Online Context-Free Recognition in OMv Time

Bartłomiej Dudek
Institute of Computer Science, University of Wrocław, Poland

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland

Abstract

One of the classical algorithmic problems in formal languages is the context-free recognition problem: for a given context-free grammar and a length-n string, check if the string belongs to the language described by the grammar. Already in 1975, Valiant showed that this can be solved in $\tilde{O}(n^\omega)$ time, where ω is the matrix multiplication exponent. More recently, Abboud, Backurs, and Vassilevska Williams [FOCS 2015] showed that any improvement on this complexity would imply a breakthrough algorithm for the k-Clique problem. We study the natural online version of this problem, where the input string $w[1..n]$ is given left-to-right, and after having seen every prefix $w[1..t]$ we should output if it belongs to the language. The goal is to maintain the total running time to process the whole input. Even though this version has been extensively studied in the past, the best known upper bound was $O(n^3/\log^2 n)$. We connect the complexity of online context-free recognition to that of Online Matrix-Vector Multiplication, which allows us to improve the upper bound to $n^3/2^{\Omega(\sqrt{\log n})}$.

2012 ACM Subject Classification Theory of computation → Grammars and context-free languages

Keywords and phrases data structures, context-free grammar parsing, online matrix-vector multiplication

Digital Object Identifier 10.4230/LIPIcs.CPM.2024.13

1 Introduction

Context-free languages, introduced by Chomsky already in 1959 [3], are one of the basic concepts considered in formal languages, with multiple applications in programming languages [2], NLP [11], computational biology [6], and databases [13]. A context-free language is a language generated by a context-free grammar, meaning that each production rule is of the form $A \to \alpha$, where A is a non-terminal symbol, and α is a string of terminal and non-terminal symbols (possibly empty). It was already established by Chomsky [3] that, without decreasing the expressive power, we can assume that the productions are of the form $A \to a$ and $A \to BC$, where A, B, C are non-terminal symbols, and a is a terminal symbol. From an algorithmic point of view, the natural (and very relevant with respect to the possible applications) question is whether, given such a grammar G and a string $w[1..n]$, we can efficiently check if $w \in \mathcal{L}(G)$. A simple application of the dynamic programming paradigm shows that this is indeed possible in $O(n^3)$ time (ignoring the dependency on the size of the grammar). This is usually called the Cocke–Younger–Kasami (CYK) approach [4, 12, 25]. In 1975, Valiant [23] designed a non-trivial algorithm that solves this problem in $O(BM(n))$ time, where $BM(n)$ denotes the complexity of multiplying two (Boolean) $n \times n$ matrices. Plugging in the currently best known bounds, $BM(n) = O(n^\omega)$, where $\omega < 2.373$ [24]. See [9] for a somewhat more approachable description of Valiant's algorithm, and [19] for a very elegant simplification (achieving the same running time). This is of course a somewhat theoretical result, and given the practical nature of the problem it is not surprising that other approaches have been developed [5, 16, 17, 22], with high worst-case time complexities, but good behaviour on instances that are relevant in practice. However, the worst-case time complexity has not seen any improvement. In 2002, Lee [15] showed a conditional lower bound that provides some explanation for this lack of improvement: multiplying two (Boolean)

$n \times n$ matrices can be reduced to parsing a string of length $O(n^{1/3})$ for a grammar of size $O(n^2)$. This does exclude a combinatorial $O(gn^{3-\varepsilon})$ algorithm for parsing (more general problem than recognition), where g is the size of the grammar, but does not contradict the existence of e.g. $O(g^2 n)$ time algorithm. However, in 2015 Abboud, Backurs, and Vassilevska Williams [1] showed a more general conditional lower bound: even for constant-size grammars, any improvement on the complexity of Valiant's recognition algorithm implies a breakthrough for the well-known k-Clique problem.

In some applications, the input string $w[1..n]$ is given character-by-character, and for each prefix $w[1..i]$ we should decide if it belongs to $\mathcal{L}(G)$ before reading the next character. This is known as the online CFG recognition. The goal is to minimise the total time to process all the characters. It is not hard to adapt the CYK approach to work in $O(n^3)$ total time for this variant, but this seems difficult (or perhaps impossible) for Valiant's algorithm. Graham, Harrison, and Ruzzo [8] designed a (slightly) subcubic algorithm, and Rytter [19] further improved the complexity to $O(n^3/\log^2 n)$. Surprisingly, no further improvements were achieved. On the lower bound, it is known that on a Turing machine, $\Omega(n^2/\log n)$ steps are required [7, 20]. This is however quite far from the upper bound, and assumes a somewhat restricted model of computation, and brings the natural question of understanding if a faster algorithm exists.

As the complexity of the offline CFG recognition is known to be close to that of (Boolean) matrix multiplication, it is natural to seek a connection between the complexity of its online variant with the so-called online matrix multiplication. As a tool for unifying the complexities of different dynamic problems, Henzinger, Krinninger, Nanongkai, and Saranurak [10] introduced the Online Matrix-Vector Multiplication problem:

▶ **Definition 1** (Online Matrix-Vector Multiplication (OMv)). *Given a matrix $M \in \{0,1\}^{n\times n}$, and a sequence of vectors $v_1, \ldots, v_n \in \{0,1\}^n$, the task is to output Mv_i before seeing v_{i+1}, for all $i = 1, \ldots, n-1$.*

and conjectured that no $O(n^{3-\epsilon})$ time algorithm exists (with the best known upper bound at the time being $O(n^3/\log^2 n)$):

▶ **Hypothesis 2** (OMv Hypothesis [10]). *Every (randomized) algorithm solving OMv must take total time $n^{3-o(1)}$.*

Surprisingly, Larsen and Williams [14] were soon able to construct a faster $n^3/2^{\Omega(\sqrt{\log n})}$ time algorithm. This does not refute the OMv hypothesis, but significantly improves the known upper bound, essentially by saying that we can shave any number of logarithms from the time complexity.

▶ **Theorem 3** ([14]). *There exists a randomized algorithm for OMv that runs in total $n^3/2^{\Omega(\sqrt{\log n})}$ time and succeeds with high probability*[1].

This suggests the possibility of leveraging the progress on the complexity of Online Matrix-Vector Multiplication to improve the complexity of online CFG recognition to improve on the $O(n^3/\log^2 n)$ time complexity from 1985.

[1] By succeeding with high probability we mean that there exists a constant $c > 0$ such that the algorithm succeeds with probability at least $1 - 1/n^c$.

Our contribution. We show that it is possible to use efficient OMv multiplication to speed up online context-free recognition:

▶ **Theorem 4.** *Let G be a context-free grammar, and w be a length-n string, revealed one character at a time. There exists a randomized algorithm that determines, after having seen $w[t]$, if $w[1..t] \in \mathcal{L}(G)$, in $n^3/2^{\Omega(\sqrt{\log n})}$ total time and succeeds with high probability.*

Our solution is based on the classical CYK dynamic-programming approach from 1960s [4, 12, 25] in which we calculate the set of non-terminals deriving each of the infixes of w. In order to avoid the $O(n^2)$ time for processing a new character $w[t]$, we maintain a division of the current prefix into segments of lengths that are powers of 2 present in the binary representation of t. For each of the segments, we build a structure responsible for processing suffixes $w[i..t]$ that start within the segment and end at t. We extensively use the approach from Theorem 3 for OMv, with a slight adaptation to matrices that grow in time. More precisely, we show that we can process a sequence of vectors $v_1, q_1, v_2, q_2, \ldots$ where $|q_i| = i$ in which we need to calculate $(v_1, \ldots, v_i) \times q_i$ online, before seeing v_{i+1}. This requires one more step of dividing the range of columns into segments of lengths that are powers of 2, and applying the structure from Theorem 3 for each of the segments separately. This results in the same running time as in the standard OMv problem, in which the matrix we multiply with does not change.

We note that our algorithm does not need to know the value of n in advance. In fact, our proof of Theorem 4 can be modified to show that the amortised time for processing the t-th character is $O(t^2/2^{\Omega(\sqrt{\log t})})$, so in particular after having seen $w[t]$ we know whether $w[1..t] \in \mathcal{L}(G)$, with the total time spent on $w[1], w[2], \ldots, w[t]$ being $O(t^3/2^{\Omega(\sqrt{\log t})})$, for every $t = 1, 2, \ldots$.

2 Preliminaries

Consider a context-free grammar $G = (V_N, V_T, P, S)$. Without loss of generality we assume that G is in Chomsky normal form [3, 21], that is every production in P is either $A \to BC$ or $A \to c$ for $A, B, C \in V_N$ and $c \in V_T$. By $v \stackrel{\star}{\Rightarrow} s$ we denote that string s can be derived from non-terminal v in the grammar G.

We are given a string w of length n character-by-character and for each $t = 1..n$ need to decide if the string $w[1..t]$ belongs to $\mathcal{L}(G)$ or not. For every t, the answer should be provided before reading the $(t+1)$-th character and we call such a procedure *online*. Our algorithm will compute the set of all non-terminals that produce every infix of w: $U[i, j] = \{v \in V_N : v \stackrel{\star}{\Rightarrow} w[i..j]\}$. Then the t-th bit of the output is whether S belongs to $U[1, t]$ or not.

Our approach has polynomial dependence on the size of the grammar G, which we omit while stating the complexity of the parsing algorithm.

In the analysis of our algorithm we will consider sums of non-constant number of distinct expressions containing the Ω function. Unless stated otherwise, all the Ωs within one sum correspond to the same function, namely there exists one constant bounding all the expressions at the same time. An example of such sum appears in the following lemma that will be useful in the next section:

▶ **Lemma 5.** *For any constant $a > 0$, we have $\sum_{k=0}^{\log n} 2^{ak-\Omega(\sqrt{k})} = n^a/2^{\Omega(\sqrt{\log n})}$.*

Proof. Let $t = \log n$. As we discussed before, various Ω functions correspond to one particular Ω function, which means that we can read the expression $(*) = \sum_{k=0}^{t} 2^{ak-\Omega(\sqrt{k})}$ as: there exists a constant $c > 0$ such that $(*) \leqslant \sum_{k=0}^{t} 2^{ak-c\sqrt{k}}$. First we show for which $0 \leqslant k < t$ we can upper bound the k-th summand by the last element from the sum:

$$\begin{aligned} ak - c\sqrt{k} &< at - c\sqrt{t} \\ &\Updownarrow \\ c(\sqrt{t} - \sqrt{k}) &< a(t-k) = a(\sqrt{t} - \sqrt{k})(\sqrt{t} + \sqrt{k}) \\ &\Updownarrow \\ \frac{c}{a} - \sqrt{t} &< \sqrt{k} \end{aligned}$$

So in particular, for $k \geqslant k_0 = (\frac{c}{a})^2$ we have that $ak - c\sqrt{k} \leqslant at - c\sqrt{t}$. For $k < k_0$ we have $2^{ak-c\sqrt{k}} < 2^{ak_0}$, so:

$$(*) \leqslant k_0 \cdot 2^{ak_0} + \sum_{k=k_0}^{t} 2^{at-c\sqrt{t}} \leqslant O(1) + t \cdot 2^{at-c\sqrt{t}} = O(2^{at-0.9c\sqrt{t}}) = 2^{at-\Omega(\sqrt{t})}. \qquad \blacktriangleleft$$

3 Parsing context-free grammars online

Our algorithm processes characters from the input one-by-one. While processing the t-th character it has already computed $U[i,j]$ for $1 \leqslant i \leqslant j < t$ and needs to compute $U[i,t]$ for $1 \leqslant i \leqslant t$. We maintain a division of the interval $[1..(t-1)]$ into $c = O(\log t)$ intervals: $[e_1, e_2), [e_2, e_3), \ldots, [e_c, e_{c+1})$ where $e_1 = 1, e_{c+1} = t$ and lengths of the intervals are exactly the powers of 2 in the binary representation of $t-1$, in the decreasing order. On a high level, for the j-th interval there is a data structure X_j responsible for computing $U[i,t]$ for $e_j \leqslant i < e_{j+1}$, based on the outputs from $X_{j'}$ for $j' > j$. We call such a data structure a *process*. We say that the *size* of process X_j is the length of the interval it corresponds to, that is $|[e_j, e_{j+1})| = e_{j+1} - e_j$. The processes are created and removed following the binary representation of t, and a process for interval $[a, a+2^k)$ exists only for $t = a+2^k, \ldots a+2^{k+1}-1$. In the following theorem we describe the calculations performed in each of the processes.

▶ **Theorem 6.** *Let $\mathcal{I} = [p, p+s)$ be an interval of positions from w. Consider the following sequence Q of at most s queries $Q_{p+s}, Q_{p+s+1}, \ldots$: in the t-th query we are given set $Q_t = \{(i,v) : v \overset{\star}{\Rightarrow} w[i,t], i \in [p+s, t]\}$ and need to compute $A_t = \{(i,v) : v \overset{\star}{\Rightarrow} w[i,t], i \in \mathcal{I}\}$.*

There exists a randomized algorithm answering online all queries from Q in total $s^3/2^{\Omega(\sqrt{\log s})}$ randomized time that succeeds with high probability.

Before we prove the above theorem, we show how to apply it to obtain the algorithm for parsing context-free grammars online.

▶ **Theorem 4.** *Let G be a context-free grammar, and w be a length-n string, revealed one character at a time. There exists a randomized algorithm that determines, after having seen $w[t]$, if $w[1..t] \in \mathcal{L}(G)$, in $n^3/2^{\Omega(\sqrt{\log n})}$ total time and succeeds with high probability.*

Proof. We show that Algorithm 1 correctly parses all prefixes of w online, in the desired time complexity. First, we show that the operations in Algorithm 1 satisfy the requirements on queries described in Theorem 6. Indeed, we always create a process β with $\mathcal{I} = [t+1-2^r, t+1)$ and the subsequent queries are $Q_{t+1}, Q_{t+2}, \ldots$, so in particular the first query concerns the position right after the end of $\mathcal{I}$, as required. Observe that due to line 7 the sequence of sizes

Algorithm 1 Parsing context-free grammar online.

Input: Context-free grammar $G = (V_N, V_T, P, S)$

```
1:  B := [ ]                                   ▷ 1-based list of processes B^1, B^2, ...
2:  for t = 1, 2, ... do
3:      Q_t := {(t, v) : (v → w[t]) ∈ P}
4:      for j = |B| to 1 do
5:          A := B^j.query(Q_t)
6:          Q_t := Q_t ∪ A
7:      let r be maximal such that Σ_{i=0}^{r-1} |B^{|B|-i}| = 2^r − 1
8:      remove the last r processes from B
9:      add new_process(I = [t + 1 − 2^r, t + 1)) to the end of B
10:     output whether (1, S) ∈ Q_t
```

of the processes follows the binary representation of t so we create a process of size 2^k for t such that $t \equiv 2^k \pmod{2^{k+1}}$ and the last query that we possibly process at β is Q_{t+2^k}, so β is queried at most $t + 2^k - t = 2^k = |\beta|$ times.

Now we calculate the complexity of the algorithm. We create a new process of size 2^k exactly $\lfloor \frac{n+2^k}{2^{k+1}} \rfloor < n/2^k$ times. Each process of size 2^k answers at most 2^k queries so we can directly apply Theorem 6 to bound the total running time of preprocessing and all queries processed by the process. Hence the total running time of the algorithm is upper bounded by:

$$\sum_{k=0}^{\log n} \frac{n}{2^k} \cdot \left(2^k\right)^3 / 2^{\Omega(\sqrt{k})} = n \cdot \sum_{k=0}^{\log n} 2^{2k - \Omega(\sqrt{k})} = n^3 / 2^{\Omega(\sqrt{\log n})}.$$

The last step follows by Lemma 5 and the claim holds. ◀

Proof of Theorem 6

In order to prove Theorem 6, we need to introduce some notation and insights following Rytter's variant of the Valiant's offline parser of context-free grammars [19]. We will operate on matrices of binary relations over the set of non-terminals V_N and we call such matrices *relational*. Formally, every element of a relational matrix is of the form $\{0,1\}^{V_N \times V_N}$. To simplify the notation, our relational matrices will be indexed by intervals $\mathcal{J}_1, \mathcal{J}_2 \subseteq [1, n]$ of consecutive numbers, corresponding to substrings of the input string w. We define $\otimes$-multiplication of matrices A, B with indices $\mathcal{J}_a \times \mathcal{J}_c$ and $\mathcal{J}_c \times \mathcal{J}_b$ respectively, as:

$$(A \otimes B)[i,j]^{X,Y} = \bigvee_{\substack{k \in \mathcal{J}_c \\ Z \in V_N}} A[i,k]^{X,Z} \cdot B[k,j]^{Z,Y} \quad \text{for } i \in \mathcal{J}_a, j \in \mathcal{J}_b, X, Y \in V_N$$

Similarly, we define *relational vectors* as vectors of subsets of V_N, that is their elements are of the form: $\{0,1\}^{V_N}$, and matrix-vector product $M \otimes F$ of matrix M (indexed with $\mathcal{J}_a \times \mathcal{J}_c$) and vector F (indexed with $\mathcal{J}_c$) as:

$$(M \otimes F)[i]^X = \bigvee_{\substack{k \in \mathcal{J}_c \\ Z \in V_N}} M[i,k]^{X,Z} \cdot F[k]^Z \quad \text{for } i \in \mathcal{J}_a, X \in V_N$$

▶ **Lemma 7** ([18]). *We can compute relational matrix-matrix $\otimes$-product in $|V_N|^3$ multiplications of two Boolean matrices and relational matrix-vector $\otimes$-product in $|V_N|^2$ multiplications of a Boolean matrix and a vector. The Boolean matrices and vectors that we multiply have the same size as the relational ones.*

Proof. By definition of $\otimes$-product, in order to multiply two relational matrices we iterate over all triples of X, Y, Z of non-terminals, create Boolean matrices $A'^{X,Z}, B^{Z,Y}$ where $A'^{X,Z}[i,j] = A[i,j]^{X,Z}$ and $B'^{Z,Y}[i,j] = B[i,j]^{Z,Y}$ and calculate $A' \cdot B'$ using the standard Boolean $(+,\cdot)$-product. Then $(A \otimes B)[i,j]^{X,Y} = \bigvee_{Z \in V_N} (A'^{X,Z} \cdot B'^{Z,Y})[i,j]$.

Matrix-vector $\otimes$-multiplication can be calculated analogously. ◀

Now we are able to show the main theorem of this section.

▶ **Theorem 6.** *Let $\mathcal{I} = [p, p+s)$ be an interval of positions from w. Consider the following sequence Q of at most s queries $Q_{p+s}, Q_{p+s+1}, \ldots$: in the t-th query we are given set $Q_t = \{(i,v) : v \overset{\star}{\Rightarrow} w[i,t], i \in [p+s, t]\}$ and need to compute $A_t = \{(i,v) : v \overset{\star}{\Rightarrow} w[i,t], i \in \mathcal{I}\}$.*

There exists a randomized algorithm answering online all queries from Q in total $s^3/2^{\Omega(\sqrt{\log s})}$ randomized time that succeeds with high probability.

Recall that $G = (V_N, V_T, P, S)$ is the considered grammar. Whenever we refer to $w[i, i-1]$ for any i, we mean an empty string.

Preprocessing. During the preprocessing phase we first run Rytter's algorithm [19] on $w[p..p+s-1]$ and compute $U[i,j]$ for all $p \leqslant i \leqslant j < p+s$ in $O(s^\omega)$ time[2]. Based on that we define a relational matrix V with rows and columns $p..(p+s)$ by setting

$$V[i,j]^{X,Y} = 1 \iff \exists_{Z \in V_N} \Big((X \to ZY) \in P \wedge Z \overset{\star}{\Rightarrow} w[i..j-1] \Big) \quad \text{for } p \leqslant i \leqslant j \leqslant p+s$$

Informally, this means that we can extend "to the left" every infix of w that starts at position j and can be derived from Y to an infix that starts at position i, ends at the same position and that can be derived from X. For empty infixes we set $V[i,i]^{X,Y} = 1 \iff X = Y$. When we do not specify the value of some entries of a matrix, it means that there are all zeros in that entry. For instance, for $i > j$ in V we have $V[i,j]^{X,Y} = 0$ for all $X, Y \in V_N$.

Now we calculate $V^\star = V^s$ with exponentiation by squaring, using $\otimes$-product at every step in total $O(s^\omega \log s) = \tilde{O}(s^\omega)$ time, by Lemma 7. Observe that $V^\star$ describes all possibilities of extending an infix "to the left" at most s times. As $j - i \leqslant s$, we never need more than s steps to extend an infix starting at position j to an infix starting at position i and then:

$$V^\star[i,j]^{X,Y} = 1 \iff \exists \underset{\substack{k_1=i, k_r=j \\ Z_1,\ldots,Z_r \in V_N \\ Z_1=X, Z_r=Y}}{k_1 < \ldots < k_r} \Big(\forall_{1 \leqslant e < r} V[k_e, k_{e+1}]^{Z_e, Z_{e+1}} = 1 \Big) \quad \text{for } p \leqslant i \leqslant j \leqslant p+s$$

See Figure 1 for an illustration.

Invariant. During the process of answering queries, before receiving a subsequent query Q_t, we maintain a relational matrix H with similar properties as V, with rows $p..(p+s)$, but with columns $(p+s)..t$, that is:

$$H[i,j]^{X,Y} = 1 \iff \exists_{Z \in V_N} \Big((X \to ZY) \in P \wedge Z \overset{\star}{\Rightarrow} w[i..j-1] \Big) \quad \text{for } p \leqslant i \leqslant p+s \leqslant j \leqslant t$$

[2] In [19] is computed $VALID(k,\ell) = \bigcup_{k \leqslant i \leqslant j \leqslant \ell} \{(A,i,j) : A \in U[i,j]\}$.

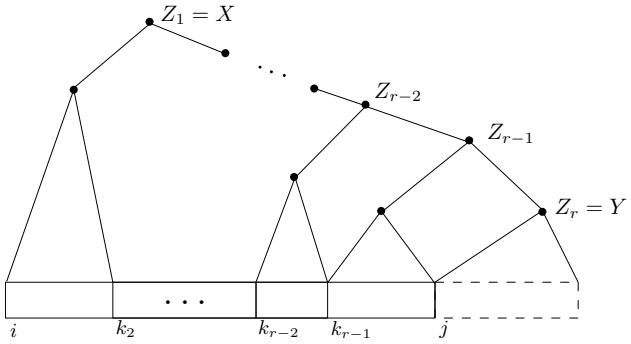

Figure 1 Illustration of the definition of $V^\star$. Note that we do not specify the endpoint of the last string, starting at position j and derived from Y, because we are only interested in the possible extensions "to the left" from such a string.

This matrix also describes extensions "to the left", but from an infix starting at position $j \geqslant p + s$ to an infix starting at position $i \leqslant p + s$. In order to satisfy the invariant, at the end of preprocessing we initialize $H[i, p + s] = V[i, p + s]$ for $p \leqslant i \leqslant p + s$.

Query. From the input set Q_t we create a relational vector $F[(p + s)..t]$ such that

$$F[j]^Y = 1 \iff Y \stackrel{\star}{\Rightarrow} w[j..t] \iff (j, Y) \in Q_t.$$

Let $A = H \otimes F$. Then $A[i]^X = 1 \implies X \stackrel{\star}{\Rightarrow} w[i..t]$ for $p \leqslant i \leqslant p + s$. However, this is not an equivalence yet, because we need to consider a larger number of extensions "to the left" using infixes fully contained in $\mathcal{I} = [p, p + s)$. For that we use matrix $V^\star$ and compute $A' = V^\star \otimes A$. Then we have $A'[i]^X = 1 \iff X \stackrel{\star}{\Rightarrow} w[i..t]$ for $p \leqslant i \leqslant p + s$ and we can construct the desired set A_t that can be returned from the procedure. As the last step of processing the query, we update matrix H by adding $(t + 1)$-th column by definition: $H[i, t + 1]^{X,Y} = 1 \iff \exists_{Z \in V_N} (X \to ZY) \in P \wedge A'[i]^Z = 1$.

Running time. The only operations that can take more than $O(s)$ time in the above procedure are the matrix-vector $\otimes$ multiplications $H \otimes F$ and $V^\star \otimes A$. Recall that by Lemma 7 it suffices to show how to perform these operations efficiently for matrices and vectors over the Boolean semiring, not the relational ones. In the case of $V^\star \otimes A$ we have one matrix $V^\star$ subsequently multiplied by different vectors A, so we can directly apply Theorem 3 and process online all the queries in total $s^3/2^{\Omega(\sqrt{\log s})}$ time.

For the multiplications $H \otimes F$ we need a slightly different approach, because the matrix H changes in time. Similarly as in Algorithm 1 we will divide the columns of H into intervals following the binary representation of the width of H and split H into a number of smaller square matrices. For each of the small matrices we will use the algorithm for OMv from Theorem 3.

▶ **Lemma 8.** *Consider the sequence of at most s operations, where in the j-th one we are given a binary vector v_j of length s and a binary vector q_j of length j and need to calculate $x_j = M_j \cdot q_j$ where M_j is the matrix with s rows and columns $v_1, \ldots, v_j$. There exists a randomized algorithm answering online all the queries in total $s^3/2^{\Omega(\sqrt{\log s})}$ time that succeeds with high probability.*

Proof. Similarly as in Algorithm 1, we maintain a partition of the interval $[1..j]$ into $c = O(\log j)$ intervals: $\mathcal{E}(j) = [e_1, e_2), [e_2, e_3), \ldots, [e_c, e_{c+1})$ where $e_1 = 1, e_{c+1} = j + 1$ and lengths of the intervals follow the binary representation of j, with $[e_1, e_2)$ being the largest one. Intervals correspond to subranges of columns of M_j and for an interval of length 2^k we divide its $s \times 2^k$ submatrix into $s/2^k$ square matrices of size $2^k \times 2^k$. For each such matrix we create a data structure for OMv multiplication, by Theorem 3.

In order to process a query, we first add the new column v_j, update the structure of intervals from $\mathcal{E}(j-1)$ to $\mathcal{E}(j)$ and run preprocessing for each of the newly-created matrices. To answer the query we divide q_j according to $\mathcal{E}(j)$ into vectors $q_j^1, \ldots, q_j^c$ and multiply each vector q_j^i by all the matrices of the same size as q_j and combine the results in one vector y_j of length s, see Figure 2. Then $x_j = \bigvee_{i=1}^{c} y_i$.

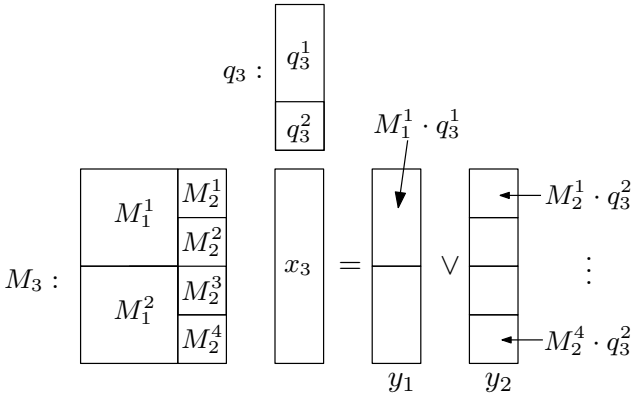

Figure 2 Example of calculating x_3 based on the results from multiplicating square matrices M_i^z by vector q_3^i for $z \in [1, s/|q_3^i|]$.

The correctness of the approach is immediate and now we need to calculate the total running time. While adding new columns to the considered matrix, we create a new interval of length 2^k for j such that $j \equiv 2^k \pmod{2^{k+1}}$, so in total less than $s/2^k$ times. We divide every interval into $s/2^k$ matrices of size $2^k \times 2^k$ and for each of them we create an OMv data structure that answers at most 2^k queries. By Theorem 3 we can process online all the queries for a single matrix in $2^{3k}/2^{\Omega(\sqrt{k})}$ total time. This gives us the following total running time of processing all the queries:

$$\sum_{k=0}^{\log s} s/2^k \cdot s/2^k \cdot (2^k)^{3-\Omega(\sqrt{k})} = s^2 \cdot \sum_{k=0}^{\log s} 2^{k-\Omega(\sqrt{k})} = s^3/2^{\Omega(\sqrt{\log s})}$$

where the last step follows from Lemma 5. ◀

Finally, the total running time of our algorithm is $\tilde{O}(s^\omega)$ for the preprocessing and $s^3/2^{\Omega(\sqrt{\log s})}$ for answering all the queries, which gives $s^3/2^{\Omega(\sqrt{\log s})}$ total time. This concludes the proof of Theorem 6.

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant's parser. In *FOCS*, pages 98–117. IEEE Computer Society, 2015.

2 Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley series in computer science / World student series edition. Addison-Wesley, 1986.

3 Noam Chomsky. On certain formal properties of grammars. *Inf. Control.*, 2(2):137–167, 1959.
4 John Cocke and Jacob T. Schwartz. Programming languages and their compilers: Preliminary notes (technical report) (2nd revised ed. *Technical report, CIMS, NYU*, 1970.
5 Shay B. Cohen, Giorgio Satta, and Michael Collins. Approximate PCFG parsing using tensor decomposition. In *HLT-NAACL*, pages 487–496. The Association for Computational Linguistics, 2013.
6 Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme J. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.
7 Hervé Gallaire. Recognition time of context-free languages by on-line turing machines. *Inf. Control.*, 15(3):288–295, 1969.
8 Susan L. Graham, Michael A. Harrison, and Walter L. Ruzzo. An improved context-free recognizer. *ACM Trans. Program. Lang. Syst.*, 2(3):415–462, 1980.
9 M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1978.
10 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, pages 21–30. ACM, 2015.
11 Dan Jurafsky and James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, 2009.
12 Tadao Kasami. An efficient recognition and syntax algorithm for context-free language. *Technical Report AFCRL-65-758, Air Force Cambridge Research Lab, Bed-ford, MA.*, 1965.
13 Flip Korn, Barna Saha, Divesh Srivastava, and Shanshan Ying. On repairing structural problems in semi-structured data. *Proc. VLDB Endow.*, 6(9):601–612, 2013.
14 Kasper Green Larsen and R. Ryan Williams. Faster online matrix-vector multiplication. In *SODA*, pages 2182–2189. SIAM, 2017.
15 Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *J. ACM*, 49(1):1–15, 2002.
16 Adam Pauls and Dan Klein. K-best A* parsing. In *ACL/IJCNLP*, pages 958–966. The Association for Computer Linguistics, 2009.
17 Alexander M. Rush, David A. Sontag, Michael Collins, and Tommi S. Jaakkola. On dual decomposition and linear programming relaxations for natural language processing. In *EMNLP*, pages 1–11. ACL, 2010.
18 Wojciech Rytter. Fast recognition of pushdown automaton and context-free languages. *Information and Control*, 67(1-3):12–22, 1985.
19 Wojciech Rytter. Context-free recognition via shortest paths computation: A version of Valiant's algorithm. *Theor. Comput. Sci.*, 143(2):343–352, 1995.
20 Joel I. Seiferas. A simplified lower bound for context-free-language recognition. *Inf. Control.*, 69(1-3):255–260, 1986.
21 Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.
22 Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. Parsing with compositional vector grammars. In *ACL (1)*, pages 455–465. The Association for Computer Linguistics, 2013.
23 Leslie G. Valiant. General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.*, 10(2):308–315, 1975.
24 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *SODA*, pages 3792–3835. SIAM, 2024.
25 Daniel H. Younger. Recognition and parsing of context-free languages in time n^3. *Inf. Control.*, 10(2):189–208, 1967.