Parsing unary Boolean grammars using online convolution[*]

Alexander Okhotin[†] Christian Reitwießner[‡]

Abstract

Consider context-free grammars generating strings over a one-letter alphabet. For the membership problem for such grammars, stated as "Given a grammar G and a string a^n, determine whether a^n is generated by G", only a naïve $O(|G| \cdot n^2)$-time algorithm is known. This paper develops a new algorithm solving this problem, which is based upon fast multiplication of integers, works in time $|G| \cdot n \log^3 n \cdot 2^{O(\log^* n)}$, and is applicable to the more general conjunctive and Boolean grammars. The algorithm is based upon (a simplification of) the online integer multiplication algorithm by Fischer and Stockmeyer ("Fast on-line integer multiplication", *Journal of Computer and System Sciences*, 1974).

1 Introduction

Context-free grammars are the foremost mathematical model of syntax. The main idea behind these grammars, which corresponds to the intuitive notion of syntax so well, is that the syntactic properties of a string are determined on the basis of the syntactic properties of its proper substrings, and that the substrings are combined by concatenating them. Furthermore, in context-free grammars, syntactic conditions are always expressed as *disjunction of concatenations*, as in the two rules $A \to BC \mid DE$.

Motivated by the fact that other Boolean connectives are as useful and natural as the disjunction, two extensions of context-free grammars were introduced. These are *conjunctive grammars* [14], in which every rule $A \to \alpha_1 \& \ldots \& \alpha_n$ contains an explicit conjunction of one or more concatenations, and *Boolean grammars* [15], which further allow the use of negation. Both conjunctive and Boolean grammars are notable for preserving the main practical properties of the context-free grammars: most importantly, the parsing techniques. In particular, the membership of a string of length n in the language generated by a grammar G can be tested in time $\Theta(|G| \cdot n^3)$ by a straightforward adaptation of the Cocke–Kasami–Younger algorithm [14, 15], and a more careful examination showed that Valiant's [20] reduction of context-free recognition to *matrix multiplication* is still applicable to Boolean grammars [17], leading to a parsing algorithm working in

[*]Supported by the Academy of Finland under grant 134860.

[†]Academy of Finland, *and* Department of Mathematics, University of Turku, Turku FI–20014, Finland. E-mail: `alexander.okhotin@utu.fi`.

[‡]Julius-Maximilians-Universität Würzburg, Theoretische Informatik, Würzburg D–97074, Germany. E-mail: `reitwiessner@informatik.uni-wuerzburg.de`.

Dagstuhl Seminar Proceedings 10501
Advances and Applications of Automata on Words and Trees
http://drops.dagstuhl.de/opus/volltexte/2011/3146

time $O(|G| \cdot BMM(n) \log n)$, where $BMM(n)$ is the number of bit operations needed to multiply two $n \times n$ Boolean matrices [17]. Using the best known upper bound on matrix multiplication [1] yields $O(|G| \cdot n^{2.376})$ time complexity of parsing.

The nontriviality of conjunctive grammars over a one-letter alphabet was discovered by Jeż [9], who presented a grammar generating $\{ a^{4^n} \mid n \geqslant 0 \}$, and later Okhotin and Rondogiannis [18] showed that a variant of this language can be generated by a conjunctive grammar with a unique nonterminal symbol. Subsequent research on conjunctive grammars over a unary alphabet revealed their nontrivial properties, such as the following. For every recursive function, one can find a strictly greater function $f \colon \mathbb{N} \to \mathbb{N}$, such that the language $\{ a^{f(n)} \mid n \geqslant 0 \}$ is generated by a conjunctive grammar [10]. The decision problem of whether a given conjunctive grammar G over a unary alphabet generates a fixed language $L_0 \subseteq a^*$ is undecidable for any conjunctive L_0 [10]. There exists a particular language $L \subseteq a^*$ generated by a conjunctive grammar with one nonterminal symbol, such that testing the membership of a string a^n in this language, with n given in binary, is an EXPTIME-complete problem [11, 12].

These nontrivial facts, especially the latter complexity lower bound, motivate an investigation of further complexity aspects of unary conjunctive and Boolean grammars. In the case of a one-letter alphabet, the basic Cocke–Kasami–Younger algorithm is naturally modified to work in time $O(|G| \cdot n^2)$ on an input string a^n, which applies to the membership problem for all families between context-free grammars and Boolean grammars. Further motivation comes from the area of circuits over sets of natural numbers, which may be regarded as a special case of grammars over a unary alphabet, without circular dependencies of nonterminals. These circuits have a long history of complexity-theoretical studies, beginning with the works of Stockmeyer and Meyer [19] and Wagner [21], and with further investigation in progress [13, 6, 7]. No method of evaluating such circuits faster than in time $O(|C| \cdot n^2)$, where $|C|$ is the size of the circuit, has been proposed.

This paper aims to improve over this naïve upper bound in the same way as Valiant's [20] algorithm improved over the basic cubic-time methods of context-free recognition. Valiant's method was based upon finding instances of the matrix multiplication problem inside the total bulk of $\Theta(|G| \cdot n^3)$ bit operations used by the Cocke–Kasami–Younger recognition. In this paper, similarly, the $\Theta(|G| \cdot n^2)$ bit operations in the obvious algorithm for the membership problem for Boolean grammars over a unary alphabet are found to contain instances of another fundamental problem of computer algebra: *convolution of bitvectors* in the Boolean semiring, which is closely related to multiplication of long numbers. The known efficient algorithms for the latter problem are hence applied to solve the membership problems for grammars in time $|G| \cdot n \log^{O(1)} n$.

Definitions and examples of conjunctive and Boolean grammars are given in Section 2, which also presents the basic algorithm for recognizing the membership of the string a^n in the language generated by a grammar G in time $O(|G| \cdot n^2)$. An improved algorithm, defined in Section 3, employs at most $|G|$ instances of the online Boolean convolution procedure, and runs in time $O(|G| \cdot OC(n))$, where $OC(n)$ is the complexity of online Boolean convolution. The online convolution enables the algorithm to gradually output the membership of subsequent numbers $1, \ldots, n$ without knowing n in advance.

The next Section 4 estimates the complexity of the algorithm using a particular convolution procedure. This procedure operates through the state-of-the-art multiplication algorithm by Fürer [4], which multiplies two n-bit numbers in $n \log n 2^{O(\log^* n)}$ bit operations. The resulting algorithm recognizes the membership of n in $|G| \cdot n \log^3 n 2^{O(\log^* n)}$ bit operations, or in $|G| \cdot n \log^2 n 2^{O(\log^* n)}$ operations if the special case of an unambiguous grammar.

2 Conjunctive and Boolean grammars

Definition 1 (Okhotin [14]). *A conjunctive grammar is a quadruple $G = (\Sigma, N, R, S)$, in which Σ and N are disjoint finite nonempty sets of terminal and nonterminal symbols, respectively; R is a finite set of rules, each of the form*

$$A \to \alpha_1 \& \ldots \& \alpha_m \quad (\textit{with } A \in N,\ m \geqslant 1 \textit{ and } \alpha_1, \ldots, \alpha_m \in (\Sigma \cup N)^*), \tag{1}$$

where "&" is a special symbol not in $\Sigma \cup N$; and $S \in N$ is a nonterminal designated as the start symbol.

A rule (1) shall be called *terminating* if it is of the form $A \to w$ with $w \in \Sigma^*$ (and with $n = 1$), and *non-terminating* otherwise.

Informally, a rule (1) states that if a string is generated by each α_i, then it is generated by A. One way of formalizing this understanding is to use rewriting of terms over conjunction and concatenation, which generalizes Chomsky's string rewriting [14]. Under this definition, a nonterminal symbol A occurring in any term may be rewritten by a subterm $(\alpha_1 \& \ldots \& \alpha_m)$ according to a rule (1). Furthermore, a subterm $(w \& \ldots \& w)$ with $w \in \Sigma^*$ may be rewritten by the string w. For each $A \in N$, let $L_G(A)$ be the set of strings over Σ generated by term rewriting from the term A. Define $L(G) = L_G(S)$.

An equivalent definition can be given using language equations. This definition generalizes the well-known characterization of the context-free grammars by equations, due to Ginsburg and Rice [5]. Given a conjunctive grammar $G = (\Sigma, N, R, S)$, the associated system of language equations is a system of equations in variables N, in which each variable assumes the value of a language over Σ, and which contains the following equation for every variable A:

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \in R} \bigcap_{i=1}^{m} \alpha_i \quad (\text{for all } A \in N). \tag{2}$$

Each occurrence of a symbol $a \in \Sigma$ in such a system defines a constant language $\{a\}$, while each empty string denotes a constant language $\{\varepsilon\}$. A solution of a system is a vector of languages $(\ldots, L_C, \ldots)_{C \in N}$, such that the substitution of L_C for C, for all $C \in N$, turns each equation (2) into an equality. Every such system has at least one solution, and among them a *least solution* with respect to componentwise inclusion. This solution consists of exactly the languages generated by term rewriting: $(\ldots, L_G(C), \ldots)_{C \in N}$.

Obviously, every finite intersection of context-free languages, such as the language $\{\, a^n b^n c^n \mid n \geqslant 0 \,\}$, is generated by a conjunctive grammar. Furthermore, there are known conjunctive grammars for some languages outside of the intersection closure of the context-free languages, such as $\{\, wcw \mid w \in \{a, b\}^* \,\}$ [14]. Since this paper is concerned with grammars over a one-letter alphabet, the following example is more relevant.

Example 1 (Jeż [9]). *The following conjunctive grammar with the start symbol A_1 generates the language $\{ a^{4^n} \mid n \geqslant 0 \}$:*

$$\begin{array}{lcl} A_1 & \to & A_1A_3 \& A_2A_2 \mid a \\ A_2 & \to & A_1A_1 \& A_2A_6 \mid aa \\ A_3 & \to & A_1A_2 \& A_6A_6 \mid aaa \\ A_6 & \to & A_1A_2 \& A_3A_3 \end{array}$$

Each nonterminal A_i generates the language $\{ a^{i \cdot 4^n} \mid n \geqslant 0 \}$.

The idea behind this example is to manipulate base-4 positional notations of numbers, and one can verify by substitution that the given languages form a solution of language equations corresponding to this grammar. This method was subsequently generalized to construct more sophisticated examples of conjunctive grammars over a unary alphabet [9, 10, 11, 12, 18].

There is a generalization of the Chomsky normal form for conjunctive grammars.

Definition 2 (Binary normal form [14]). *A conjunctive grammar $G = (\Sigma, N, R, S)$ is in binary normal form if every rule in R is of the form*

$$\begin{array}{l} A \to B_1C_1 \& \ldots \& B_nC_n \quad (n \geqslant 1,\ B_i, C_i \in N) \\ A \to a \\ S \to \varepsilon \quad \text{(only if } S \text{ does not appear in right-hand sides of rules).} \end{array}$$

Every conjunctive grammar can be effectively transformed to a conjunctive grammar in binary normal form generating the same language [14].

The rules in *Boolean grammars* are the same as in conjunctive grammars, but every conjunct in every rule may be negated.

Definition 3 (Okhotin [15]). *A Boolean grammar is a quadruple $G = (\Sigma, N, R, S)$, where Σ and N are disjoint finite non-empty sets of terminal and nonterminal symbols respectively; R is a finite set of rules of the form*

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n, \tag{3}$$

where $m + n \geqslant 1$, $\alpha_i, \beta_i \in (\Sigma \cup N)^*$; $S \in N$ is the start symbol of the grammar.

A grammar is interpreted as a system of language equations in variables N, in which the equation for each $A \in N$ is

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in R} \left[\bigcap_{i=1}^{m} \alpha_i \cap \bigcap_{j=1}^{n} \overline{\beta_j} \right]. \tag{4}$$

According to the simplest definition of Boolean grammars, this system is required to have a unique solution, and this unique solution defines the languages generated by the nonterminals of the grammar. This definition is assumed in this paper.

Definition 4. *The concatenation of two languages, K and L, is said to be unambiguous, if every string $w \in K \cdot L$ has a unique factorization $w = uv$ with $u \in K$ and $v \in L$.*

A Boolean grammar $G = (\Sigma, N, R, S)$ has unambiguous concatenation, if all concatenations in the corresponding system of language equations (4) are unambiguous under the substitution $A = L_G(A)$.

A Boolean grammar is unambiguous if it has unambiguous concatenation and furthermore, every union in the system (4) is disjoint under the substitution $A = L_G(A)$.

It is known that every Boolean grammar can be transformed to an equivalent grammar in *binary normal form* [15], in which every rule in R is of the form

$$\begin{aligned} &A \to B_1C_1\& \ldots \& B_nC_m \& \neg D_1E_1 \& \ldots \& \neg D_nE_n \& \neg\varepsilon \\ &\qquad\qquad\qquad\qquad (m \geqslant 1,\ n \geqslant 0,\ B_i, C_i, D_j, E_j \in N) \\ &A \to a \\ &S \to \varepsilon \quad \text{(only if } S \text{ does not appear in right-hand sides of rules).} \end{aligned}$$

Furthermore, if the original grammar has unambiguous concatenation, then the resulting grammar in the normal form is unambiguous [16].

In particular, this normal form is used to obtain a simple generalization of the Cocke–Kasami–Younger parsing algorithm to conjunctive and Boolean grammars, which still works in time $O(n^3)$ [14, 15]. Given a grammar G and an input string $w = a_1 \ldots a_n$, this algorithm computes the sets $T_{i,j} = \{\, A \in N \mid a_{i+1} \ldots a_j \in L_G(A) \,\}$, for all nonempty substrings of w. The bottleneck of this algorithm is the need to compute n^2 unions of the form $\bigcup_{i<k<j} T_{i,k} \times T_{k,j}$, which represent the set of all concatenations BC that generate the corresponding substring $a_{i+1} \ldots a_j$. A more efficient way of calculating these sets via Boolean matrix multiplication was invented by Valiant [20], and it can be used to obtain a $O(BMM(n))$-time parsing algorithm for Boolean grammars [17], where $BMM(n)$ is the complexity of multiplying two $n \times n$ Boolean matrices.

In the case of a unary alphabet, a string of length n has only n distinct nonempty substrings, and the basic cubic-time parsing algorithm can be simplified down to the following straightforward procedure.

Algorithm 1. *Input: Boolean Grammar $G = (\{a\}, N, S, R)$ in binary normal form, $n \geqslant 1$. Data structures: set $P \subseteq N \times N$ containing all pairs of nonterminals occuring (positively or negatively) in right-hand-sides of rules; for each $A \in N$, a bitvector $V_A[1..n]$, initialized to* false*; for each $(B, C) \in P$, a Boolean variable W_{BC}.*

```
1: **if** $n = 0$ **then**
2:     **return** $S \to \varepsilon \in R$
3: **for all** $A \in N$ with $A \to a \in P$ **do**
4:     $V_A[1] = $ **true**
5: **for** $i := 2$ to $n$ **do**
6:     **for all** $(B, C) \in P$ **do**
7:         $W_{BC} :=$ **false**
8:         **for** $j := 1$ to $i - 1$ **do**
9:             $W_{BC} := W_{BC} \vee (V_B[j] \wedge V_C[i-j])$
10:    **for all** $A \to B_1C_1\& \ldots \& B_mC_m \& \neg D_1E_1 \& \ldots \& \neg D_rE_r \in R$ **do**
11:        $V_A[i] := V_A[i] \vee (W_{B_1C_1} \wedge \ldots \wedge W_{B_kC_k} \wedge \neg W_{D_1E_1} \wedge \ldots \wedge \neg W_{D_rE_r})$
12: **return** $V_S[n]$
```

3 Recognition by convolution

The computation that is done in lines 7 to 9 of Algorithm 1, essentially calculates one output bit of the Boolean convolution of the bitvectors V_B and V_C. As we will see later,

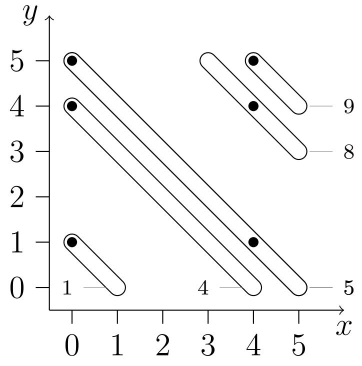

Figure 1: Convolution of the bitvectors $x = (1, 0, 0, 0, 1, 0)$ and $y = (0, 1, 0, 0, 0, 1, 1)$, illustrated graphically. Circles represent true conjunctions and true disjunctions are grouped together. The result is $z = (0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0)$.

this computation is done quite inefficiently. The term "unambiguous" in the following definition will become important in section 4.

Definition 5. *The* Boolean convolution *maps two bitvectors $x = (x_0, \ldots, x_{n-1})$, $y = (y_0, \ldots, y_{n-1}) \in \mathbb{B}^n$ to another bitvector $z = x \circ y = (z_0, \ldots, z_{2n-2}) \in \mathbb{B}^{2n-1}$, where $z_i = \bigvee_{j=0}^{i} x_j \wedge y_{j-i}$.*

The convolution of two bitvectors x and y is unambiguous *if for every $0 \leq k \leq 2n-2$ there is at most one combination of i, j such that $i + j = k$ and $x_i = y_j = 1$.*

This definition is illustrated in Figure 1, where the axes correspond to the two bitvectors, circles represent true conjunctions and true disjunctions are grouped together. A naïve approach to computing the convolution, as the one implemented in Algorithm 1, involves evaluating all $\Theta(n^2)$ Boolean operations, as per the definition. Much more efficient $n \log^{O(1)} n$-time methods were developed in connection with the problem of fast multiplication of integers.

Using these algorithms, one can determine the membership of a string a^n in the language defined by a regular expression e over a unary alphabet, by calculating the language generated by each substring modulo $\{\varepsilon, a, \ldots, a^n\}$. This works in time $|e| \cdot n \log^{O(1)} n$, where $|e|$ is the number of symbols in the regular expression.

However, this method of evaluating subexpressions one by one is not directly applicable to Boolean grammars, which may include circularities in the definition. In general, the membership of a word a^n in a language $L_G(A)$ depends upon the membership of all words $a^1, \ldots, a^{n-1}$ in the languages generated by all nonterminals of the grammar G, and there is no known way to compute the membership of a^n in $L_G(A)$ without first computing the membership of all shorter words in all languages. To be precise, the membership of a^n in $L_G(A)$ is a function of the membership of a^n in $L_G(BC)$, for all $B, C \in N$, and the latter is one bit of the convolution computed in the lines 7–9 of Algorithm 1. Since the Boolean vectors being convolved depend on the previously calculated bits of the convolution, one must use the online variant of Boolean convolution, defined as follows.

Definition 6. *Let $x = (x_0, \ldots, x_{n-1})$, $y = (y_0, \ldots, y_{n-1}) \in \mathbb{B}^n$ be two bitvectors. An* online convolution algorithm*, which computes their convolution $z = x \circ y = (z_0, \ldots, z_{2n-2})$,* *receives x and y bit by bit, and writes each i-th bit of z before reading any input x_j, y_j for $j > i$.*

Note that here, the output bit number i can depend on the input bits $0, \dots, i$, and not only on $0, \dots, i-1$, as required for computing the concatenation. As we will see later, this is compensated by the fact that $\varepsilon \notin L_G(B), L_G(C)$ when compting $L_G(B) \cdot L_G(C)$.

Fischer and Stockmeyer [3] showed how to transform ordinary convolution algorithms into their online variants with not much overhead in the computation. Though they are mainly concerned with integer multiplication, it is also shown (in Section 4.1) how this result can be extended to so-called *generalized linear products* defined by Fischer and Paterson [2]. Since Boolean convolution is such a generalized linear product, the following holds.

Theorem 1 (Fischer, Stockmeyer [3])**.** *Let $C(n)$ be the time needed to compute the Boolean convolution of two bitvectors of length n satisfying $n \leqslant C(n) \leqslant C(2n)/2 \leqslant k\,C(n)$ for some k. There is an algorithm that computes the online Boolean convolution in time $O(C(n) \cdot \log n)$.*

Since Fischer and Stockmeyer do not explicitly prove the case of Boolean convolution, a full version of the present paper will contain a simplified version of their proof for multiplication applied to Boolean convolution.

Using this result, we can change Algorithm 1 to directly use online convolution and can concentrate on finding a good algorithm for ordinary convolution.

Algorithm 2. *Input: Boolean Grammar $G = (\{a\}, N, S, R)$ in binary normal form, $n \geqslant 0$. Data structures: set $P \subseteq N \times N$ containing all pairs of nonterminals occuring (positively or negatively) in right-hand-sides of rules; for each $A \in N$, a bitvector $V_A[1..n]$, initialized to* false*; for each $(B, C) \in P$, a Boolean variable W_{BC}.*
Furthermore, we run parallel instances of an online convolution algorithm $\texttt{conv}_{BC}$ for each $(B, C) \in P$ such that $\texttt{conv}_{BC}$ computes the convolution of $V_B[1..n]$ and $V_C[1..n]$. We access its output bits by $\texttt{conv}_{BC}[0..2n]$.

```
1: if $n = 0$ then
2:     return $S \to \varepsilon \in R$
3: for all $A \in N$ with $A \to a \in R$ do
4:     $V_A[1] :=$ true
5: for $i := 2$ to $n$ do
6:     feed all $V_A[i-1]$, $A \in N$ to the convolution algorithms
7:     for all $(B, C) \in P$ do
8:         $W_{BC} := \texttt{conv}_{BC}[i-2]$
9:     for all $A \to B_1C_1 \& \dots \& B_kC_k \& \neg D_1E_1 \& \dots \& \neg D_rE_r \in R$ do
10:        $V_A[i] := V_A[i] \vee (W_{B_1C_1} \wedge \dots \wedge W_{B_kC_k} \wedge \neg W_{D_1E_1} \wedge \dots \wedge \neg W_{D_rE_r})$
11: return $V_S[n]$
```

Lemma 1. *Let $OC(n) \geqslant n$ be the complexity of computing an n-bit online Boolean convolution. The problem whether a given string a^n is generated by a given Boolean grammar G in binary normal form can be solved in time $O(|G| \cdot OC(n))$.*

Proof. We show that Algorithm 2 solves the stated problem.

Correctness. We first show that the algorithms $\texttt{conv}_{BC}$ is used in a way such that they always have enough bits of the input available to produce the requested output bits.

Let $2 \leq i \leq n$ and $(B, C) \in P$. Because of line 6, the bit $\texttt{conv}_{BC}[i-2]$ is requested after the bits $V_B[1..(i-1)]$ and $V_C[1..(i-1)]$ have been fed to the algorithm. Transforming these bitvectors to the bitvectors x and y from Definition 6, it holds that $x = (x_0, \ldots, x_{n-1}) = (V_B[1], \ldots, V_B[n])$, $y = (y_0, \ldots, y_{n-1}) = (V_C[1], \ldots, V_C[n])$. Since $V_B[i-1]$ corresponds to x_{i-2}, this means that, by the time we want to read output bit $i-2$ (i.e. z_{i-2}), the algorithm has access to $V_B[1..(i-1)]$ and $V_C[1..(i-1)]$, i.e. to $(x_0, \ldots, x_{i-2})$ and $(y_0, \ldots, y_{i-2})$. Thus, the online convolution algorithm can correctly produce the requested output bit.

It remains to show that this algorithm does the same as Algorithm 1. Since the replacement of lines 7 to 9 by the lines 6 to 8 is the only difference in the algorithms, we have to argue that $\texttt{conv}_{BC}[i-2] = (V_B[1] \wedge V_C[i-1]) \vee \cdots \vee (V_B[i-1] \wedge V_C[1])$ for $(B, C) \in P$ and $2 \leq i \leq n$. Using the same index transformation as before, i.e. $x = (x_0, \ldots, x_{n-1}) = (V_B[1], \ldots, V_B[n])$, $y = (y_0, \ldots, y_{n-1}) = (V_C[1], \ldots, V_C[n])$, we get

$$\begin{aligned}
\texttt{conv}_{BC}[i-2] &= \bigvee_{j=0}^{i-2} x_j \wedge y_{i-2-j} \\
&= \bigvee_{j=0}^{i-2} V_B[j+1] \wedge V_C[i-2-j+1] \\
&= \bigvee_{r=1}^{i-1} V_B[r] \wedge V_C[i-r] \\
&= (V_B[1] \wedge V_C[i-1]) \vee \cdots \vee (V_B[i-1] \wedge V_C[1]),
\end{aligned}$$

and thus both algorithms compute the same result.

Time Complexity. Because we run one online convolution algorithm for each pair $(B, C) \in P$, we get $O(|G| \cdot OC(n))$ for the online convolutions alone. All other operations need $O(|G| \cdot n)$ time and since $OC(n) \geqslant n$, we get the stated complexity. $\square$

4 Boolean convolution

We have reduced parsing unary Boolean grammars to online Boolean convolution which in turn was reduced to ordinary Boolean convolution. We now apply the final reduction and show how to use an arbitrary integer multiplication algorithm to compute Boolean convolutions. As a consequence, progress in algorithms or implementations for integer multiplication will also improve the complexity of parsing unary Boolean grammars.

Lemma 2. *Let $M(n)$ be the time complexity of multiplying two n-bit integers. The Boolean convolution of two bitvectors of length n can be computed in time $O(M(n \log n))$ and in time $O(M(n))$ if the convolution is unambiguous.*

Proof. In the following, we interpret $\mathbb{B} = \{0, 1\}$ as a subset of the integers. Let $x = (x_0, x_1, \ldots, x_{n-1})$, $y = (y_0, y_1, \ldots, y_{n-1}) \in \mathbb{B}^n$. We begin with the unambiguous case.

Define the n-bit numbers

$$a = \sum_{i=0}^{n-1} x_i 2^i \qquad\qquad b = \sum_{j=0}^{n-1} y_j 2^j .$$

The product of these numbers is

$$a \cdot b = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} x_i y_j 2^{i+j} = \sum_{k=0}^{2n-2} 2^k \sum_{i+j=k} x_i y_j = \sum_{k=0}^{2n-2} 2^k \sum_{i=0}^{k} x_i y_{k-i}$$

and since the convolution is unambiguous, the coefficient of each 2^k is either zero or one. Thus the bitvector of the convolution coincides with the binary representation of the product.

This is not the case anymore if the convolution is not unambiguous, but there, we can avoid carries by padding in the following way: For

$$a = \sum_{i=0}^{n-1} x_i 2^{i\lceil \log n \rceil} \qquad\qquad b = \sum_{j=0}^{n-1} y_j 2^{j\lceil \log n \rceil}$$

we get

$$a \cdot b = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} x_i y_j 2^{(i+j)\lceil \log n \rceil} = \sum_{k=0}^{2n-2} 2^{k\lceil \log n \rceil} \sum_{i+j=k} x_i y_j . = \sum_{k=0}^{2n-2} 2^{k\lceil \log n \rceil} \sum_{i=0}^{k} x_i y_{k-i} .$$

Since for any k, it holds that $\sum_{i+j=k} x_i y_j \leq n \leq 2^{\lceil \log n \rceil}$, there will be no carry between blocks of $\lceil \log n \rceil$ bits and thus the bitvector of the convolution can be extracted from the binary representation of the product. As the multiplication is done on integers with binary length $n\lceil \log n \rceil$, the assertion follows. $\square$

Applying the currently best known upper bound for integer multiplication by Fürer [4], we get the following result.

Proposition 1. *The Boolean convolution of two bitvectors of length n can be computed in time $n \log^2 n \cdot 2^{O(\log^* n)}$ and in time $n \log n \cdot 2^{O(\log^* n)}$ if the convolution is unambiguous.*

Proof. We apply Fürer's algorithm for multiplication with time complexity $n \log n \cdot 2^{O(\log^* n)}$. The unambiguous case is obvious and in the general case we get a time complexity of $O((n \log n) \log(n \log n) 2^{O(\log^*(n \log n))}) = n \log^2 n 2^{O(\log^* n)}$. $\square$

Theorem 2. *Let $M(n)$ be the time complexity of multiplying two n-bit integers.*

The problem of determining whether a word a^n is generated by a Boolean grammar G in binary normal form can be solved in time $O(|G| \cdot M(n \log n) \cdot \log n)$. If the grammar is known to be unambiguous, a variant of the algorithm works in time $O(|G| \cdot M(n) \cdot \log n)$.

The currently best known value for $M(n)$ yields the complexities $|G| \cdot n \log^3 n \cdot 2^{O(\log^* n)}$ and $|G| \cdot n \log^2 n \cdot 2^{O(\log^* n)}$, respectively.

Proof. By Lemma 2, Boolean convolution can be computed in time $C(n) = O(M(n \log n))$ and unambiguous Boolean convolution in time $O(M(n))$. By Theorem 1, these algorithms can be convorted to online variants with complexity $OC(n) = O(M(n \log n) \log n)$ and $O(M(n) \log n)$, respectively. Finally, from Lemma 1, we obtain the assertions. Using Fürer's result $M(n) = n \log n \cdot 2^{O(\log^* n)}$ [4], the explicit bounds follow. $\square$

References

[1] D. Coppersmith, S. Winograd, “Matrix multiplication via arithmetic progressions”, *Journal of Symbolic Computation*, 9:3 (1990), 251–280.

[2] M. J. Fischer, M. S. Paterson, “String-matching and other products”, Technical report MIT-LCS-TM-041, Massachusetts Institute of Technology, 1974.

[3] M. J. Fischer, L. J. Stockmeyer, “Fast on-line integer multiplication”, *Journal of Computer and System Sciences*, 9:3 (1974), 317–331.

[4] M. Fürer, “Faster integer multiplication”, *SIAM Journal on Computing*, 39:3 (2009), 979–1005.

[5] S. Ginsburg, H. G. Rice, “Two families of languages related to ALGOL”, *Journal of the ACM*, 9 (1962), 350–371.

[6] C. Glaßer, K. Herr, C. Reitwießner, S. D. Travers, M. Waldherr, “Equivalence problems for circuits over sets of natural numbers”, *Theory of Computer Systems*, 46:1 (2010), 80–103.

[7] C. Glaßer, C. Reitwießner, S. D. Travers, M. Waldherr, “Satisfiability of algebraic circuits over sets of natural numbers”, *Discrete Applied Mathematics*, 158:13 (2010), 1394–1403.

[8] D. T. Huynh, “Commutative grammars: the complexity of uniform word problems”, *Information and Control*, 57:1 (1983), 21–39.

[9] A. Jeż, “Conjunctive grammars can generate non-regular unary languages”, *International Journal of Foundations of Computer Science*, 19:3 (2008), 597–615.

[10] A. Jeż, A. Okhotin, “Conjunctive grammars over a unary alphabet: undecidability and unbounded growth”, *Theory of Computing Systems*, 46:1 (2010), 27–58.

[11] A. Jeż, A. Okhotin, “Complexity of solutions of equations over sets of natural numbers”, *25th Annual Symposium on Theoretical Aspects of Computer Science* (STACS 2008, Bordeaux, France, 21–23 February, 2008), 373–383.

[12] A. Jeż, A. Okhotin, “One-nonterminal conjunctive grammars over a unary alphabet”, *Computer Science in Russia* (CSR 2009, Novosibirsk, Russia, 18–23 August, 2009), LNCS 5675, 191–202.

[13] P. McKenzie, K. W. Wagner, “The complexity of membership problems for circuits over sets of natural numbers”, *Computational Complexity*, 16:3 (2007), 211–244.

[14] A. Okhotin, “Conjunctive grammars”, *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.

[15] A. Okhotin, “Boolean grammars”, *Information and Computation*, 194:1 (2004), 19–48.

[16] A. Okhotin, “Unambiguous Boolean grammars”, *Information and Computation*, 206 (2008), 1234–1247.

[17] A. Okhotin, “Fast parsing for Boolean grammars: a generalization of Valiant’s algorithm”, *Developments in Language Theory* (DLT 2010, London, Ontario, Canada, August 17–20, 2010), LNCS 6224, 340–351.

[18] A. Okhotin, P. Rondogiannis, “On the expressive power of univariate equations over sets of natural numbers”, *IFIP Intl. Conf. on Theoretical Computer Science* (TCS 2008, Milan, Italy, 8–10 September, 2008), IFIP vol. 273, 215–227.

[19] L. J. Stockmeyer, A. R. Meyer, “Word problems requiring exponential time”, *5th Annual ACM Symposium on Theory of Computing* (STOC 1973, Austin, USA, April 30–May 2, 1973), 1–9.

[20] L. G. Valiant, “General context-free recognition in less than cubic time”, *Journal of Computer and System Sciences*, 10:2 (1975), 308–314.

[21] K. W. Wagner, “The complexity of problems concerning graphs with regularities”, *Mathematical Foundations of Computer Science* (MFCS 1984, Prague, Czechoslovakia, September 3–7, 1984), LNCS 176, 544–552.