

Formal languages over $\text{GF}(2)$ [☆]

Ekaterina Bakinova^a, Artem Basharin^b, Igor Batmanov^c, Konstantin Lyubort^d, Alexander Okhotin^e,
Elizaveta Sazhneva^e

^a*Gymnasium №1, ul. Abelmana, 15, Kovrov 601900, Vladimir region, Russia*

^b*School №179, Bolshaya Dmitrovka, 5/6 building 7, Moscow 125009, Russia*

^c*Moscow Institute of Physics and Technology, Russia*

^d*St. Petersburg Academic University, ul. Khlopina, 8, Saint Petersburg 194021, Russia*

^e*St. Petersburg State University, 7/9 Universitetskaya nab., Saint Petersburg 199034, Russia*

Abstract

Variants of the union and concatenation operations on formal languages are investigated, in which Boolean logic in the definitions (that is, conjunction and disjunction) is replaced with the operations in the two-element field $\text{GF}(2)$ (conjunction and exclusive OR). Union is thus replaced with symmetric difference, whereas concatenation gives rise to a new $\text{GF}(2)$ -concatenation operation, which is notable for being invertible. All operations preserve regularity, and for a pair of languages recognized by an m -state and an n -state DFA, their $\text{GF}(2)$ -concatenation is recognized by a DFA with $m \cdot 2^n$ states, and this number of states is in the worst case necessary. Similarly, the state complexity of $\text{GF}(2)$ -inverse is $2^n + 1$. Next, a new class of formal grammars based on $\text{GF}(2)$ -operations is defined, and it is shown to have the same computational complexity as ordinary grammars with union and concatenation: in particular, simple parsing in time $O(n^3)$, fast parsing in the time of matrix multiplication, and parsing in NC^2 .

Key words: Formal languages, finite fields, finite automata, formal grammars, state complexity, computational complexity, parsing.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 2 |
| 2 | $\text{GF}(2)$-operations and their basic properties | 3 |
| 3 | $\text{GF}(2)$-operations on regular languages | 5 |
| 4 | Formal grammars with $\text{GF}(2)$-operations | 10 |
| 5 | Normal form and parsing for $\text{GF}(2)$-grammars | 13 |
| 6 | Complexity of $\text{GF}(2)$-grammars | 15 |
| 7 | Conclusion | 20 |

[☆]A preliminary version of this paper [2] was presented at the 12th International Computer Science Symposium in Russia (LATA 2018) held in Tel Aviv, Israel, on April 9–11, 2018.

Email addresses: katya-bakinova@mail.ru (Ekaterina Bakinova), artemvit@bk.ru (Artem Basharin), igorbat99@gmail.com (Igor Batmanov), lyubortk@gmail.com (Konstantin Lyubort), alexander.okhotin@spbu.ru (Alexander Okhotin), sazhneva-eliza@ya.ru (Elizaveta Sazhneva)

1. Introduction

The classical operations on formal languages are union and concatenation; both regular expressions and formal grammars are based on these operations. Union and concatenation are defined in terms of Boolean logic: the union $K \cup L$ is the set of all strings w with $w \in K$ or $w \in L$, which is a disjunction of two conditions; similarly, the membership of a string w in a concatenation $K \cdot L$ is a disjunction, over all partitions $w = uv$, of the conjunction of $u \in K$ and $v \in L$.

The purpose of this paper is to investigate a pair of related operations on languages defined using the *exclusive OR* instead of the disjunction, so that Boolean logic is effectively replaced with the two-element field $\text{GF}(2)$. Thus, the union operation turns into the symmetric difference, whereas concatenation gives rise to a new *GF(2)-concatenation* operation. A $\text{GF}(2)$ -concatenation of K and L , denoted by $K \odot L$, consists of all strings w that have an *odd number of partitions* $w = uv$, with $u \in K$ and $v \in L$: this is the result of applying the exclusive OR operation to these conjunctions.

The interest in these operations is motivated by the fact that they generalize the two unambiguous operations on formal languages: the *disjoint union* and the *unambiguous concatenation* [6], that is, concatenation $K \cdot L$ with the restriction that each string $w \in K \cdot L$ has a unique partition $w = uv$ with $u \in K$ and $v \in L$. Indeed, as long as two languages are disjoint, their symmetric difference equals their union, and if the concatenation of K and L is unambiguous, then their $\text{GF}(2)$ -concatenation $K \odot L$ coincides with the standard concatenation $K \cdot L$. The unambiguous operations are important, for instance, for being the only operations expressible in *unambiguous grammars*, as well as in their practically valuable subclasses with efficient parsing algorithms, such as the LL and the LR grammars. Accordingly, any model featuring the $\text{GF}(2)$ -operations has the corresponding unambiguous model as a special case. Thus, the base unambiguous model has two different extensions: one with classical operations (union and concatenation), and the other with $\text{GF}(2)$ -operations (symmetric difference and $\text{GF}(2)$ -concatenation).

The two proposed operations can be equally obtained by regarding K and L as formal power series in non-commuting variables with coefficients in $\text{GF}(2)$; then, their sum is $K \Delta L$ and their product is $K \odot L$. However, the authors' intention is to treat them as operations on standard formal languages, and to carry out a usual language-theoretic study of these operations.

The basic algebraic properties of $\text{GF}(2)$ -operations, listed in Section 2, follow from the known facts on formal power series: formal languages form a ring with the symmetric difference as addition and with $\text{GF}(2)$ -concatenation as multiplication. Furthermore, every language L containing the empty string is an *invertible element*, that is, there exists another language L^{-1} with $L \odot L^{-1} = L^{-1} \odot L = \{\varepsilon\}$. This gives a somewhat unexpected property of *inverting the concatenation*, and accordingly presents another operation to study: the *GF(2)-inverse*, $f(L) = L^{-1}$.

In Section 3, it is proved that the family of regular languages is closed under both $\text{GF}(2)$ -concatenation and $\text{GF}(2)$ -inverse. This is established by direct constructions on finite automata: for every DFA with m states and another DFA with n states, there is a DFA with $m \cdot 2^n$ states that recognizes their $\text{GF}(2)$ -concatenation. It is proved that this construction is in the worst case optimal with respect to the number of states. For every n -state DFA, its $\text{GF}(2)$ -inverse is recognized by a DFA with $2^n + 1$ states, and this upper bound is also proved to be exact.

The next subject is a new family of formal grammars based on $\text{GF}(2)$ -operations. The semantics of ordinary grammars with union and concatenation (Chomsky's "context-free") are defined by least solutions of language equations [8], owing to the monotonicity of both union and concatenation. Since $\text{GF}(2)$ -operations are not monotone, such a definition is not applicable to the desired new class of grammars. In Section 4, the new class of *GF(2)-grammars* is defined by imposing a restriction on a grammar, so that each string may have only finitely many parses. Under this restriction, the system of language equations always has a solution, making the definition both mathematically correct and intuitively clear.

The basic parsing algorithm for $\text{GF}(2)$ -grammars is defined in Section 5. First, it is proved that every such grammar can be transformed to an analogue of the Chomsky normal form. For a grammar in the normal form, there are simple variants of the Cocke–Kasami–Younger and Valiant's [28] parsing algorithms. The latter algorithm has subcubic complexity and is particularly efficient over $\text{GF}(2)$, because matrix multiplication over $\text{GF}(2)$ is easier than in the Boolean semiring.

In the last Section 6, the computational complexity of the new class of *GF(2)-grammars* is investigated. The NC^2 parallel parsing algorithm by Brent and Goldschlager [3] and by Rytter [24] is also adapted to handle $\text{GF}(2)$ -grammars, thus establishing the same complexity upper bound for these grammars as for the ordinary grammars with union and concatenation. The uniform membership problem for $\text{GF}(2)$ -grammars is shown to be P-complete, whereas for the subclass of *linear GF(2)-grammars*, this problem naturally turns out to be $\oplus\text{L}$ -complete.

2. GF(2)-operations and their basic properties

Two binary operations on formal languages are considered. the symmetric difference $K\Delta L$ and the $\text{GF}(2)$ -concatenation $K\odot L$. Unlike the union, the symmetric difference excludes the strings that belong to both languages at once.

$$\begin{aligned} K \cup L &= \{ w \mid w \text{ is in } K \text{ or in } L \} \\ K \Delta L &= \{ w \mid w \text{ is in } K \text{ or in } L, \text{ but not in both} \} \end{aligned}$$

The $\text{GF}(2)$ -concatenation is defined by replacing the condition on the *existence of a partition* in the classical concatenation with the condition that *the number of partitions must be odd*.

$$\begin{aligned} K \cdot L &= \{ w \mid \text{the number of partitions } w = uv, \text{ with } u \in K \text{ and } v \in L, \text{ is non-zero} \} \\ K \odot L &= \{ w \mid \text{the number of partitions } w = uv, \text{ with } u \in K \text{ and } v \in L, \text{ is odd} \} \end{aligned}$$

By definition, $K \odot L \subseteq K \cdot L$. Furthermore, if the concatenation $K \cdot L$ is known to be *unambiguous*, that is, if every string $w \in K \cdot L$ has a unique representation as $w = uv$ with $u \in K$ and $v \in L$, then these two languages coincide: $K \odot L = K \cdot L$. Thus, all interesting examples of $\text{GF}(2)$ -concatenation involve ambiguity.

Example 1. $\{\varepsilon, a\} \odot \{\varepsilon, a\} = \{\varepsilon, aa\}$. *The string a is missing from the $\text{GF}(2)$ -concatenation, because its two partitions cancel each other.*

Example 2. $a^* \odot a^* = (aa)^*$. *Indeed, each string a^n has $n + 1$ partitions, and thus all strings of odd length are cancelled out.*

Elementary algebraic properties of $\text{GF}(2)$ -concatenation and symmetric difference on formal languages can be inferred from the general results on the representation of languages as formal power series with coefficients from an arbitrary semiring, using $\text{GF}(2)$ as the semiring of coefficients. These properties are summarized below, and could be established directly, without referring to formal power series.

Proposition 1. *For every alphabet Σ , the set of all languages 2^{Σ^*} over this alphabet, forms a ring, with symmetric difference as sum and with $\text{GF}(2)$ -concatenation as product. Namely, the following properties hold.*

1. *Symmetric difference is associative: $K\Delta(L\Delta M) = (K\Delta L)\Delta M$.*
2. *Symmetric difference is commutative: $K\Delta L = L\Delta K$.*
3. *Symmetric difference has zero, which is the empty set: $L\Delta\emptyset = L$.*
4. *Under symmetric difference, every language L has an opposite language $-L$, which is the same language: $L\Delta L = \emptyset$.*
5. *$\text{GF}(2)$ -concatenation is associative: $K\odot(L\odot M) = (K\odot L)\odot M$.*
6. *$\text{GF}(2)$ -concatenation has an identity, which is a singleton empty string: $L\odot\{\varepsilon\} = \{\varepsilon\}\odot L = L$.*
7. *$\text{GF}(2)$ -concatenation is left- and right-distributive over symmetric difference: $K\odot(L\Delta M) = (K\odot L)\Delta(K\odot M)$, $(K\Delta L)\odot M = (K\odot M)\Delta(L\odot M)$.*

A further interesting property is that some languages have *inverses* with respect to $\text{GF}(2)$ -concatenation, that is, the product of two non-identity languages is the identity.

Example 3. $\{\varepsilon, ab\} \odot (ab)^* = \{\varepsilon\}$. The empty string has a unique partition as $\varepsilon \cdot \varepsilon$. Every string $(ab)^n$, with $n \geq 1$, can be represented both as $\varepsilon \cdot (ab)^n$ and as $ab \cdot (ab)^{n-1}$, and these two representations cancel each other.

There is no difference between left- and right-inverses: if L has both a left-inverse K , with $K \odot L = \{\varepsilon\}$, and a right-inverse M , with $L \odot M = \{\varepsilon\}$, then K and M must coincide (this property holds in every monoid).

No language L with $\varepsilon \notin L$ may have an inverse, because in this case $\varepsilon \notin K \odot L$ for every language K . On the other hand, every language containing the empty string has a GF(2)-inverse.

Theorem 1. For every language $L \subseteq \Sigma^*$ with $\varepsilon \in L$, there exists a unique language $L^{-1} \subseteq \Sigma^*$ that satisfies $L \odot L^{-1} = L^{-1} \odot L = \{\varepsilon\}$.

Although a direct proof can be given, this result is a direct adaptation of a known fact on formal power series—namely, that it is invertible if and only if its constant term is invertible in the semiring of coefficients.

Example 4. $\{\varepsilon, ab\}^{-1} = (ab)^*$ and, accordingly, $((ab)^*)^{-1} = \{\varepsilon, ab\}$.

Example 5. $(a^*b^*)^{-1} = \{\varepsilon, a, b, ba\}$.

The inverse can be equivalently represented as the following adaptation of the Kleene star.

Definition 1. For every language L , its GF(2)-star, denoted by L^\circledast , is the set of all strings w that have an odd number of representations of the form $w = w_1w_2 \dots w_k$, with $k \geq 0$ and $w_1, \dots, w_k \in L \setminus \{\varepsilon\}$.

The membership of a string in the GF(2)-star of a language is expressed through GF(2)-concatenation as follows.

Lemma 1. For every language L , a string w is in L^\circledast if and only if the number of representations $w = uv$, with $u \in L \setminus \{\varepsilon\}$ and $v \in L^\circledast$, is odd.

Proof. Let n_w be the number of representations of w as $w = w_1w_2 \dots w_k$, where $k \geq 0$, $w_1, w_2, \dots, w_k \neq \varepsilon$ and $w_1, w_2, \dots, w_k \in L$. For each $\ell \in \{1, \dots, k\}$, let $n_{w,\ell}$ be the number of such representations satisfying $|w_1| = \ell$. Then, $n_w = \sum_{\ell=1}^{|w|} n_{w,\ell}$. By Definition 1, $w \in L^\circledast$ if and only if n_w is odd, which holds if and only if the number of odd summands $n_{w,\ell}$ is odd. Each $n_{w,\ell}$ is odd if and only if the string w is representable as $w = uv$, with $|u| = \ell$, $u \in L$ and $v \in L^\circledast$. Therefore, the string w is in L^\circledast if and only if it has an odd number of representations $w = uv$, with $u \in L$, $u \neq \varepsilon$ and $v \in L^\circledast$. \square

Exactly the same representation holds for the GF(2)-inverse.

Lemma 2. For every language L with $\varepsilon \in L$, a non-empty string w is in L^{-1} if and only if the number of representations $w = uv$, with $u \in L \setminus \{\varepsilon\}$ and $v \in L^{-1}$, is odd.

Proof. The condition that the number of representations $w = uv$, where $u \in L \setminus \{\varepsilon\}$ and $v \in L^{-1}$, is odd, means exactly that the string w belongs to the GF(2)-concatenation $(L\Delta\{\varepsilon\}) \odot L^{-1}$. The latter expression is transformed as follows.

$$(L\Delta\{\varepsilon\}) \odot L^{-1} = (L \odot L^{-1})\Delta L^{-1} = \{\varepsilon\}\Delta L^{-1}$$

Since $w \neq \varepsilon$, the membership of w in this language is equivalent to $w \in L^{-1}$. \square

Theorem 2. For every language $L \subseteq \Sigma^*$ with $\varepsilon \in L$, the inverse L^{-1} equals the star L^\circledast .

Proof. It is claimed that if the sets L^{-1} and L^\circledast coincide on all strings of length less than n , then they also coincide on all strings of length n . This is true for strings of length zero, as $\varepsilon \in L^{-1}$ and $\varepsilon \in L^\circledast$.

Let w be a string of length n . Then, by Lemma 2, $w \in L^{-1}$ if and only if w has an odd number of representations as $w = uv$, with $u \in L$, $u \neq \varepsilon$ and $v \in L^{-1}$. Since $|v| < |w|$, by the induction hypothesis, $v \in L^{-1}$ if and only if $v \in L^\circledast$. Therefore, w has an odd number of representations $w = uv$, where $u \in L$, $u \neq \varepsilon$ and $v \in L^\circledast$, and, by Lemma 1, $w \in L^\circledast$. It follows that $w \in L^{-1}$ is equivalent to $w \in L^\circledast$. \square

3. GF(2)-operations on regular languages

For every operation on languages, the first basic question is whether it preserves the class of regular languages. If it does, the next question is its descriptive complexity, that is, how large an automaton is necessary to represent this operation on finite automata of a given size. The closure property holds both for GF(2)-concatenation and for GF(2)-inverse, and deterministic finite automata (DFA) implementing these operations, which are optimal with respect to the number of states, are constructed below.

Theorem 3. *Let $\mathcal{A} = (\Sigma, P, p_0, \eta, E)$ and $\mathcal{B} = (\Sigma, Q, q_0, \delta, F)$ be two DFA. Then the language $L(\mathcal{A}) \odot L(\mathcal{B})$ is recognized by a DFA \mathcal{C} with the set of states $P \times 2^Q$.*

Proof. The automaton \mathcal{C} should accept an input string w if and only if the number of partitions $w = uv$, with u accepted by \mathcal{A} and v accepted by \mathcal{B} , is odd. The automaton \mathcal{C} uses states of the form (p, S) , with $p \in P$ and $S \subseteq Q$. In the first component p , it simulates the computation of \mathcal{A} on the same input string, while S is the set of all states reached *an odd number of times* in the ongoing simulated computations of \mathcal{B} .

The initial state of \mathcal{C} depends on whether \mathcal{A} accepts the empty string. If it does, then a single computation of \mathcal{B} is started from the beginning, and the initial state of \mathcal{C} is $(p_0, \{q_0\})$. Otherwise, if \mathcal{A} does not accept ε , then nothing is started yet, and the initial state of \mathcal{C} shall be (p_0, \emptyset) .

When \mathcal{C} reads the next input symbol $a \in \Sigma$ in a state (p, S) , the simulated \mathcal{A} proceeds to the state $p' = \eta(p, a)$. Each ongoing computation of \mathcal{B} , currently in a state $q \in S$, proceeds to the next state $q' = \delta(q, a)$. However, if the same state q' is reached in this way from two different states in S , then, from this point on, these two computations are indistinguishable, and both either accept or reject; this means that they contribute an even number of partitions satisfying the definition of GF(2)-concatenation, and both can be cancelled out. The only states of \mathcal{B} that have to be remembered are those reachable by an odd number of computation paths. For this reason, the states of simulated computations of \mathcal{B} at the next step are

$$S' = \{q' \mid \text{the number of states } q \in S, \text{ with } q' = \delta(q, a), \text{ is odd}\}$$

Furthermore, if \mathcal{A} is in an accepting state, this indicates a possible partition of the input string at this point, and a new computation of \mathcal{B} on the remaining suffix beginning in the state q_0 should be started. However, if one of the ongoing computations of \mathcal{B} is already in the state q_0 , then these two states cancel each other. Overall, the transition of \mathcal{C} by a symbol a in a state (p, S) is defined as follows.

$$\pi((p, S), a) = \begin{cases} (\eta(p, a), S'), & \text{if } \eta(p, a) \notin E \\ (\eta(p, a), S' \triangle \{q_0\}), & \text{if } \eta(p, a) \in E \end{cases}$$

When \mathcal{C} finishes reading an input string w , it is in a state (p, S) , where p is the state of \mathcal{A} upon reading the same string w , while S is the set of all such states q of \mathcal{B} , that the number of partitions $w = uv$, with u accepted by \mathcal{A} , and with \mathcal{B} finishing reading v in the state q , is odd. Accordingly, a state (p, S) is marked as accepting, if S contains an odd number of accepting states of \mathcal{B} .

$$F' = \{(p, S) \mid |S \cap F| \text{ is odd}\}$$

This completes the construction. □

As shown in the next theorem, using all states in $P \times 2^Q$ is necessary in the worst case.

Theorem 4. *For every $m, n \geq 3$, there exist languages K and L over an alphabet $\{a, b\}$, recognized by an m -state DFA and by an n -state DFA, respectively, for which every DFA recognizing their GF(2)-concatenation $K \odot L$ must have at least $m \cdot 2^n$ states.*

Proof. The witness languages are defined by the following automata. For K , this is an m -state DFA $\mathcal{A} = (\Sigma, P, 0, \eta, E)$, with $P = \{0, \dots, m-1\}$ and $E = \{m-1\}$. It counts the number of symbols a modulo m , ignoring all symbols b .

$$\begin{aligned} \eta(i, a) &= i && \text{for all } i \in P \\ \eta(i, b) &= i + 1 \pmod{m} && \text{for all } i \in P \end{aligned}$$

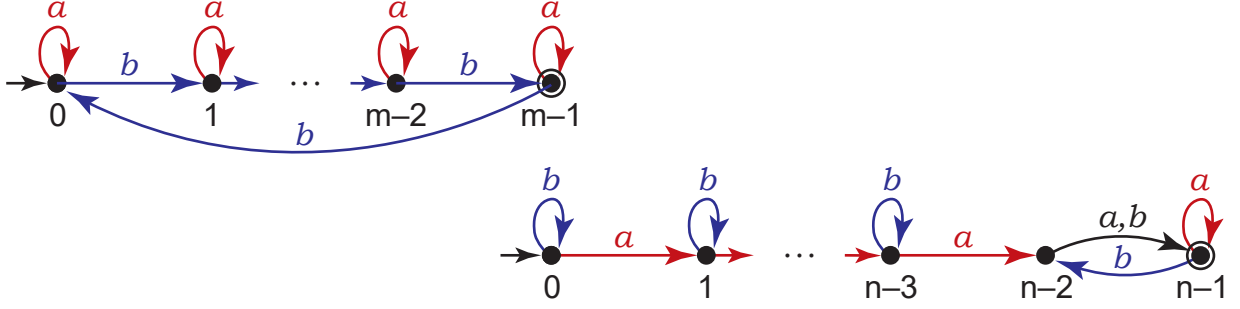


Figure 1: A pair of an m -state and an n -state DFA, for which the GF(2)-concatenation requires $m \cdot 2^n$ states.

The n -state DFA recognizing L is $\mathcal{B} = (\Sigma, Q, 0, \delta, F)$, with $Q = \{0, \dots, n-1\}$ and $E = \{n-1\}$, and with the following transitions.

$$\delta(i, a) = \begin{cases} i+1, & \text{if } 0 \leq i \leq n-2 \\ n-1, & \text{if } i = n-1 \end{cases}$$

$$\delta(i, b) = \begin{cases} i, & \text{if } 0 \leq i \leq n-3 \\ n-1, & \text{if } i = n-2 \\ n-2, & \text{if } i = n-1 \end{cases}$$

Let $\mathcal{C} = (\Sigma, Q', (0, \emptyset), \delta', F')$, with $Q' = \{(p, S) \mid p \in P, S \subseteq Q\}$, be the automaton defined in Theorem 3, which recognizes the GF(2)-concatenation of $L(\mathcal{A})$ and $L(\mathcal{B})$. Its set of accepting states is $F' = \{(p, S) \mid p \in P, n-1 \in S\}$.

It is claimed that every state $(p, S) \in Q'$ is reachable from the initial state $(0, \emptyset)$ by some string, and that every two states of \mathcal{C} are distinguished by a string that is accepted from one of them and not from the other. This shall confirm that every DFA recognizing the same language must have at least $m \cdot 2^n$ states.

The reachability is first proved for states of the following form.

Claim 1. *Every state of the form $(0, S)$, with $S \subseteq Q$ and $0 \notin S$, is reachable, that is, there is a string $u_{0,S} \in \Sigma^*$, with $\delta'((0, \emptyset), u_{0,S}) = (0, S)$.*

Let $S = \{i_1, \dots, i_k\}$, with $1 \leq i_1 < \dots < i_k \leq n-1$. It is claimed that the state $(0, S)$ is reached by the string $u_{0,S} = b^m a^{i_k - i_{k-1}} b^m \dots a^{i_2 - i_1} b^m a^{i_1}$.

The computation is made of substrings that operate as follows. First, for each state $(0, S)$, with $S \neq \emptyset$ and $0, n-2, n-1 \notin S$, by reading the string b^m , the automaton moves from the state $(0, S)$ to the state $(0, \{0\} \cup S)$, because, according to the transition function of \mathcal{B} , none of the states in S are affected, while the first component, by the transitions of \mathcal{A} , makes a full circle and finally contributes the state 0 to S . Second, for each state $(0, S)$ and for each number $j \geq 0$, with $\max S + j \leq n-1$, by reading the string a^j , the automaton moves from the state $(0, S)$ to the state $(0, \{i+j \mid i \in S\})$.

In this way, in the course of the computation on $u_{0,S}$, all states in S are added one by one.

Claim 2. *Each state (p, S) , with $p \neq 0$ and $0 \notin S$, is reachable by a string $u_{p,S} \in \Sigma^*$, with $\delta'((0, \emptyset), u_{p,S}) = (p, S)$.*

There are the following cases to consider.

Case I: either both $n-2$ and $n-1$ are in S , or none of them are. If $p \neq m-1$, then the state (p, S) is reachable from $(0, S)$ by the string b^p . If $p = m-1$, then $(m-1, S)$ is reachable from $(0, S)$ by b^{2m-1} .

Case II: one of $n-2$ and $n-1$ is in S , and the other is not.

If p is even and $p \neq m-1$, then (p, S) is reachable from $(0, S)$ by b^p .

If p is odd and $p \neq m-1$, then (p, S) is reachable by b^p from a state $(0, T)$, where $T = S \Delta \{n-2, n-1\}$.

If $p = m-1$, then (p, S) is reachable by b^{2m-1} from a state $(0, T)$, where $T = S \Delta \{n-2, n-1\}$.

All states with S containing the zero are reached analogously.

Claim 3. *Every state (p, S) , with $0 \in S$, is reachable by a string $u_{p,S} \in \Sigma^*$.*

Case I: either both $n-2$ and $n-1$ are in S , or none of them are. If $p \neq m-1$, then the state (p, S) is reachable from $(n-2, S \setminus \{0\})$ by the string b^{p+2} . If $p = m-1$, then $(m-1, S)$ is reachable from $(n-2, S \setminus \{0\})$ by b^1 .

Case II: one of $n-2$ and $n-1$ is in S , and the other is not.

If p is even and $p \neq m-1$, then (p, S) is reachable from $(0, S \setminus \{0\})$ by b^{p+2} .

If p is odd and $p \neq m-1$, then (p, S) is reachable by b^{p+2} from a state $(0, T)$, where $T = S \Delta \{0, n-2, n-1\}$.

If $p = m-1$, then (p, S) is reachable by b^1 from a state $(0, T)$, where $T = S \Delta \{0, n-2, n-1\}$.

This completes the reachability argument.

Claim 4. *All states are pairwise distinguishable, that is, for any two distinct states (p, S) and (p', S') in Q' , there is a string $w_{(p,S),(p',S')}$ that is accepted from one of these states and rejected from the other.*

Case I: $p = p'$ and $S \neq S'$.

Let $i = \max S \Delta S'$. Then the string a^{n-1-i} is accepted from exactly one of the states (p, S) and (p, S') . To be more precise, if the number of elements greater than i in S is even, then a^{n-1-i} is accepted from (p, S) and rejected from (p, S') , and if the number of elements of S exceeding i is odd, then the same string is rejected from (p, S) and accepted from (p, S') .

Case II: $p \neq m-1$, $p' = m-1$.

Upon reading the string a^{n-1} from the state (p, S) , the automaton enters either the rejecting state (p, \emptyset) , or the accepting state $(p, \{n-1\})$, and if it reads any further symbols a , it remains in the same state. If the same string a^{n-1} is read from the state $(m-1, S')$, the automaton may enter either the rejecting state $(m-1, \{0, 1, \dots, n-2\})$ or the accepting state $(m-1, \{0, 1, \dots, n-2, n-1\})$, and by reading any further symbols a the automaton alternates between these two states. Therefore, one of the strings a^{n-1} and a^n distinguishes between these two states.

Case III: $p < p' < m-1$.

Then, by the string $b^{m-1-p'}$, the automaton moves from (p, S) to some state (j, T) , with $j < m-1$, whereas from the state (p', S') it goes to some state $(m-1, T')$. As proved in the previous case, these two states have a distinguishing string. \square

\square

This establishes the precise state complexity of GF(2)-concatenation for DFA as $m \cdot 2^n$. To compare, the state complexity of the classical concatenation is $m2^n - 2^{n-1}$, as proved by Maslov [16], see also Yu et al. [29]. For the base operation, the unambiguous concatenation, Daley et al. [6] proved that its state complexity is $m2^{n-1} - 2^{n-2}$.

The number of states in an NFA representing GF(2)-concatenation of two NFA remains open: the only available construction is by determinizing the given automata and then applying Theorem 3, which yields an upper bound of $2^m \cdot 2^{2^n}$ states. On the other hand, for the class of *symmetric difference automata* (\oplus FA), studied by van Zijl [30], an automaton for GF(2)-concatenation can be naturally obtained by a series composition of two automata, using $m+n$ states.

The closure of the regular languages under the GF(2)-inverse and the GF(2)-star, is established by the following construction.

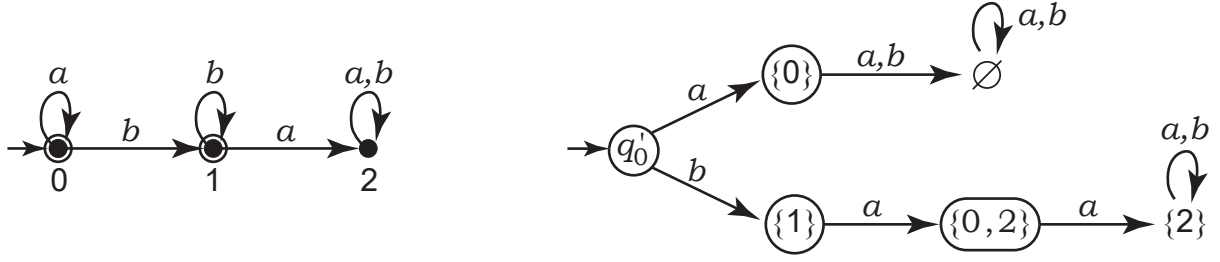


Figure 2: Example of construction according to Theorem 5: (left) the original DFA; (right) the resulting DFA.

Theorem 5. For every n -state DFA $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, the language $L(\mathcal{A})^\otimes$ is recognized by a DFA \mathcal{C} with the set of states $2^Q \cup \{q'_0\}$.

Proof. The construction relies on the representation of the GF(2)-star according to Lemma 1. Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$. The states of \mathcal{C} are all subsets of Q and a new initial state q'_0 , and $\mathcal{C} = (\Sigma, 2^Q \cup \{q'_0\}, q'_0, \delta', F')$.

The automaton \mathcal{C} should accept an input string w if and only if the number of partitions $w = u_1 \dots u_n$, with $u_i \in L(\mathcal{A})$, is odd. While considering different partitions of this form, \mathcal{C} simulates multiple copies of \mathcal{A} , storing the set of their states in its own state S , with $S \subseteq 2^Q$.

The automaton \mathcal{C} initially invokes one copy of \mathcal{A} that begins reading the input string w . Whenever the computations of \mathcal{A} being simulated visit accepting states, the automaton \mathcal{C} may start a new computation of \mathcal{A} on the suffix of the input string beginning at the current position. All these computations are simulated simultaneously, so that, after finishing reading the input string, the automaton could test whether the number of successful computations is odd.

The transition in the state q'_0 by each symbol $a \in \Sigma$ produces a singleton state corresponding to a single computation of \mathcal{A} .

$$\pi(q'_0, a) = \{\delta(q_0, a)\}$$

For every subset-state S , the transition by $a \in \Sigma$ simulates all transitions from all states $q \in S$, and cancels out all states reached an even number of times.

$$\pi(S, a) = \begin{cases} \{q \mid \text{the number of states } p \text{ with } \delta(p, a) = q \text{ is odd}\}, & \text{if } |S \cap F| \text{ is even} \\ \{q \mid \text{the number of states } p \text{ with } \delta(p, a) = q \text{ is odd}\} \Delta \delta(q_0, a), & \text{if } |S \cap F| \text{ is odd} \end{cases}$$

The initial state q'_0 is accepting, and a subset-state is accepting if it contains an odd number of accepting states of \mathcal{A} .

$$F' = \{S \mid |S \cap F| \text{ is odd}\} \cup \{q'_0\}$$

The correctness statement for the construction takes the following form. Let S be the state of \mathcal{C} after reading a string w . Then, S is the set of all states $q \in Q$, that the number of the following partitions is odd: these are partitions of w into $w = u_1 \dots u_k$, where u_1, \dots, u_k are all non-empty, each u_i , with $i \in \{1, \dots, k-1\}$, is accepted by \mathcal{A} , while the computation of \mathcal{A} on u_k beginning in the state q_0 finishes in the state q . The statement is established by induction on the length of the string, a formal proof is omitted. \square

Example 6 (continued from Example 5). The 3-state DFA in Figure 5(left) recognizes the language a^*b^* . The DFA obtained for this automaton according to Theorem 5 is presented in Figure 5(right), and it recognizes its GF(2)-inverse $\{\varepsilon, a, b, ba\}$.

This construction is optimal as well, and it is optimal even for the special case of GF(2)-inverses—that is, for languages containing the empty string.

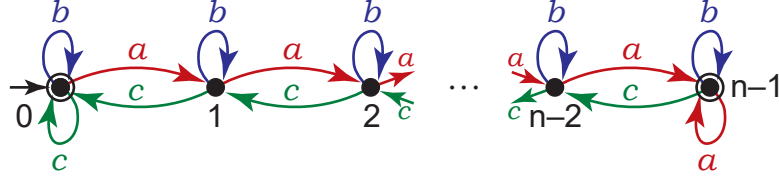


Figure 3: An n -state DFA, for which the $\text{GF}(2)$ -inverse requires $2^n + 1$ states.

Theorem 6. For every $n \geq 3$, there exist a language L over an alphabet $\Sigma = \{a, b, c\}$, with $\varepsilon \in L$, which is recognized by a DFA with n states, whereas every DFA recognizing its $\text{GF}(2)$ -inverse L^{-1} must have at least $2^n + 1$ states.

Proof. The witness language is given by an n -state DFA $\mathcal{A} = (\Sigma, Q, 0, \delta, F)$ with the set of states $Q = \{0, \dots, n-1\}$, where 0 is the initial state, and the accepting states are 0 and $n-1$. The transitions in each state $i \in Q$ are defined as follows.

$$\begin{aligned} \delta(a, i) &= \min(i+1, n-1) \\ \delta(b, i) &= i \\ \delta(c, i) &= \max(i-1, 0) \end{aligned}$$

The automaton is illustrated in Figure 3.

Let $\mathcal{C} = (\Sigma, 2^Q \cup \{q'_0\}, q'_0, \delta', F')$ be the DFA constructed for A by Theorem 5. It is claimed that every subset $S \subseteq Q$ is reachable from the initial state q'_0 by some string, and that for every two states of \mathcal{C} , there exists a string that is accepted from one of them and not from the other. This shall confirm that every DFA recognizing the same language must have at least $2^n + 1$ states.

Claim 5. Every state is reachable, that is, for each state $S \subseteq Q$ there is a string $u_S \in \Sigma^*$, with $\delta'(q'_0, u_S) = S$.

Case I: $0 \notin S$ and $n-1 \in S$. Let $S = \{i_1, i_2, \dots, i_k, n-1\}$, with $i_1 > 0$. Then the state S is reached by a string $u_S = a^{n-1}(ba)(baca)^{i_k-i_{k-1}-1}(ba) \dots (baca)^{i_2-i_1-1}ba(baca)^{i_1-1}$

Case II: $0 \in S$ and $n-1 \in S$. Then the state $S = \{0, i_1, \dots, n-1\}$ is reached from the state $S \setminus \{0\} = \{i_1, \dots, n-1\}$ by b , since $\delta'(T, b) = \bigcup_{i \in T} \delta'(i, b) = \bigcup_{i \in T} \delta(i, b) \Delta \delta(0, b) = S$. The state $S \setminus \{0\}$ is reachable by Case I.

Case III: $\max S < n-1$. All states containing $n-1$ have now been proved reachable, and it remains to reach all states with a smaller maximum element. The proof is by induction on $k = \max S$. The base case, $k = n-1$, has been established above. For the induction step, if all states with the maximum element k are reachable, then each state $S = \{i_1, i_2 \dots k-1\}$ with the maximum element $k-1$ is reached from $T = \{i_1+1, i_2+1 \dots k\}$ by c .

For the pairwise distinguishability of states, first, there is a special case of distinguishing the initial state from every subset-state.

Claim 6. The state q'_0 is distinguishable from each state $S \subseteq Q$, that is, there is a string $w_{q'_0, S}$ that is accepted from one of q'_0, S and rejected from the other.

1. If $S \subseteq \{0, 1\}$, then b is accepted from q'_0 , but not from S , since $\delta'(\{0\}, b) = \emptyset$; $\delta'(\{0, 1\}, b) = \{1\}$; $\delta'(\{1\}, b) = \{1\}$; $\delta'(q'_0, b) = \{0\}$.
2. If $S \not\subseteq \{0, 1\}$, let $i = \max S$; it is known that $i \geq 2$. Then the string c^i is accepted from S , but not from q'_0 , as $\delta'(S, c^i) = \{0\}$ and $\delta'(q'_0, c^i) = \emptyset$.

Claim 7. All subset-states are pairwise distinguishable, that is, for any two distinct states $S, S' \subseteq Q$, there is a string $w_{S, S'}$ that is accepted from one of S, S' and rejected from the other.

| | Sum | | Concatenation | | Star | |
|-------------|--------------|---------------|------------------|---------------------------------|-------------------------|-----------------------------|
| Unambiguous | (\uplus) | $mn - 1$ [14] | (UNAMB \cdot) | $m2^{n-1} - 2^{n-2}$ [6] | (UNAMB*) | $\frac{3}{2}2^n + 1$ [14] |
| Classical | (\cup) | mn [16] | (\cdot) | $m2^n - 2^{n-1}$ [16] | (*) | $\frac{3}{4}2^n$ [16] |
| GF(2) | (Δ) | mn [4] | (\odot) | $\mathbf{m} \cdot \mathbf{2}^n$ | (\otimes / $^{-1}$) | $\mathbf{2}^n + \mathbf{1}$ |

Table 1: State complexity of unambiguous, classical and GF(2)-variants of sum, concatenation and star.

Case I: \emptyset and S . Since $S \neq \emptyset$, let $i = \max S$. Then the string c^i is accepted from S , but not from \emptyset .

Case II: $S, S' \neq \emptyset, S \neq S'$. Let $i = \max S \Delta S'$, and assume, without loss of generality, that $i \in S$ and $i \notin S'$. Then the string c^i is accepted from S and rejected from S' .

□

The state complexity of unambiguous, classical and GF(2)-variants of the three main operations on formal languages is compared in Table 1. All results refer to DFA.

For NFA, the GF(2)-star is representable using $2^{2^n} + 1$ states. On the other hand, for \oplus FA, the same operation can be naturally represented using $n + 1$ states.

Calculations carried out by the authors suggest that the GF(2)-inverse over the unary alphabet apparently requires DFA with $2^{n-1} + 1$ states, with the worst-case examples producing DFA with period $2^{n-1} - 1$.

4. Formal grammars with GF(2)-operations

In the ordinary kind of formal grammars, called “context-free grammars” in Chomsky’s tradition, the available operations are union and concatenation. Other grammar families, such as linear grammars or conjunctive grammars [18], differ from the ordinary grammars in the sets of operations allowed in the rules: in linear grammars, the operations include concatenation with a single symbol on either side, as well as union, whereas in conjunctive grammars, the operations are union, intersection and concatenation. This paper initiates the study of a new model, the *GF(2)-grammars*, with the operations of *symmetric difference* and *GF(2)-concatenation*.

One should note that none of the aforementioned grammar families involves any context dependencies, all of them are in a certain sense “context-free”, and differ only in the set of allowed operations. Hence, grammars with union and concatenation (those that Chomsky named “context-free”) shall be referred to as *ordinary grammars* [20], to distinguish them from the GF(2)-grammars, which are defined in almost the same way.

Definition 2. A *GF(2)-grammar* is a quadruple $G = (\Sigma, N, R, S)$, in which:

- Σ is the alphabet of the language;
- N is the set of nonterminal symbols, each representing a syntactic category defined in the grammar, that is, a property that each string may have or not have;
- every rule in R is of the form $A \rightarrow X_1 \odot \dots \odot X_\ell$, with $\ell \geq 0$ and $X_1, \dots, X_\ell \in \Sigma \cup N$, which represents all strings that have an odd number of partitions into $w_1 \dots w_\ell$, with each w_i representable as X_i ;
- the initial symbol $S \in N$ stands for the syntactic category of all well-formed sentences in the language.

A grammar is *linear GF(2)*, if, in each rule, at most one of X_1, \dots, X_ℓ is a nonterminal symbol.

Whenever there are multiple rules for some nonterminal symbol, there is an implicit *exclusive OR* connective between them, similar to the implicit disjunction in ordinary grammars. Accordingly, these rules can be denoted using the sum modulo 2 operator (\oplus), so that two rules, $A \rightarrow B \odot C$ and $A \rightarrow D \odot E$, can be written down as $A \rightarrow (B \odot C) \oplus (D \odot E)$.

The general plan is to define the language described by a grammar, so that it is a solution of the corresponding system of language equations in variables N , where the *sum modulo 2* operator is implemented as the symmetric difference of languages. For each variable, the system has the following equation with this variable on the left-hand side and an expression representing all its rules on the right-hand side.

$$A = \bigtriangleup_{A \rightarrow X_1 \odot \dots \odot X_\ell \in R} X_1 \odot \dots \odot X_\ell \quad (A \in N) \quad (*)$$

For each X_i that is a symbol, $X_i = a \in \Sigma$, it denotes the singleton language $\{a\}$. This is the same kind of system as for ordinary grammars [8], only the operations have changed.

However, changing the operations leads to certain complications: in particular, in some cases, this system has no solutions. For instance, such is the system corresponding to the grammar with two rules, $S \rightarrow S \odot S$ and $S \rightarrow \varepsilon$. The general theory of weighted grammars with weights from a semiring—see, for instance, a survey by Petre and Salomaa [22]—is not applicable here, because it relies upon certain monotonicity and continuity conditions, which do not hold for coefficients in $\text{GF}(2)$. The given example of a grammar has no intuitive meaning either, and the natural solution is to eliminate these degenerate cases from the very beginning. For this reason, the proposed definition imposes a restriction upon a grammar: the number of parse trees for every string must be finite.

Definition 3. For each $\text{GF}(2)$ -grammar $G = (\Sigma, N, R, S)$, let $\widehat{G} = (\Sigma, N, \widehat{R}, S)$ be the corresponding ordinary grammar with \widehat{R} containing a rule $A \rightarrow X_1 \dots X_\ell$ for each rule $A \rightarrow X_1 \odot \dots \odot X_\ell$ in R . Assuming that, for all $A \in N$ and $w \in \Sigma^*$, the number of parse trees of w as A in \widehat{G} is finite, the language $L_G(A)$ is defined as the set of all strings w with an odd number of parse trees as A in \widehat{G} . If the finiteness condition does not hold, then G is considered ill-formed.

The languages thus defined satisfy the system of language equations, and hence implement the desired definition.

Proposition 2. Let $G = (\Sigma, N, R, S)$ be a $\text{GF}(2)$ -grammar that satisfies the condition in Definition 3. Then the substitution $A = L_G(A)$ for all $A \in N$ is a solution of the system (*).

The following example of a grammar is given to illustrate their intuitive meaning and their formal definition.

Example 7. The $\text{GF}(2)$ -grammar given below describes the language $\{ab, abb\}$, because the $\text{GF}(2)$ -concatenation in the rule for S contains two strings with a unique partition (ab, abb) , and excludes the string abb with two partitions.

$$\begin{aligned} S &\rightarrow A \odot B \\ A &\rightarrow a \oplus ab \\ B &\rightarrow b \oplus bb \end{aligned}$$

Another way of seeing this is to observe that the corresponding ordinary grammar given below defines four parse trees: one for ab , two for abb and one for abb , shown in Figure 4. The two parse trees for abb cancel each other.

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \mid ab \\ B &\rightarrow b \mid bb \end{aligned}$$

What kind of languages can be described by $\text{GF}(2)$ -grammars? First of all, if an unambiguous grammar is transcribed as a $\text{GF}(2)$ -grammar, it still defines the same language, because each string has either a unique parse tree or none at all. On the other hand, taking any ambiguous grammar and reinterpreting it as a $\text{GF}(2)$ -grammar leads to some simple non-trivial examples of these grammars.

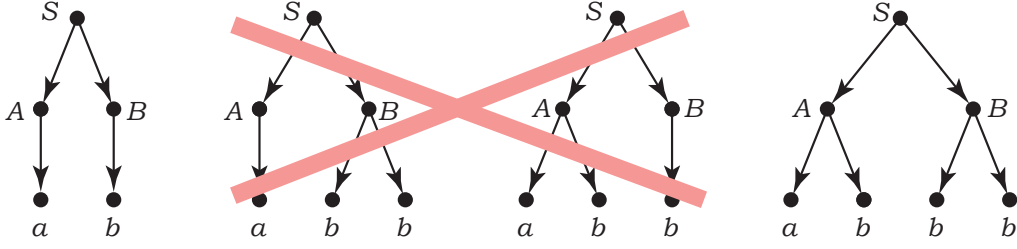


Figure 4: Four parse trees in the grammar in Example 7.

Example 8. *The linear GF(2)-grammar given below describes the language $\{a^\ell b^m c^n \mid \ell = m \text{ or } m = n, \text{ but not both}\}$.*

$$\begin{aligned}
S &\rightarrow A \oplus C \\
A &\rightarrow aA \oplus B \\
B &\rightarrow bBc \oplus \varepsilon \\
C &\rightarrow Cc \oplus D \\
D &\rightarrow aDb \oplus \varepsilon
\end{aligned}$$

Here, as well as in all later examples, the GF(2)-concatenation with fixed strings u and v is denoted simply by uLv , owing to the fact that $\{u\} \cdot L \cdot \{v\} = \{u\} \odot L \odot \{v\}$ for every language L .

This language is not representable by any ordinary grammar, which can be proved by a standard argument based on Ogden's lemma. Although it is natural to expect that the language $\{a^\ell b^m c^n \mid \ell = m \text{ or } m = n\}$, which is described by a similarly defined ordinary grammar, cannot be defined by any GF(2)-grammar, it is unclear whether this is indeed true, and how such statements could be proved.

Example 9. *The following linear GF(2)-grammar describes the language $\{a^m b^n \mid \binom{m+n}{n} \text{ is odd}\}$.*

$$S \rightarrow aS \oplus Sb \oplus ab$$

Indeed, the number of parse trees of each string $a^m b^n$ in the grammar $S \rightarrow aS \mid Sb \mid ab$ is $\binom{m+n}{n}$.

The next example is a language related to the standard example $\{uw \mid w \in \{a,b\}^*\}$. The complement of the latter language to the set of even-length strings is the language $\{uv \mid u, v \in \{a,b\}^*, |u| = |v|, u \text{ and } v \text{ differ in at least one position}\}$, and an inherently ambiguous ordinary grammar describing this language is known. Once that grammar is reformulated as a GF(2)-grammar, the following language is obtained.

Example 10. *The following GF(2)-grammar describes the language $\{uv \mid u, v \in \{a,b\}^*, |u| = |v|, u \text{ and } v \text{ differ in an odd number of positions}\}$.*

$$\begin{aligned}
S &\rightarrow (A \odot B) \oplus (B \odot A) \\
A &\rightarrow aAa \oplus aAb \oplus bAa \oplus bAb \oplus a \\
B &\rightarrow aBa \oplus aBb \oplus bBa \oplus bBb \oplus b
\end{aligned}$$

In particular, it is not known whether conjunctive grammars can describe any languages of this kind [18, Prob. 6].

The next example of a GF(2)-grammar describes a non-regular language over a unary alphabet. The first example of a language equation for a non-regular unary language was given by Leiss [15], who used an equation of the form $X = \varphi(X)$, with φ using the operations of concatenation and complementation. For

conjunctive grammars, that is, for language equations with union, intersection and concatenation, the first example of a representable unary language was discovered by Jež [10], and, based on his ideas, some fairly sophisticated unary languages were represented later [11, 12, 21].

For GF(2)-grammars, at the moment, there is only a small example of a grammar for the language $\{a^{2^n} \mid n \geq 0\}$. The grammar is based on an interesting fact that, over the unary alphabet, the GF(2)-square $L \odot L$ doubles the length of each string.

Lemma 3. *Let $L \subseteq a^*$ be a unary language. Then $L \odot L = \{a^{2^n} \mid a^n \in L\}$.*

Proof. Indeed, for a string a^ℓ , where ℓ is not twice the length of any string in L , each partition $a^\ell = a^i \cdot a^{\ell-i}$, with $a^i, a^{\ell-i} \in L$, has a symmetric partition $a^\ell = a^{\ell-i} \cdot a^i$, and therefore the number of such partitions is always even. On the other hand, a string a^{2^n} , with $a^n \in L$, has one more partition $a^{2^n} = a^n \cdot a^n$, which is symmetric to itself, making the total number of partitions odd. \square

Example 11. *The following grammar describes the language $\{a^{2^n} \mid n \geq 0\}$.*

$$S \rightarrow (S \odot S) \oplus a$$

Since the square $S \odot S$ doubles the length of each string, this equation is equivalent to the following one.

$$S = \{a^{2^n} \mid a^n \in S, n \geq 1\} \cup \{a\}$$

Its unique solution is, obviously, the desired language.

Alternatively, one can verify that the grammar describes the desired language directly by the definition: the number of parse trees of a string a^ℓ in the ordinary grammar $S \rightarrow SS \mid a$ is the $(\ell - 1)$ -th Catalan number, which is odd if and only if ℓ is a power of 2.

The latter example indicates that GF(2)-grammars are not in all respects symmetric to ordinary grammars with concatenation and disjunction: whereas ordinary grammars over a unary alphabet can define only regular languages, GF(2)-grammars are strictly more powerful in this case.

It follows from Christol's [5] theorem that a unary language $L \subseteq a^*$ is described by a unary GF(2)-grammar if and only if it is a *2-automatic set*, that is, the set of binary representations of the lengths of strings in L is a regular language over $\{0, 1\}$. In particular, the language $\{a^{3^n} \mid n \geq 0\}$ is not described by any GF(2)-grammar. This shows that, over a unary alphabet, the expressive power of GF(2)-grammars is strictly between ordinary grammars and unambiguous conjunctive grammars [13].

5. Normal form and parsing for GF(2)-grammars

In order to develop parsing algorithms for GF(2)-grammars and to assess their computational complexity, it is convenient to obtain a normal form for these grammars first. The following adaptation of the Chomsky normal form shall be established.

Theorem 7. *Every GF(2)-grammar can be effectively transformed in polynomial time to a GF(2)-grammar that describes the same language, and has all rules of the form $A \rightarrow B \odot C$, with $B, C \in N$, or $A \rightarrow a$, with $a \in \Sigma$.*

Notably, every grammar in the normal form satisfies the condition in Definition 3, because the height of parse trees is bounded by the length of the string.

The transformation follows the standard procedure. First, long rules are cut into rules with at most two symbols on the right-hand side, Next, *null rules* of the form $A \rightarrow \varepsilon$ are eliminated. The next step is the elimination of *unit rules* of the form $A \rightarrow B$, with $B \in N$. Finally, all occurrences of terminal symbols on the right-hand sides are moved to separate rules.

The elimination of null rules begins with determining all nonterminals that define the empty string.

Lemma 4. For every ordinary grammar $G = (\Sigma, N, R, S)$ with finitely many parse trees of ε from each $A \in N$, the set \oplus -NULLABLE = $\{A \mid \text{the number of parses of } \varepsilon \text{ from } A \text{ is odd}\}$ can be constructed in polynomial time.

Sketch of a proof. Since the number of parse trees is finite, the set \oplus -NULLABLE is well-defined, and the only question is how to construct it efficiently.

The method of constructing the set \oplus -NULLABLE of all nonterminals that have an odd number of parse trees of ε is based on the following observation: as long as the number of parse trees of ε from A is finite, no path in this tree may contain multiple copies of the same nonterminal symbol B . Indeed, if there is such a cyclic segment in one of the paths, then this segment could be reduplicated, as in the pumping lemma, thus obtaining infinitely many parse trees of ε . This implies that, for any two distinct nonterminals A and B , it cannot be the case that B occurs in some parse tree of ε from A , and at the same time A occurs in some parse tree of ε from B . This induces a *partial order of presence in each other's parse trees of ε* on the set of nonterminals, with $A \succ B$ if there is a parse tree of ε from A that contains B .

In order to compute this partial order, one can begin with computing the classical set NULLABLE, which consists of all nonterminals that have at least one parse tree of ε . Then, whenever there is a rule $A \rightarrow \theta B \theta'$, with $\theta, \theta' \in \text{NULLABLE}^*$ and $B \in N$, one can infer that $A \succ B$. It remains to compute the transitive closure of this relation.

Then, in order to determine the number of parse trees of ε from each nonterminal A , it is sufficient to calculate it in this order, from lesser to greater elements. \square

Using this set, the null rules are removed by the standard construction, modified to use parity instead of existence.

Lemma 5. For every GF(2) grammar $G = (\Sigma, N, R, S)$, let $G' = (\Sigma, N, R', S)$ be another GF(2) grammar that contains a rule $A \rightarrow X_1 \odot \dots \odot X_\ell$, if $\ell \geq 1$ and the number of rules $A \rightarrow \theta_0 \odot X_1 \odot \theta_1 \odot \dots \odot X_\ell \odot \theta_\ell$ in R , with each θ_i being a GF(2)-concatenation of zero or more nonterminals in \oplus -NULLABLE, is odd. Then, in the new grammar, every $A \in N$ defines the language $L_{G'}(A) = L_G(A) \setminus \{\varepsilon\}$.

Sketch of a proof. Let w be a non-empty string. For each parse tree of w from A in G , let $A \rightarrow \theta_0 \odot X_1 \odot \theta_1 \odot \dots \odot X_\ell \odot \theta_\ell$ be the rule in its root, where X_1, \dots, X_ℓ are all its children with non-empty substrings in their subtrees, and $\theta_0, \dots, \theta_\ell \in N^*$. If at least one of θ_i is not in \oplus -NULLABLE, then there is an even number of such trees, and they do not affect the total number of parse trees of w . If, $\theta_0, \dots, \theta_\ell \in \oplus$ -NULLABLE*, then the number of these trees is given by the following expression, where $n(u, X)$ denotes the number of parse trees of u as X .

$$\sum_{w=w_1 \dots w_\ell} \prod_{i=1}^{\ell} n(w_i, X_i)$$

In the new grammar G' , the parity of $n(w_i, X_i)$ is preserved, and the rule $A \rightarrow X_1 \odot \dots \odot X_\ell$ defines all the same parse trees of w . \square

There is also a similar ‘‘parity’’ version of the classical unit rules elimination.

Lemma 6. For every GF(2) grammar $G = (\Sigma, N, R, S)$ with no rules of the form $A \rightarrow \varepsilon$, let $G' = (\Sigma, N, R', S)$ be another GF(2) grammar that contains a rule $A \rightarrow X_1 \odot \dots \odot X_\ell$, with $\ell \geq 2$ or $X_1 \in \Sigma$, if there is an odd number of chains of the form $A \rightarrow B_1, B_1 \rightarrow B_2, \dots, B_{n-1} \rightarrow B_n, B_n \rightarrow X_1 \odot \dots \odot X_\ell$. Then, for every $A \in N$, $L_{G'}(A) = L_G(A)$.

Sketch of a proof. All chains of this form are of bounded length, because the original grammar satisfies the condition in Definition 3. Thus, the parity of this number is well-defined.

The parse trees in the new grammar are obtained by condensing all chains in parse trees of the original grammar. Furthermore, the condition on the parity of the number of chains ensures that a condensed parse tree exists in the new grammar if and only if the number of different expanded parse trees in the original grammar is odd. \square

The proof of Theorem 7 follows through the three lemmata above.

With the normal form theorem established, an adaptation of the Cocke–Kasami–Younger algorithm running in cubic time follows immediately. The fact that this algorithm can be employed to compute weights over an arbitrary semiring is folklore, and for GF(2) it is stated as follows.

Let $G = (\Sigma, N, R, S)$ be a GF(2)-grammar in the normal form. Given an input string $w = a_1 \dots a_n$, the algorithm constructs the following sets inductively on the length of substring.

$$T_{i,j} = \{ A \in N \mid a_{i+1} \dots a_j \in L_G(A) \} \quad (\text{for } 0 \leq i < j \leq n)$$

For each one-symbol substring, the set is $T_{j-1,j} = \{ A \mid A \rightarrow a_j \in R \}$, and for every longer substring it is determined by the following expression.

$$P_{i,j} = \bigtriangleup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$$

$$T_{i,j} = \{ A \mid \text{the number of rules } A \rightarrow BC, \text{ with } (B, C) \in P_{i,j}, \text{ is odd} \}$$

Overall, there are $O(n^2)$ elements in the table, each of them is calculated in time $O(n)$, and thus the algorithm works in time $O(n^3)$.

Using Valiant's [28] algorithm, the same table $T_{i,j}$ can be constructed in time $O(n^\omega)$, where $O(n^\omega)$ is an upper bound on the number of operations needed to multiply a pair of $n \times n$ matrices in GF(2).

Theorem 8. *Every language described by a GF(2)-grammar is in DTIME(n^ω).*

In fact, practical algorithms for matrix multiplication over GF(2) are substantially faster than those for Boolean matrix multiplication: for smaller matrices there is a particularly efficient implementation of the Four Russians method [1], and for larger matrices one can apply Strassen's algorithm directly in GF(2). This makes the algorithm even more efficient than the parsing algorithms for ordinary grammars.

6. Complexity of GF(2)-grammars

In order to assess the practical value of GF(2)-grammars, it is important to know the computational complexity of their main decision problems.

The first natural decision problem to be investigated is whether a given grammar satisfies the condition on the finiteness of the number of parse trees.

Lemma 7. *Let $G = (\Sigma, N, R, S)$ be an ordinary grammar, in which every nonterminal symbol is used in some parse tree. Then, a string with infinitely many parse trees exists if and only if there is a sequence of rules $A_1 \rightarrow \eta_1 A_2 \theta_1$, $A_2 \rightarrow \eta_2 A_3 \theta_2$, \dots , $A_k \rightarrow \eta_k A_1 \theta_k$, for some $k \geq 1$, $A_1, \dots, A_k \in N$, $\eta_i, \theta_i \in \text{NULLABLE}^*$.*

Proof. \ominus Assume that such a sequence exists. It is known that there exists a parse tree of some string w containing A_1 . Then one can pump this parse tree by replacing A_1 with a path of length k formed by this sequence of rules, with subtrees deriving ε from each η_i and θ_i spawned off this path. The resulting larger parse tree still describes the same string w , and repeating this procedure produces infinitely many parse trees of w .

$\omin�$ Let w be a string with infinitely many parse trees. Then there exists a parse tree of height at least $(|w| + 1) \cdot (|N| + 1)$. The longest path in this tree can be split into $|w| + 1$ sections of length $|N| + 1$ each. Since there are $|w| + 1$ sections in the path, and only $|w|$ leaves labelled with symbols in the entire tree, there is a section, in which no leaves are spawned off in either direction. The section itself is of length $|N| + 1$, and therefore there is a repeated nonterminal symbol A in this section. The sequence of rules between the two instances of A is as desired. \square

This characterization yields a polynomial-time algorithm for testing the condition in Definition 3 for a given grammar.

Theorem 9. *The problem of testing, for a given ordinary grammar $G = (\Sigma, N, R, S)$, whether every string has finitely many parses, is P-complete.*

Proof. An algorithm for testing this condition works as follows. First, the set NULLABLE is constructed. Then the condition in Lemma 7 is just a reachability problem in a graph. Each step can be carried out in polynomial time.

The P-hardness of this problem is proved by the classical reduction from the Monotone Circuit Value Problem by Golschlager [9]. A circuit with gates C_0, \dots, C_n contains conjunction gates of the form $C_i = C_j \wedge C_k$, with $j, k < i$, disjunction gates $C_i = C_j \vee C_k$, with $j, k < i$, and constants $C_i = 0$ and $C_i = 1$. In the reduction, the circuit is transformed to an ordinary grammar $G = (\{a\}, \{C_0, \dots, C_n, S\}, R, S)$, with each gate represented by a nonterminal symbol that should define the empty string if and only if the gate evaluates to 1. The grammar has a rule $C_i \rightarrow \varepsilon$ for constant 1, no rules for constant 0, a rule $C_i \rightarrow C_j C_k$ for a conjunction gate, and two rules $C_i \rightarrow C_j \mid C_k$ for a disjunction gate. Finally, the rules for the initial symbol are $S \rightarrow SS \mid C_n$.

Now, if the circuit evaluates to 1, then $L_G(C_n) = \{\varepsilon\}$, and there are infinitely many parse trees of ε , and if the value of the circuit is 0, then $L_G(C_n) = \emptyset$, and accordingly $L(G) = \emptyset$. \square

The next question concerns the complexity of the uniform membership problem, where both the string and the grammar are given.

Theorem 10. *The uniform membership problem for GF(2)-grammars, stated as “Given a GF(2) grammar G and a string w , determine whether w is in $L(G)$ ”, is P-complete.*

Sketch of a proof. The problem is solved in polynomial time by first transforming the given grammar to the normal form and then applying the cubic-time parsing algorithm. For a grammar G , this can be implemented in time $|G|^2 \cdot |w|^3$.

The P-hardness is proved by another reduction from the Circuit Value Problem (CVP). This time, let a circuit with gates C_0, \dots, C_n have conjunction gates $C_i = C_j \wedge C_k$, with $j, k < i$, negation gates $C_i = \neg C_j$, with $j < i$, and constants $C_i = 0$ and $C_i = 1$. In the corresponding GF(2)-grammar, each gate is represented by a nonterminal symbol that should define ε if and only if the gate has value 1. Again, constant 1 is represented by a rule $C_i \rightarrow \varepsilon$, there are no rules for constant 0, a conjunction gate turns into $C_i \rightarrow C_j C_k$, whereas a negation gate has two rules, $C_i \rightarrow C_j \oplus \varepsilon$. The last gate C_n is the initial symbol of the grammar.

The number of parse trees of ε at every next nonterminal symbol is at most the number at the previous nonterminal symbol, squared. Accordingly, the total number of parse trees of ε is finite, and the grammar is well-formed.

The system of language equations corresponding to the grammar directly expresses the desired Boolean equations for the values in the gates. Therefore, it defines the empty string if and only if the circuit evaluates to 1. \square

Thus, the complexity of the uniform membership problem for GF(2)-grammars is the same as for ordinary grammars and for conjunctive grammars.

For the fixed membership problem, which is in NC² for ordinary grammars and P-complete for conjunctive grammars, GF(2)-grammars have it in NC². This practically very important property is established by a variant of the algorithm for ordinary grammars independently discovered by Brent and Golschlager [3] and by Rytter [24]. This algorithm requires more substantial changes than in the algorithms considered so far. The original Brent–Golschlager–Rytter $(\log n)^2$ -time parallel parsing algorithm determines the *existence* of parse trees, and while doing so, it may consider the same tree an unspecified number of times. In the algorithm for GF(2)-grammars, which has to test that the number of trees is *odd*, it is imperative that every subtree is considered exactly once. In a different context, Rossmanith and Rytter [23] ensured this property in a similar algorithm for unambiguous grammars. The following algorithm does this without the unambiguity assumption.

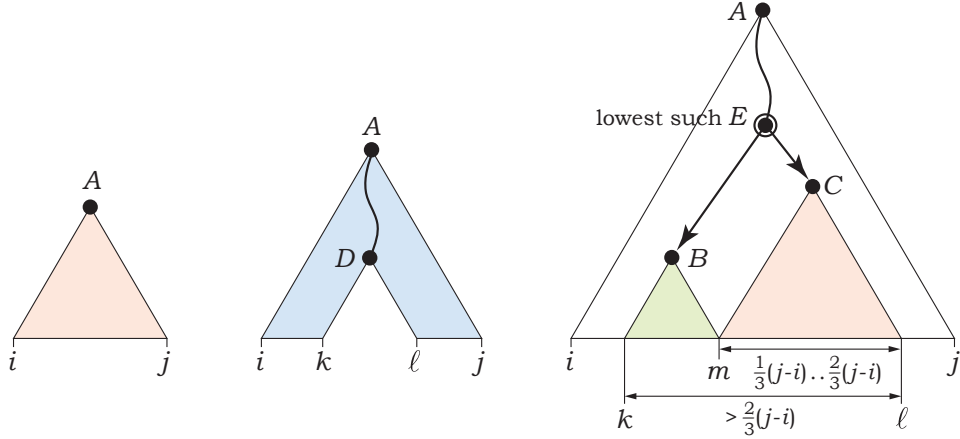


Figure 5: GF(2)-grammar parsing in NC^2 : (left) a parse tree represented by $A(i, j)$; (centre) a parse tree with a hole represented by $\frac{A}{D}(i, k, \ell, j)$; (right) a critical node in a parse tree.

Theorem 11. *Let $G = (\Sigma, N, R, S)$ be a GF(2)-grammar. Then there is a uniform family of circuits for testing strings of length n for being in $L(G)$, which are of depth $O((\log n)^2)$ and contain $O(n^7)$ nodes.*

Sketch of a proof. In order to test whether a GF(2)-grammar defines a given string $w = a_1 \dots a_n$, one should consider the corresponding ordinary grammar $G = (\Sigma, N, R, S)$ and determine the number of parse trees of w modulo 2. The circuit computes the values of predicates of two kinds.

- For each $A \in N$, a predicate $A(i, j)$, with $0 \leq i < j \leq n$, refers to a *substring* $a_{i+1} \dots a_j$. It determines whether the number of parse trees of this substring from A is odd.
- For $A, D \in N$, a predicate $\frac{A}{D}(i, k, \ell, j)$, with $0 \leq i \leq k \leq \ell \leq j \leq n$, refers to a *substring with a gap*, which is a pair $(a_{i+1} \dots a_k, a_{\ell+1} \dots a_j)$. It determines whether the number of the following trees is odd: these are parse trees with A as the root and with a D -subtree removed, where the leaves left of the path from A to D are $a_{i+1} \dots a_k$, and the leaves right of that path are $a_{\ell+1} \dots a_j$.

These predicates, illustrated in Figure 5(left, centre) are the same as used by Brent and Goldschlager [3] and by Rytter [24], except for being concerned with the *parity* rather than with the *existence* of parse trees. In other words, the Brent–Goldschlager–Rytter algorithm computes an element of a *Boolean semiring* for each substring and for each substring with a gap, whereas the new algorithm should compute an element of $GF(2)$. In fact, the Brent–Goldschlager–Rytter algorithm essentially uses the fact that $1 \vee 1 = 1$, while in the case of GF(2), a more careful calculation is needed.

The proposed circuit shall compute each of these $O(n^4)$ values of the form $A(i, j)$ and $\frac{A}{D}(i, k, \ell, j)$ in a gate of its own. Furthermore, $O(n^3)$ intermediate gates shall be used to compute each of these values from the values of other predicates.

Each **parse tree** τ matching the definition of $A(i, j)$ is known to have at least one subtree containing more than $\frac{1}{3}(j-i)$ and at most $\frac{2}{3}(j-i)$ leaves. For GF(2)-parsing, it is essential to define a unique subtree with this property. Thus, let a *critical node* be the deepest node containing more than $\frac{2}{3}(j-i)$ leaves in its subtree; let it be called E , and let B and C be its children. Let $a_{k+1} \dots a_m$ be the leaves in the B -subtree, and let the leaves in the C -subtree be $a_{m+1} \dots a_\ell$. Each subtree has at most $\frac{2}{3}(j-i)$ leaves, and at least one of them must have more than $\frac{1}{3}(j-i)$ leaves; the case when the C -subtree is larger is illustrated in Figure 5(right).

Then the subtree τ can be split into two subtrees corresponding to $B(k, m)$ and $C(m, \ell)$, and a subtree with a hole corresponding to $\frac{A}{E}(i, k, \ell, j)$. The total number of parse trees of $a_{i+1} \dots a_j$ from A that can be split with this particular placement of E , B and C is the product of these three numbers, and hence, $A(i, j)$

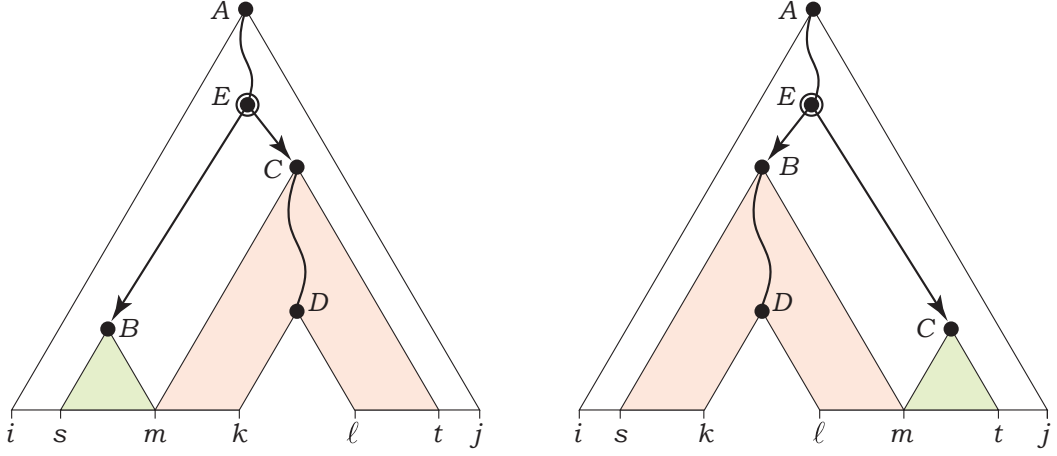


Figure 6: A critical node *on the path* from the root to the hole: (left) when the path continues to the left subtree, (right) when the path continues to the right subtree.

is determined by the following formula.

$$A(i, j) = \sum_{\substack{E \rightarrow BC \in R \\ k, m, \ell: i \leq k < m < \ell \leq j \\ \ell - k > \frac{2}{3}(j - i) \\ \frac{1}{3}(j - i) < \max(m - k, \ell - m) \leq \frac{2}{3}(j - i)}} \frac{A}{E}(i, k, \ell, j) \cdot B(k, m) \cdot C(m, \ell)$$

Each $A(i, j)$ is a sum of $O(n^3)$ values, for different rules and different k, ℓ, m . Modulo two, this can be computed by a circuit with $O(n^3)$ gates of depth $\log O(n^3) = O(\log n)$. Since there are $O(n^2)$ different gates $A(i, j)$, they are computed in $O(n^5)$ gates.

If τ is a **parse tree with a hole** corresponding to the definition of $\frac{A}{D}(i, k, \ell, j)$, then it contains a subtree with more than $\frac{1}{3}(k - i + j - \ell)$ and at most $\frac{2}{3}(k - i + j - \ell)$ leaves as well. Depending on the position of the hole D relative to this subtree—that is, whether the hole is in the subtree, right of the subtree or left of the subtree—there are several cases to consider.

If D is in the subtree, then let E be the parent node of this subtree, let $E \rightarrow BC$ be the rule applied in it, and assume that the next node on the path to D is C , as illustrated in Figure 6(left). Then the E -subtree contains more than $\frac{2}{3}(k - i + j - \ell)$ leaves, whereas the C -subtree contains more than $\frac{1}{3}(k - i + j - \ell)$ and at most $\frac{2}{3}(k - i + j - \ell)$ leaves. This is represented by the following formula.

$$\sum_{\substack{E \rightarrow BC \in R \\ s, m, t: i \leq s < m \leq k, \ell \leq t \leq j \\ k - s + t - \ell > \frac{2}{3}(k - i + j - \ell) \\ \frac{1}{3}(k - s + t - \ell) < k - m + t - \ell \leq \frac{2}{3}(k - s + t - \ell)}} \frac{A}{E}(i, s, t, j) \cdot B(s, m) \cdot \frac{C}{D}(m, k, \ell, t)$$

The case when the next node on the path to D is B , as in Figure 6(right), is handled symmetrically.

$$\sum_{\substack{E \rightarrow BC \in R \\ s, m, t: i \leq s \leq k, \ell \leq m < t \leq j \\ k - s + t - \ell > \frac{2}{3}(k - i + j - \ell) \\ \frac{1}{3}(k - s + t - \ell) < k - s + m - \ell \leq \frac{2}{3}(k - s + t - \ell)}} \frac{A}{E}(i, s, t, j) \cdot \frac{B}{D}(s, k, \ell, m) \cdot C(m, t)$$

Another case is when the subtree containing more than $\frac{1}{3}(k - i + j - \ell)$ and at most $\frac{2}{3}(k - i + j - \ell)$ leaves is to the left or to the right of the path from A to D . Then there exists a uniquely determined node

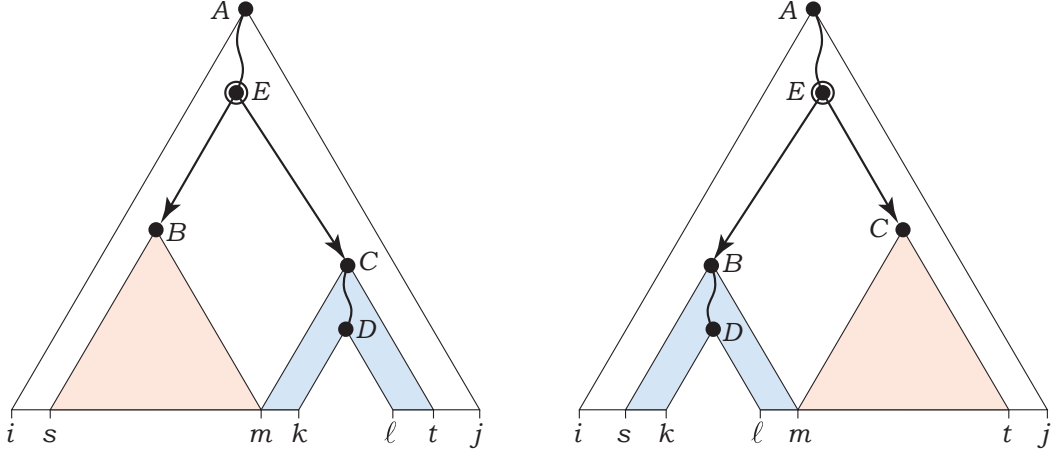


Figure 7: A critical node *off the path* from the root to the hole: (left) to the left of the path, (right) to the right of the path.

E on this path, with two descendants, B and C , one of which continues the path to D , whereas the other contains more than $\frac{2}{3}(k - i + j - \ell)$ leaves. Assume that the large subtree is the B -subtree, as illustrated in Figure 7(left). The next formula represents this case.

$$\sum_{\substack{E \rightarrow BC \in R \\ s, m, t: i \leq s < m \leq k, \ell \leq t \leq j \\ m - s > \frac{2}{3}(k - i + j - \ell)}} \frac{A}{E}(i, s, t, j) \cdot B(s, m) \cdot \frac{C}{D}(m, k, \ell, t)$$

In this case, the B -subtree may contain almost all leaves; the partition into more or less equal parts is then ensured by the formula defining $B(s, m)$.

For the case when the C -subtree contains more leaves, the formula is defined symmetrically and is illustrated in Figure 7(right).

$$\sum_{\substack{E \rightarrow BC \in R \\ s, m, t: i \leq s < m \leq k, \ell \leq t \leq j \\ t - m > \frac{2}{3}(k - i + j - \ell)}} \frac{A}{E}(i, s, t, j) \cdot \frac{B}{D}(s, k, \ell, m) \cdot C(m, t)$$

The total number of parse trees with a hole corresponding to the definition of $\frac{A}{D}(i, k, \ell, j)$ is a sum of the four above formulae. Each of them is a sum of $O(n^3)$ values, over all rules and all k, ℓ, m . Modulo two, this is computed by a circuit with $O(n^3)$ gates of depth $O(\log n)$. Since there are $O(n^4)$ gates to compute by these formulae, these parts of the circuit contain $O(n^7)$. \square

The last complexity question concerns *linear GF(2) grammars*, which have all rules of the form $A \rightarrow uBv$ or $A \rightarrow w$, with $B \in N$ and $u, v, w \in \Sigma^*$. Whereas ordinary linear grammars with the union operation are known to have an NL-complete uniform membership problem [27], the same problem for linear GF(2)-grammars is—by all means, predictably—complete for the complexity class $\oplus L$ of all problems decided by a nondeterministic logarithmic-space Turing machine that accepts by having an odd number of accepting paths.

Theorem 12. *The uniform membership problem for linear GF(2)-grammars is $\oplus L$ -complete. Furthermore, there exists a linear GF(2)-grammar that describes an $\oplus L$ -complete language.*

Sketch of a proof. The uniform membership problem for linear GF(2)-grammars is solved in $\oplus L$ by attempting to parse a given string using two pointers, choosing each rule nondeterministically, under the model of parity nondeterminism.

| | fixed membership | | | uniform |
|--|---------------------------------|-----------------------------------|---|---|
| | time | space | class | |
| Unambiguous linear ($\uplus, \text{LIN}\cdot$) | $O(n^2)$ | $O((\log n)^2)$ | in UL | in UL |
| Linear ($\cup, \text{LIN}\cdot$) | $O(n^2)$ | $O((\log n)^2)$ | NL-complete [27] | NL-complete [27] |
| Linear GF(2) ($\Delta, \text{LIN}\cdot$) | $O(n^2)$ | $O((\log n)^2)$ | $\oplus\mathbf{L}$-complete | $\oplus\mathbf{L}$-complete |
| Unambiguous ($\uplus, \text{UNAMB}\cdot$) | $O(n^2)$ | $O((\log n)^2)$ | in NC^2 [23] | P-complete |
| Ordinary (\cup, \cdot) | $O(n^\omega)$ [28] | $O((\log n)^2)$ | in NC^2 [3, 24] | P-complete [9] |
| GF(2) (Δ, \odot) | $O(n^\omega)$ | $O((\log n)^2)$ | in NC^2 | P-complete |
| Conjunctive (\cup, \cap, \cdot) | $O(n^\omega)$ [19] | $O(n)$ [18] | P-complete [18] | P-complete [18] |

Table 2: Complexity of grammars with different operations.

The “hardest” linear GF(2)-grammar is based on the problem of testing whether the number of s - t paths in a given directed graph is odd; this problem is $\oplus\mathbf{L}$ -complete, see Damm [7]. A grammar for the yes-instances of this problem can reuse the classical construction by Sudborough [27]: the encoding of graphs remains the same, and the union operation in the grammar is replaced with the symmetric difference. \square

The complexity of grammar families is compared in Table 2.

It would be important to know whether the emptiness problem for GF(2)-grammars is decidable. However, the equivalence problem for the unambiguous grammars reduces to this problem, and whether it is decidable is a major and long-standing open problem in formal language theory: two related problems include the equivalence problem for LR grammars [26] and the equivalence between an unambiguous grammar and a regular language [25]: both problems are decidable, which was established by remarkably non-trivial constructions. On the other hand, if the emptiness of GF(2)-grammars is undecidable, proving that would require new methods.

7. Conclusion

Although formal languages can be generalized to formal power series over any semiring, only two semirings correspond to formal languages as such, without multiplicities: these are the **Boolean semiring** and the **GF(2) field**. The former is the classical case, whereas the latter kind of formal languages have been investigated in this paper. This initial study has confirmed that language operations defined over GF(2) give rise to models that share some of the general appeal of the classical models of formal language theory, yet their properties are sufficiently different to warrant further study of these new operations.

An important question left open for GF(2)-grammars is developing a method for proving that some languages are not described by any such grammar. At the moment, the only remaining tool is Christol’s theorem [5] for formal power series, which is applicable only to languages over a unary alphabet. Perhaps for linear GF(2)-grammars, which are a special case of linear conjunctive grammars [17], finding such a method could be easier.

Acknowledgements

This work was supported by the Russian Science Foundation, project 18-11-00100.

References

- [1] M. Albrecht, G. Bard, W. Hart, “Algorithm 898: Efficient multiplication of dense matrices over GF(2)”, *ACM Transactions on Mathematical Software*, 37:1 (2010), article 9.
- [2] E. Bakinova, A. Basharin, I. Batmanov, K. Lyubort, A. Okhotin, E. Sazhneva, “Formal languages over GF(2)”, *Language and Automata Theory and Applications (LATA 2018, Bar-Ilan near Tel Aviv, Israel, 9–11 April 2018)*, LNCS 10792, 68–79.

- [3] R. P. Brent, L. M. Goldschlager, “A parallel algorithm for context-free parsing”, *Australian Computer Science Communications*, 6:7 (1984), 7.1–7.10.
- [4] J. A. Brzozowski, “Quotient complexity of regular languages”, *Journal of Automata, Languages and Combinatorics*, 15:1/2 (2010), 71–89.
- [5] G. Christol, “Ensembles presque periodiques k -reconnaissables”, *Theoretical Computer Science*, 9 (1979), 141–145.
- [6] M. Daley, M. Domaratzki, K. Salomaa, “Orthogonal concatenation: Language equations and state complexity”, *Journal of Universal Computer Science*, 16:5 (2010), 653–675.
- [7] C. Damm, “Problems complete for $\oplus L$ ”, *Information Processing Letters*, 36 (1990), 247–250.
- [8] S. Ginsburg, H. G. Rice, “Two families of languages related to ALGOL”, *Journal of the ACM*, 9 (1962), 350–371.
- [9] L. M. Goldschlager, “The monotone and planar circuit value problems are log space complete for P”, *SIGACT News*, 9:2 (1977), 25–29.
- [10] A. Jež, “Conjunctive grammars can generate non-regular unary languages”, *International Journal of Foundations of Computer Science*, 19:3 (2008), 597–615.
- [11] A. Jež, A. Okhotin, “Conjunctive grammars over a unary alphabet: undecidability and unbounded growth”, *Theory of Computing Systems*, 46:1 (2010), 27–58.
- [12] A. Jež, A. Okhotin, “Complexity of equations over sets of natural numbers”, *Theory of Computing Systems*, 48:2 (2011), 319–342.
- [13] A. Jež, A. Okhotin, “Unambiguous conjunctive grammars over a one-symbol alphabet”, *Theoretical Computer Science*, 665 (2017), 13–39.
- [14] G. Jirásková, A. Okhotin, “State complexity of unambiguous operations on deterministic finite automata”, *Descriptive Complexity of Formal Systems (DCFS 2018, Halifax, Canada, 25–27 July 2018)*, LNCS 10952, to appear.
- [15] E. L. Leiss, “Unrestricted complementation in language equations over a one-letter alphabet”, *Theoretical Computer Science*, 132 (1994), 71–93.
- [16] A. N. Maslov, “Estimates of the number of states of finite automata”, *Soviet Mathematics Doklady*, 11 (1970), 1373–1375.
- [17] A. Okhotin, “On the equivalence of linear conjunctive grammars to trellis automata”, *RAIRO Informatique Théorique et Applications*, 38:1 (2004), 69–88.
- [18] A. Okhotin, “Conjunctive and Boolean grammars: the true general case of the context-free grammars”, *Computer Science Review*, 9 (2013), 27–59.
- [19] A. Okhotin, “Parsing by matrix multiplication generalized to Boolean grammars”, *Theoretical Computer Science*, 516 (2014), 101–120.
- [20] A. Okhotin, “Underlying principles and recurring ideas of formal grammars”, *Language and Automata Theory and Applications (LATA 2018, Bar-Ilan near Tel Aviv, Israel, 9–11 April 2018)*, LNCS 10792, 36–59.
- [21] A. Okhotin, P. Rondogiannis, “On the expressive power of univariate equations over sets of natural numbers”, *Information and Computation*, 212 (2012), 1–14.
- [22] I. Petre, A. Salomaa, “Algebraic systems and pushdown automata”, in: Droste, Kuich, Vogler (Eds.), *Handbook of Weighted Automata*, Springer, 2009, 257–289.
- [23] P. Rossmanith, W. Rytter, “Observation on $\log(n)$ time parallel recognition of unambiguous cfl’s”, *Information Processing Letters*, 44:5 (1992), 267–272.
- [24] W. Rytter, “On the recognition of context-free languages”, *Fundamentals of Computation Theory (FCT 1985, Cottbus, Germany)*, LNCS 208, 315–322.
- [25] A. L. Semenov, “Algorithmic problems for power series and for context-free grammars”, *Doklady Akademii Nauk SSSR*, 212 (1973), 50–52.
- [26] G. Sénizergues, “ $L(A) = L(B)$? decidability results from complete formal systems”, *Theoretical Computer Science*, 251:1–2 (2001), 1–166.
- [27] I. H. Sudborough, “A note on tape-bounded complexity classes and linear context-free languages”, *Journal of the ACM*, 22:4 (1975), 499–500.
- [28] L. G. Valiant, “General context-free recognition in less than cubic time”, *Journal of Computer and System Sciences*, 10:2 (1975), 308–314.
- [29] S. Yu, Q. Zhuang, K. Salomaa, “The state complexity of some basic operations on regular languages”, *Theoretical Computer Science*, 125 (1994), 315–328.
- [30] L. van Zijl, “On binary \oplus -NFAs and succinct descriptions of regular languages”, *Theoretical Computer Science*, 328:1–2 (2004), 161–170.