, it is possible to solve Problem 2 in  $O(n^3 \cdot (|G_1| + |G_2|))$  time with bounded one-sided error from randomization.*

*Remark.* Of course, Theorem 1 cannot help with Problem 1 at all, as long as we are interested in *decidability only*. However, the running time improvement over the naive algorithm is drastic, making iterating over all small  $n$  a viable heuristic for Problem 1.

The rest of this Section is dedicated to the proof of Theorem 1, and, even more importantly, overview of the main ideas. The next several paragraphs only provide motivation for the ideas that are used in the proof, so I will keep details a bit vague in them. Moreover, these paragraphs use the ideas from the “prequel” paper; feel free to skip them, as the proof itself should also be clear enough.

Let us look back at the Amitsur–Levitsky identity. As I noted before, it looks like the definition of determinant. To get *exactly* the definition of determinant, we need to add another index to each  $X_i$ :

$$\text{before: } \sum_{\sigma \in S_{2d}} (-1)^{\text{sgn}(\sigma)} X_{\sigma(1)} X_{\sigma(2)} \dots X_{\sigma(2d)} \quad (1)$$

$$\text{after: } \sum_{\sigma \in S_{2d}} (-1)^{\text{sgn}(\sigma)} X_{1,\sigma(1)} X_{2,\sigma(2)} \dots X_{2d,\sigma(2d)} \quad (2)$$

While the Expression (1) is a polynomial identity for  $d \times d$  matrices, the Expression (2) is not, even for  $1 \times 1$  matrices (that is, scalars). Indeed, there *are* some matrices with non-zero determinant.

Now, suppose that we have two unambiguous grammars  $G_+$  and  $G_-$  over an alphabet  $\Sigma := \{a_1, a_2, \dots, a_n\}$ , generating even and odd permutations of length  $2d$  respectively. More formally,  $L(G_+) = \{a_{\sigma(1)} a_{\sigma(2)} \dots a_{\sigma(2d)} \mid \sigma \in S_{2d}, \text{sgn}(\sigma) = +1\}$  and  $L(G_-) = \{a_{\sigma(1)} a_{\sigma(2)} \dots a_{\sigma(2d)} \mid \sigma \in S_{2d}, \text{sgn}(\sigma) = -1\}$ . Then, by Amitsur–Levitsky theorem, we cannot tell them apart by substituting matrices of size  $d \times d$  and smaller. Of course, we can pick larger matrices or bash this specific instance of the Problem 1 in many other ways, because both languages are finite. But it is not the point, so let us not do that.

In the “prequel”, we had no choice except for representing everything implicitly via equations and rely on Tarski–Seidenberg theorem to rule everything out. There are two reasons for that: we wanted everything to work for all matrices of a small norm and we wanted to handle all  $n$ -slices at the same time. If we know the matrices beforehand and want to consider only a specific slice (the  $(2d)$ -th slice in our case), we can compute the values of  $f$  with dynamic programming. And if we somehow manage incorporate the second indices (as in Expression (2)) in the dynamic programming, then there will not be any systematic problem with polynomial identities (accidental coincidences can still happen, though).

As it turns out, to solve Problem 2, we need only these two ideas (an *indexation trick*, as I call it, and a somewhat careful dynamic programming), but we do not need to substitute matrices anymore, just numbers will be enough. These ideas will be crucial through the rest of the text, so I recommend to understand them well.

*Proof of Theorem 1, assuming Theorem 2.* Specifically, consider the following mapping  $f$  from subsets of  $\Sigma^n$  to polynomials with integer coefficients in  $|\Sigma| \cdot n$  variables  $x_{a,i}$ , where  $a$  ranges over  $\Sigma$  and  $i$  ranges over  $[1, n]$ : a single word  $w_1 w_2 \dots w_n$  maps to a monomial  $f(w) := x_{w_1,1} \cdot x_{w_2,2} \cdot \dots \cdot x_{w_n,n}$  and a language  $L \subset \Sigma^n$  maps to  $f(L) := \sum_{w \in L} f(w)$ . Here, I abuse notation a little bit, because  $f$  is actually two different mappings: one is from  $\Sigma^n$  to

monomials and the other is from subsets of  $\Sigma^n$  to polynomials. I hope that this does not cause confusion.

It is important that these polynomials are normal, commutative polynomials, no trickery here (at least in the proof of Theorem 1). And there is no need to substitute matrices in place of variables, because the indexing itself takes care of noncommutativity of word concatenation. For example,  $f(ab) = x_{a,1}x_{b,2}$ , but  $f(ba) = x_{b,1}x_{a,2} = x_{a,2}x_{b,1} \neq x_{a,1}x_{b,2}$ . Hence, different subsets of  $\Sigma^n$  have different images under  $f$ : if a word  $w$  is present in  $K$ , but not in  $L$ , then the monomial  $f(w) = x_{w_1,1}x_{w_2,2} \dots x_{w_n,n}$  is present in  $f(K)$ , but not in  $f(L)$ . Moreover, all coefficients of  $f(K)$  are either 0 or 1, depending on whether the corresponding word belongs to  $K$  or not.

In particular, if  $L_1$  and  $L_2$  are  $n$ -slices of  $L(G_1)$  and  $L(G_2)$  respectively, then  $f(L_1) = f(L_2)$  if and only if  $L_1 = L_2$ . Moreover, both  $f(L_1)$  and  $f(L_2)$  are polynomials of degree at most  $n$ , because they both are sums of several monomials of degree exactly  $n$ . Therefore, we can use Schwartz-Zippel lemma here to compare them without computing them explicitly.

To be exact, let us fix some large finite field  $F$ . Because all coefficients of  $f(L_1)$  and  $f(L_2)$  are zeroes or ones,  $f(L_1)$  and  $f(L_2)$  are equal as integer polynomials if and only if they are equal as polynomials over  $F$ . Evaluate them in a random point from  $F^{|\Sigma|^n}$  by the following Theorem 2. By Schwartz-Zippel lemma, the probability of a false positive (polynomials are not equal, but their values are) is at most  $\frac{n}{|F|}$ , which is at most  $1/2$  when  $|F| \geq 2n$ . Of course, there are no false negatives. Repeat several times to obtain the desired error probability.  $\square$

**Theorem 2.** *Given an unambiguous grammar  $G$  in Chomsky normal form and a field  $F$ , it is possible to evaluate  $f(K)$  in a point  $x \in F^{|\Sigma|^n}$  in  $O(n^3 \cdot |G|)$  field operations, where  $K$  is the  $n$ -th slice of  $L(G)$ .*

*Proof.* The proof is an adaptation of standard Cocke-Younger-Kasami cubic parsing algorithm. For a non-empty subsegment  $[\ell, r]$  of  $[1, n+1]$  and a language  $K \subset \Sigma^{r-\ell}$ , define  $f_{[\ell, r]} := \sum_{w \in K} x_{w_1, \ell} x_{w_2, \ell+1} \dots x_{w_{r-\ell}, r-1}$ . This is a generalisation of  $f$ . Indeed,  $f_{[1, n+1]}(K) = f(K)$  when  $K \subset \Sigma^n$ , and, therefore, both sides of the equation are defined.

Then,  $f_{[\ell, r]}(K \sqcup L) = f_{[\ell, r]}(K) + f_{[\ell, r]}(L)$  and  $f_{[\ell, r]}(KL) = f_{[\ell, m]}(K)f_{[m, r]}(L)$ , as long as everything is defined. Note that there is no requirement for the concatenation  $KL$  to be unambiguous, because it is unambiguous automatically. Indeed, all words in  $K$  have length  $m - \ell$  and all words in  $L$  have length  $r - m$ . Hence, for any word  $w$  in  $KL$ , there is only one way to represent it as a concatenation of a word from  $K$  and  $L$ :  $w_1 \dots w_{m-\ell} \in K$  and  $w_{m-\ell+1} \dots w_{r-\ell} \in L$ .

Hence, for a nonterminal  $C$  of  $G$  and a non-empty subsegment  $[\ell, r]$  of  $[1, n+1]$ ,

$$f_{[\ell, r]}(L(C)) := \begin{cases} \sum_{(C \rightarrow a) \in R} x_{a, \ell}, & \text{if } r - \ell = 1, \\ \sum_{(C \rightarrow DE) \in R} \sum_{m=\ell+1}^{r-1} f_{[\ell, m]}(L(D))f_{[m, r]}(L(E)), & \text{otherwise.} \end{cases} \quad (3)$$

In the second of the above cases, we implicitly use the unambiguity of concatenation  $L(D)L(E)$  by iterating over  $m$ .

Now, let us evaluate  $f_{[\ell,r]}(L(C))$  in the point  $x$ . To do so, apply the above formulas in the order of increasing  $r - \ell$ . In the end, the value of  $f_{[1,n+1]}(L(S))$  in  $x$  is exactly what we needed to compute. The total number of field operations is  $O(n^3 \cdot |R|) = O(n^3 \cdot |G|)$ : for each triple  $\ell < m < r$ , we iterate over all “normal” rules of the grammar.  $\square$

*Remark.* It is possible to check whether  $L(G_1)$  and  $L(G_2)$  have a difference of length *at most*  $n$  (rather than *exactly*  $n$ , as in Theorem 1) with the same running time. To do so, run the above procedure, but compare the sums  $\sum_{i=2}^{n+1} f_{[1,i]}(L(G_1))$  and  $\sum_{i=2}^{n+1} f_{[1,i]}(L(G_2))$  instead of  $f_{[1,n+1]}(L(G_1))$  and  $f_{[1,n+1]}(L(G_2))$ .

Let us look at the polynomial  $f$  more closely. As a polynomial with integer coefficients, it can be seen as a function from  $\mathbb{Z}^{|\Sigma| \cdot n}$  to  $\mathbb{Z}$ . An *input* for  $f$  is any assignment of integer values to variables  $x_{a,i}$ , where  $a$  ranges over  $\Sigma$  and  $i$  ranges over  $[1, n]$ . Intuitively, the variable  $x_{a,i}$  represents “to which extent” the letter  $a$  appears in the  $i$ -th position. There are two special kinds of inputs: null-like and word-like.

**Definition 2.** An input  $x$  is *null-like* if there exists an index  $i$  from 1 to  $n$ , such that  $x_{a,i} = 0$  for all  $a \in \Sigma$ .

**Definition 3.** An input  $x$  is *word-like* if for all indices  $i$  from 1 to  $n$  there is *exactly one* such  $a \in \Sigma$ , that  $x_{a,i} = 1$  and  $x_{b,i} = 0$  for all  $b \in \Sigma \setminus \{a\}$ .

Null-like inputs are not interesting: because all  $x_{a,i} = 0$  for all  $a \in \Sigma$ , the  $i$ -th variable in each monomial of  $f$  evaluates to 0. Hence, each monomial evaluates to 0 and so does  $f$ .

Word-like inputs are more important. A word-like input is a redundant representation of a word: instead of just  $n$  letters, we have  $|\Sigma| \cdot n$  answers to questions “is there a letter  $a$  on the  $i$ -th position?”. And, fittingly, there is exactly one letter on each position.

Consider a word-like input  $x$  corresponding to a word  $w$ . Clearly, on the input  $x$ , the monomial  $x_{u_1,1}x_{u_2,2} \dots x_{u_n,n}$  evaluates to 1 if  $w = u$  and to 0 otherwise. Hence, the value of  $f(K)$  on a word-like input simply shows whether the corresponding word lies in  $K$  or not. This is just the parsing problem for the grammar  $G$  and the word  $w$ .

In fact, under a closer examination (I will not dwell in the details), the proof of Theorem 2 becomes *exactly* the standard cubic-time parsing algorithm when we consider only word-like inputs  $x$ .

In a sense, the whole approach has even remote chances of working *exactly* because we consider a much larger set of inputs, not only word-like ones. Restricting ourselves only to word-like input would be the same as picking random *words* of length  $n$  and checking whether they belong to  $L_1$  and  $L_2$ . Picking random *words* works poorly, for example, if  $L_1$  and  $L_2$  are obtained by explicitly adding two different words to the same unambiguous grammar (and in the myriad of more complicated situations).

The above observation is extremely important in the following Sections. We will see its “positive” side in Section 3 and its “negative” side in Section 4.

### 3 Relations to circuit complexity

Of course, there is a more general idea in the proof of Theorem 1:

**Idea 1.** *To prove something about  $n$ -slices of  $L(G)$ , where  $G$  is some type of grammar (unambiguous, ordinary, GF(2), et cetera), define  $f(K)$  in terms of the operations that are “inherent” to the corresponding grammar formalism. Evaluate  $f(L(G))$  similarly to Theorem 2 and somehow exploit the fact that evaluating  $f(L(G))$  is a much more general problem than string parsing for  $G$ .*

For example, let me sketch the proof of the following result:

**Theorem 3.** *Given a GF(2)-grammar  $G$ , it is possible to tell whether  $n$ -slice of  $L(G)$  is empty in  $O(n^3 \cdot |G|)$  time with randomization and bounded one-sided error.*

*A sketch.* The proof is mostly the same. Let us focus on the differences. The first difference is that we define  $f$  and  $f_{[\ell,r]}$  as polynomials over  $\mathbb{F}_2$ , not  $\mathbb{Z}$ , because we have symmetric difference and GF(2)-concatenation instead of disjoint union and unambiguous concatenation now.

The second difference is that  $F$  has to be a field of characteristic 2 (otherwise the notion of evaluating  $f(K)$  on a point of  $F$  makes no sense). Again, this is not a problem: for each  $b$ , there is a field of characteristic 2 and size  $2^b$ .  $\square$

*Remark.* Of course, Theorem 1 immediately follows from Theorem 3, but I think that the ideas are easier to understand when they are illustrated on the proof of Theorem 1.

What is, in my opinion, much more interesting and important, is that we can use Idea 1 to prove some *lower bounds* style results. The general “line of attack” is like this:

1. Consider a language  $L_{\text{orig}}$  over an alphabet  $\Sigma$ . Assume that there is a grammar  $G$  of some type (unambiguous, ordinary, GF(2), et cetera) for  $L_{\text{orig}}$ .
2. By applying standard grammar closure properties (closure under certain types of transductions, set operations, et cetera), obtain a family  $L_n$  of languages, each over its own alphabet  $\Sigma_n$ , such that a grammar  $G$  for  $L$  leads to a grammar  $G_n$  of size  $\text{poly}(n, |G|)$  for  $L_n$ . The alphabets  $\Sigma_n$  can depend on  $n$  and even grow in size polynomially with  $n$ .
3. Use the Idea 1 to evaluate some kind of function on  $|\Sigma| \cdot n$  variables with some kind of circuit of size  $\text{poly}(n, |G_n|) = \text{poly}(n, |G|)$ . The exact nature of the variables, the function and the circuit depends on the type of grammar.
4. Use some kind of circuit lower bound (either already known or a conjectured one) to show that this function cannot be evaluated by a small circuit of a small size.
5. Hence, there was no grammar for  $L_{\text{orig}}$  in the first place.

The exact details of the plan are flexible. For example, sometimes it is possible to omit the first step altogether and still get some kind of meaningful result. Moreover, if the circuits on the step 4 are unrestricted boolean circuits and step 3 takes polynomial time, then we can replace “circuit lower bound” by a “algorithm running time lower bound” in the step 4, because all circuits here are generated by a uniform polynomial-time procedure. In principle, it is possible to replace all mentions of “polynomial time” with explicit functions that may not even be polynomial (say, like,  $2^{n/10}$ ), but I do not yet know any situation where that would significantly help.

To see, how the plan would work for ordinary grammars, consider the language  $P_n$  of all permutations of length  $n$  over the alphabet  $\Sigma_n = \{1, 2, \dots, n\}$  (yes, the digits are the letters here). For example,  $P_3 = \{123, 132, 213, 231, 312, 321\}$ . It is already known that the size of the smallest ordinary grammar for  $P_n$  grows with  $n$  exponentially [10, Theorem 30]. Let us prove that the growth is superpolynomial with our method.

Let us define  $f(K)$  as a boolean function:  $f(w) := x_{w_1,1} \wedge x_{w_2,2} \wedge \dots \wedge x_{w_n,n}$  and  $f(K) := \bigvee_{w \in K} f(w)$ . Why a boolean function? Because set union and concatenation are expressed in terms of disjunction and conjunction. Such a definition works well with concatenation and set union, which correspond to the conjunction and disjunction respectively.

Specifically,  $f_{[\ell,r]}(K \cup L) = f_{[\ell,r]}(K) \vee f_{[\ell,r]}(L)$  and  $f_{[\ell,r]}(KL) = f_{[\ell,m]}(K) \wedge f_{[m,r]}(L)$ , as long as everything is defined. The latter is obvious, so let me illustrate the former on an example:  $f_{[1,3]}(\{ab, aa\}) \wedge f_{[3,5]}(\{cc, ad\}) = ((x_{a,1} \wedge x_{b,2}) \vee (x_{a,1} \wedge x_{a,2})) \wedge ((x_{c,3} \wedge x_{c,4}) \vee (x_{c,3} \wedge x_{d,4})) = ((x_{a,1} \wedge x_{b,2} \wedge x_{c,3} \wedge x_{c,4}) \vee (x_{a,1} \wedge x_{a,2} \wedge x_{c,3} \wedge x_{c,4}) \vee (x_{a,1} \wedge x_{a,2} \wedge x_{c,3} \wedge x_{d,4}) \vee (x_{a,1} \wedge x_{a,2} \wedge x_{d,3} \wedge x_{d,4})) = f_{[1,5]}(\{abcc, abad, aacc, aaad\}) = f_{[1,5]}(\{ab, aa\} \cdot \{cc, ad\})$  by distributivity properties.

In the end, we get a *monotone* (only disjunctions and conjunctions) boolean circuit of size  $O(n^2 \cdot |G_n|)$  that computes  $f(P_n) = \bigvee_{\sigma \in S_n} (x_{\sigma(1),1} \wedge x_{\sigma(2),2} \dots \wedge x_{\sigma(n),n})$ . In other words, a monotone circuits that checks whether there is a perfect matching in a bipartite graph with  $n$  vertices in each part. (the variables  $x_{i,j}$  correspond to the presence or absence of an edge between the  $i$ -th vertex on the left and the  $j$ -th vertex on the right). Razborov [25] proved that this problem requires monotone circuits of size  $n^{\Omega(\log n)}$ . Hence,  $|G_n|$  grows superpolynomially.

To get a grammar nonexistence result for a specific language (as I said, this is often unnecessary), encode the letter  $i$  in  $\Sigma_k$  by the word  $w_{k,i} := 0^{i-1}10^{k-i}$  over an alphabet  $\Sigma = \{0, 1\}$ . Now, define  $P_{\text{orig}}$  as the union of  $P_{\text{base}} := \bigcup_{k=0}^{+\infty} \{w_{k,\sigma(1)}w_{k,\sigma(2)} \dots w_{k,\sigma(k)} \mid \sigma \in S_k\}$  and *any* set  $P_{\text{extra}}$  of words that do not decode to anything. That is,  $P_{\text{extra}}$  can be any subset of  $\{0, 1\}^*$  that does not intersect  $\bigcup_{k=0}^{+\infty} \{w_{k,i} \mid 1 \leq i \leq n\}^*$ . Then, there is no ordinary grammar for  $P_{\text{orig}}$ , independently of the choice of the  $P_{\text{extra}}$ .

Indeed, to get  $P_n$  from  $P_{\text{orig}}$  we need to intersect with  $\Sigma^{n^2}$  (to leave only the words of correct length) and decode the letters by replacing  $w_{n,i}$  with  $i$  (this can be done by a deterministic transducer). Because  $P_{\text{extra}}$  specifically can contain only words that do not successfully decode to anything, the resulting language will be exactly  $P_n$ . Moreover, each

of the two steps blows up the grammar size only by a polynomial in  $n$  factor.

Hence, there is no ordinary grammar for  $P$  (grammar for  $P$  implies small grammars for  $P_n$ , and that in turn implies small monotone circuits for perfect matching). To be honest, this is not a particularly interesting (or new) result, but it does a good job of highlighting the steps of the plan.

## 4 Can we do the same with conjunctive grammars?

In the previous Section we have seen how the indexing trick helps with lower-bound style results for unambiguous and even arbitrary ordinary grammars. Can we do something similar with conjunctive grammars? This is an interesting question, because there is currently no known satisfying methods of proving that some language cannot be described by a conjunctive grammar, except for the following theorem which is based on the complexity of standard algorithms for parsing:

**Theorem A.** *If  $L$  is described by a conjunctive grammar, then  $L \in \text{DTIME}(O(n^3)) \cap \text{DSpace}(O(n))$ .*

Unfortunately, no. Indeed, while there is no small monotone circuits for bipartite matching, *there is* a polynomially-sized conjunctive grammar for  $P_n$ . Moreover,  $P_n$  is an intersection of several regular languages:  $P_n = \Sigma^n \cap \bigcap_{i=1}^n (\Sigma^* i \Sigma^*)$ , where  $\Sigma = \{1, 2, \dots, n\}$ . So, what does go wrong? In short, everything.

Because conjunctive grammars are an extension of ordinary grammars, we need to define  $f$  the same way as we did for ordinary grammars:  $f(w_1 \dots w_n) := (x_{w_1,1} \wedge x_{w_2,2} \wedge \dots \wedge x_{w_n,n})$  and  $f(L) := \bigvee_{w \in L} f(w)$ . Then, the concatenation and the union of languages will work as expected. Unfortunately, the same is not true for the intersection:  $f(\{ab\} \cap \{ac\}) = f(\emptyset) = 0 \neq (x_{a,1} \wedge x_{b,2} \wedge x_{c,2}) = (x_{a,1} \wedge x_{b,2}) \wedge (x_{a,1} \wedge x_{c,2}) = f(\{ab\}) \wedge f(\{bc\})$ .

In fact, not only the conjunction does not work, but there is no such boolean function  $\varphi$ , that  $f(K \cap L) = \varphi(f(K), f(L))$ . Indeed, assume the contrary. Consider an assignment of variables that maps all  $|\Sigma| \cdot n$  variables to 1. Under such an assignment,  $0 = f(\emptyset) = f(\{ab\} \cap \{ac\}) = \varphi(f(\{ab\}), f(\{ac\})) = \varphi(x_{a,1} \wedge x_{b,2}, x_{a,1} \wedge x_{c,2}) = \varphi(1, 1) = \varphi(x_{a,1} \wedge x_{b,2}, x_{a,1} \wedge x_{b,2}) = \varphi(f(\{ab\}), f(\{ab\})) = f(\{ab\} \cap \{ab\}) = f(\{ab\}) = x_{a,1} \wedge x_{b,2} = 1$ , contradiction (all these equation signs correspond to equality in the aforementioned point and not to the equality of functions).

Hence, there is no way to “translate” the intersection of languages to a boolean operation on the corresponding functions. The issue here is the existence of *extra* conjuncts like  $x_{a,1} \wedge x_{b,2} \wedge x_{c,2}$ , which do not correspond to any word, because of a repeated index (in this case, index 2). So, if we blindly transform the grammar into a monotone circuit, the resulting circuit will not compute  $f(K)$  itself, but, rather,  $f(K)$  with some extra conjuncts.

So, how to deal with extra conjuncts? I do not know any promising way, except for considering only such inputs, that two variables with the same index cannot be mapped to 1 at the same time (then, each extra conjunct is automatically mapped to 0). Or, in



existing terms, by considering word-like inputs only. Null-like inputs are irrelevant, because  $f(K)$  always evaluates to 0 on them.

Then, the rest of argument will proceed without a hitch, leading to the following result: if  $G$  is a conjunctive grammar and  $K$  is the  $n$ -slice of  $L(G)$ , then there is a polynomial-size monotone boolean circuit that outputs the same values as the monotone boolean function  $f(K) = \bigvee_{w \in K} (x_{w_1,1} \wedge \dots \wedge x_{w_n,n})$  on all word-like inputs.

Unfortunately, this is an extremely weak statement. By the above argument, because *there exists* a polynomially-sized conjunctive grammar for  $P_n$ , there must be a polynomial-sized monotone circuit that works on all word-like inputs. And, indeed, there is. A boolean formula, even:  $\bigwedge_{i=1}^n \bigvee_{j=1}^n x_{i,j}$ . Informally, this formula checks whether each letter of the alphabet is present somewhere in the word. The uncanny resemblance between this formula and the original conjunctive grammar is not a coincidence.

Indeed, as mentioned in the previous Section, the result of evaluating  $f(K)$  on a word-like input corresponding to word  $w$ , is 1 when  $w \in K$  and 0 when  $w \notin K$ . So, if we want to prove that there is no conjunctive grammar for  $L_{\text{orig}}$ , we need to prove that the inclusion problem for  $L_n$  is harder time- or memory-wise than the cubic parsing algorithm that we used in the reduction! In the end, our best hope here is arriving to a *weaker* version of Theorem A in a *very* roundabout way. This is dissapointing, but not surprising. As I mentioned before, the Idea 1 works *exactly* because it expands the “working space” from only word-like inputs to a much richer set. So, if we are forced to shrink the space back, we are left with a simple restatement of Cocke–Younger–Kasami algorithm.

## 5 Conclusion

The above Sections show that the ideas behind CYKSZ algorithm are somewhat universal and can be applied in different and, sometimes, unexpected ways. Except for the above, I suggest another interesting direction of research. As Theorem 1 shows, the limited equivalence is in P for reasonable enough statement of the problem. However, the limited equivalence for ordinary grammars is in NP even when one of the grammars describes the whole language. The reason is the same as in proof of undecidability of ordinary grammar equivalence: ordinary grammars can encode computation histories of Turing machines. Hence, the existence of Theorem 1 suggest in some vague way that unambiguous grammars cannot encode computation histories of Turing machines. Therefore, it seems that we have some heuristic evidence that suggests that equivalence for unambiguous grammars is “much weaker” compared to the equivalence for arbitrary ordinary grammars.

## References

- [1] V. Arvind, P. Mukhopadhyay, S. Raja, “Randomized polynomial time identity testing for noncommutative circuits”, *STOC 2017: Proceedings of the 49th annual ACM symposium on Theory of computing*, 831–841.

- [2] J.-P. Allouche, J. Shallit, *Automatic Sequences: Theory, Applications, Generalizations*, Cambridge University Press, 2003.
- [3] J.-M. Autebert, J. Beauquier, L. Boasson, M. Nivat, “Quelques problèmes ouverts en théorie des langages algébriques”, *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, Volume 13, number 4 (1979), 363–378.
- [4] E. Bakinova, A. Basharin, I. Batmanov, K. Lyubort, A. Okhotin, E. Sazhneva, “Formal languages over  $\text{GF}(2)$ ”, *Language and Automata Theory and Applications (LATA 2018, Bar-Ilan near Tel Aviv, Israel, 9–11 April 2018)*, LNCS 10792, 68–79.
- [5] Y. Bar-Hillel, M. Perles, E. Shamir, “On formal properties of simple phrase-structure grammars”, *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14 (1961), 143–177.
- [6] V. S. Drensky, *Free algebras and PI-algebras: graduate course in algebra*, Springer-Verlag, 1996.
- [7] V. S. Drensky, “A minimal basis for identities of a second-order matrix algebra over a field of characteristic 0”, *Algebra Logika*, 20:3 (1981), 282–290.
- [8] S. Chien, P. Harsha, A. Sinclair, S. Srinivasan “Almost Settling the Hardness of Non-commutative Determinant” *STOC 2011: Proceedings of the 43rd annual ACM symposium on Theory of computing*, 499–508.
- [9] N. Chomsky, M. P. Schützenberger, “Algebraic theory of context-free languages”, *Studies in Logic and the Foundations of Mathematics*, 35 (1963), 118–161
- [10] K. Ellul, J. Shallit, M. Wang, “Regular Expressions: New Results and Open Problems”, *Descriptive Complexity of Formal Systems*, (London, Canada, 21–24 August 2002), 17:34
- [11] P. Flajolet, “Analytic methods and ambiguity of context-free languages”, *Theoretical Computer Science*, 49 (1987), 283–309
- [12] G. Christol, “Ensembles presque periodiques  $k$ -reconnaissables”, *Theoretical Computer Science*, 9 (1979), 141–145.
- [13] S. Ginsburg, H. G. Rice, “Two families of languages related to ALGOL”, *Journal of the ACM*, 9 (1962), 350–371.
- [14] S. Ginsburg, E. H. Spanier, “Bounded ALGOL-like languages”, *Transactions of the American Mathematical Society*, Volume 113, number 2 (1964), 333–368
- [15] S. Ginsburg, E. H. Spanier, “Semigroups, Presburger formulas, and languages”, *Pacific Journal of Mathematics*, Volume 16, number 2 (1966), 285–296

- [16] S. Ginsburg, J. Ullian, Ambiguity in context-free languages *Journal of the ACM*, 13 (1966), 62–89
- [17] T. N. Hibbard, J. Ullian, “The independence of inherent ambiguity from complementedness among context-free languages”, *Journal of the ACM*, 13 (1966), 588–593
- [18] P. Jancar, “Decidability of DPDA Language Equivalence via First-Order Grammars”, *Proceedings – Symposium on Logic in Computer Science*, (2012)
- [19] G. Jirásková, A. Okhotin, “Nondeterministic state complexity of positional addition”, *Journal of Automata, Languages and Combinatorics*, 15:1–2 (2010), 121–133.
- [20] H. Joos, R. Marie-Francoise, S. Pablo, “On the theoretical and practical complexity of the existential theory of reals”, *The Computer Journal*, 36:5 (1993), 427–431.
- [21] I. Katsányi, “Sets of integers in different number systems and the Chomsky hierarchy”, *Acta Cybernetica*, 15:2 (2001), 121–136.
- [22] V. Makarov, “Algebraic and analytic methods for grammar ambiguity”, *Research Repository Saint Petersburg State University*, June 2021
- [23] V. Makarov, A. Okhotin, “On the expressive power of GF(2)-grammars”, *SOFSEM 2019: Theory and Practice of Computer Science* (Nový Smokovec, Slovakia, 27-30 January 2019), LNCS 11376, 310–323.
- [24] A. Okhotin, E. Sazhneva, “State complexity of GF(2)-concatenation and GF(2)-inverse on unary languages”, *Descriptive Complexity of Formal Systems* (DCFS 2019, Kosice, Slovakia, 17-19 July 2019), to appear
- [25] A. A. Razborov, “Lower bounds on the Monotone Network Complexity of the Logical Permanent”, *Mat. Zametki*, 37 (1985), 887–900.
- [26] A. L. Semenov, “Algorithmic Problems for Power Series and Context-free Grammars”, *Soviet Mathematics*, 14 (1973), 1319
- [27] G. Sénizergues, “ $L(A) = L(B)$ ? decidability results from complete formal systems”, *Theoretical Computer Science*, 251:1–2 (2001), 1–166.
- [28] L. van Zijl, “On binary  $\oplus$ -NFAs and succinct descriptions of regular languages”, *Theoretical Computer Science*, 328:1–2 (2004), 161–170.