SUBLINEAR-SPACE RECOGNITION FOR BOOLEAN GRAMMARS

WILL HOWE, GRANT HUFFMAN, JENSEN KOHLMEYER, AND LIAM KRUER

Abstract. For every fixed Boolean grammar in the standard language family considered by Okhotin, we give a deterministic membership algorithm using $O_G(n^{2/3}(\log(n+2))^{2/3})$ working space on an input of length n. In particular, the whole family is contained in DSPACE($n^{3/4}$), answering Okhotin's Problem 4 affirmatively. The construction partitions substring endpoints into blocks of size b. It represents the parsing recurrence by a constant-fan-in circuit with $O_G(b^2)$-bit values and dependency height $O(n/b+1)$. Splits in the endpoint blocks are resolved locally, and an ordering of the remaining splits controls the height even when a split is highly unbalanced. Applying the Cook–Mertz tree-evaluation algorithm to the implicitly unfolded circuit gives the space bound.

hexagon:2611.00269v1

1. Introduction

Boolean grammars extend context-free grammars by allowing conjunction and negation in production rules [1]. Their standard recognition algorithms exploit the dependence of a substring's syntactic properties on properties of its proper substrings. In his 2007 list of open problems, Okhotin asked whether the entire family of Boolean languages is contained in DSPACE($n^{1-\varepsilon}$) for some constant $\varepsilon > 0$ [2, Section 3.4, Problem 4]. The question concerns working space, permits a different recognizer for each fixed grammar, and allows a tradeoff in time.

Theorem 1.1. *Let G be a fixed Boolean grammar in the standard family of* [2]. *There is a deterministic algorithm deciding membership in $L(G)$, with read-only access to its input, using*

$$O_G\Big(n^{2/3}(\log(n+2))^{2/3}\Big)$$

bits of working space on every input of length n.

Corollary 1.2. *The family of Boolean languages is contained in* DSPACE($n^{3/4}$). *More generally, it is contained in* DSPACE($n^{1-\varepsilon}$) *for every fixed $0 < \varepsilon < 1/3$.*

All constants implicit in O_G may depend on G. The proof uses the normal-form substring recurrence together with Cook–Mertz tree evaluation [5]. Williams's expanded author version [6, Theorem 2.2 and Appendix A] makes explicit the variable-height trees and implicit access used here. The contribution is a block circuit for the grammar recurrence whose value length and dependency height can be balanced.

2. Normal form and the interval recurrence

We use the standard Boolean-language family of [2, Definitions 6 and 7]. In that presentation, the semantics are defined through strongly unique solutions of the associated language equations. A grammar in this family admits an equivalent binary normal form [1, 3]. Its nonterminal set N and terminal alphabet Σ are finite, and its rules have the forms

$$\begin{aligned} &A \longrightarrow B_1C_1 \,\&\, \cdots \,\&\, B_pC_p \,\&\, \neg D_1E_1 \,\&\, \cdots \,\&\, \neg D_qE_q \,\&\, \neg\varepsilon, \qquad p \geq 1,\ q \geq 0, \\ &A \longrightarrow a, \qquad a \in \Sigma. \end{aligned} \tag{2.1}$$

Date: November 7, 2026.

Key words and phrases. Boolean grammars, conjunctive grammars, deterministic space, tree evaluation.

Prepared with AI assistance from OpenAI Codex in mathematical exploration, checking, and manuscript preparation.

There may also be a rule $S \longrightarrow \varepsilon$, with the start symbol S absent from all right-hand sides. Membership of the empty word is handled separately. The condition $\neg\varepsilon$ in (2.1) is automatic on nonempty substrings.

Fix an input $w = w_1 \cdots w_n$, where $n \geq 1$. For $A \in N$ and $0 \leq i < j \leq n$, define

$$X_A(i,j) = 1 \quad \Longleftrightarrow \quad w_{i+1} \cdots w_j \in L_G(A).$$

Let $\mathcal{P} \subseteq N \times N$ consist of the ordered nonterminal pairs that occur in positive or negative binary conjuncts. For each $(B,C) \in \mathcal{P}$, put

$$Y_{BC}(i,j) = \bigvee_{i<k<j} \big(X_B(i,k) \wedge X_C(k,j)\big). \tag{2.2}$$

For $j - i \geq 2$, there is a fixed Boolean function $F_A : \{0,1\}^{\mathcal{P}} \to \{0,1\}$ such that

$$X_A(i,j) = F_A\big((Y_{BC}(i,j))_{(B,C)\in\mathcal{P}}\big). \tag{2.3}$$

Explicitly, F_A disjoins the alternatives for A, conjoining positive pair flags and complemented negative pair flags within each alternative. For $j - i = 1$, the value is determined by the terminal rules.

Equations (2.2) and (2.3) are the usual binary-normal-form recognition recurrence [3, Section 2, pp. 4–5]. Every interval used on the right of (2.2) is strictly shorter. In particular, the negation in (2.3) is applied after the complete disjunction over splits has been computed.

3. The block circuit

Choose an integer $b \geq 1$ and partition the endpoint set $\{0, \ldots, n\}$ into consecutive blocks

$$I_0, I_1, \ldots, I_{m-1}, \qquad m = \left\lceil \frac{n+1}{b} \right\rceil,$$

each of size at most b, with all but the last of size b. For $0 \leq r \leq s < m$, a chart word $T_{r,s}$ encodes

$$\big(X_A(i,j) : A \in N,\ i \in I_r,\ j \in I_s,\ i < j\big).$$

Fix a coordinate order and use zero padding so that every value in the circuit has the same bit length

$$W = c_G b^2,$$

where the fixed positive integer c_G is large enough for either a chart word or a word with one flag for each $(B,C,i,j) \in \mathcal{P} \times I_r \times I_s$. Padding coordinates are ignored by each gate function.

3.1. **Diagonal words.** For each r, $T_{r,r}$ is a leaf whose value can be generated on demand. Compute all intervals with endpoints in I_r in increasing length, using (2.2) and (2.3). Every smaller interval needed remains in the same block. Store the partial chart and a constant number of flags. The resulting space requirement is

$$O_G(b^2 + \log(n+2)). \tag{3.1}$$

A terminal query uses a logarithmic-size position counter and a scan of the read-only input.

3.2. **Proper intermediate blocks.** For $r < t < s$, let $Q_{r,s,t}$ be a gate with inputs $T_{r,t}$ and $T_{t,s}$. Its output coordinate (B,C,i,j) is

$$\bigvee_{k\in I_t} \big(X_B(i,k) \wedge X_C(k,j)\big). \tag{3.2}$$

Every such k is an eligible split because the blocks are consecutive and $r < t < s$. Both input chart words have strictly smaller block span than $s - r$. The output is computed by loops over the designated coordinates and split positions, in the space (3.1).

To aggregate these products, fix $r < s$, put $d = s - r$, and list the interior indices as $t_1, \dots, t_{d-1}$ in increasing order of $\min(t - r, s - t)$, breaking ties by increasing t. Let $U_{r,s,d}$ be the zero word and define

$$U_{r,s,a} = Q_{r,s,t_a} \vee U_{r,s,a+1}, \qquad 1 \le a < d, \tag{3.3}$$

where $\vee$ acts componentwise. The aggregate is $U_{r,s,1}$. For $d = 1$, this is the zero leaf, since there are no proper intermediate blocks.

3.3. **Completing a cross-block word.** The gate for $T_{r,s}$, where $r < s$, takes the three inputs

$$U_{r,s,1}, \quad T_{r,r}, \quad T_{s,s}.$$

It fills its output coordinates in increasing interval length. For a cell $(i, j) \in I_r \times I_s$ and each $(B, C) \in \mathcal{P}$, start with the aggregate flag and include by OR exactly those splits with $i < k < j$ and $k \in I_r \cup I_s$.

If $k \in I_r$, then $X_B(i, k)$ is in the supplied diagonal word, and $X_C(k, j)$ is an already computed cell of the current output. If $k \in I_s$, then $X_B(i, k)$ is an already computed output cell, and $X_C(k, j)$ is in the other diagonal word. In both cases the output cell used has strictly smaller interval length. Once all pair flags for (i, j) have been accumulated, apply (2.3). A length-one cell is filled from the terminal rules, including when its endpoints lie in adjacent blocks.

This is a total Boolean function of its three input words. Even when the inputs are arbitrary bit strings, the local computation has no circular dependency. The output chart, temporary flags, and loop counters fit in (3.1). To obtain length order, one may loop through all global lengths and inspect the cells with endpoints in $I_r \times I_s$. Only space is being bounded.

Lemma 3.1. *The circuit is acyclic and computes all chart words correctly. Every gate has fan-in at most three and is evaluable on arbitrary Boolean arguments in the space* (3.1).

Proof. Product gates use chart words of smaller block span. At a fixed span, the aggregate gates form the finite chain (3.3). The cross-block gate uses that aggregate and two diagonal words. These observations give acyclicity.

Induct on the block span. The diagonal case is the ordinary recurrence. For a cross-block word, induct also on interval length within the word. The possible split positions partition into the left endpoint block, the proper intermediate blocks, and the right endpoint block. The product and aggregate gates compute exactly the intermediate contribution. The local procedure supplies exactly the two endpoint contributions using known shorter cells. Thus the pair flags agree with (2.2), and (2.3) gives the correct membership bits. The descriptions of the gates give the fan-in and space bounds. □

3.4. **Height of the circuit.** Height counts edges on a path from a gate toward a leaf.

Lemma 3.2. *Every path from $T_{r,s}$ to a leaf has length at most $4(s - r)$. The circuit required for the membership bit $X_S(0, n)$ has height $O(m)$.*

Proof. The diagonal words are leaves. Suppose $d = s - r \ge 1$. Consider a path that leaves $T_{r,s}$ through the aggregate and reaches Q_{r,s,t_a}. Put

$$u = \min(t_a - r, s - t_a).$$

There are at most two interior blocks at each distance from the nearer endpoint. The prescribed ordering therefore gives $a \le 2u$. The path to either input chart word of the product uses $a + 2$ edges. Since $u \ge 1$,

$$a + 2 \le 2u + 2 \le 4u.$$

Either input chart word has block span at most $d - u$. By induction, the remainder of the path has length at most $4(d - u)$, so the total is at most $4d$.

The route from $T_{r,s}$ to the aggregate zero leaf has length d, and its direct routes to diagonal leaves have length one. These exhaust the possibilities and complete the induction. An additional constant-fan-in gate can move the desired bit into the first coordinate of the output, increasing the height by only a constant. □

4. Space-efficient uniform evaluation

We use the following established evaluation bound. For a tree with W-bit node values, maximum height H, and fan-in at most three, Cook–Mertz evaluation uses

$$O\big(W + H\log(W+2)\big) \tag{4.1}$$

bits [5, Theorem 7]. The bound applies to variable-height trees and smaller arities, as described in the expanded author version of [6, Theorem 2.2 and Appendix A].

Here the tree is represented implicitly by unfolding the circuit backward from the output. Multiple occurrences of a circuit gate are treated as separate tree nodes. This preserves the output and the height bound.

Lemma 4.1. *The unfolded block circuit can be evaluated in deterministic working space*

$$O_G\big(W + H\log(W+2) + \log(n+2)\big).$$

Proof. We verify the access operations needed by the evaluator. A circuit gate name consists of its type and a constant number of block indices or aggregation ranks. It requires $O(\log(n+2))$ bits. Given a gate name and a child choice, the child name is computable in the same space. In particular, the rank-a interior index can be found by enumerating distances from the nearer endpoint and emitting the two corresponding indices in increasing order, with a coincident midpoint emitted once.

The evaluator stores a tree node as its sequence of child choices from the root. To recover the associated circuit gate, replay that sequence from the output, keeping only the current gate name and a position counter. This uses $O(\log(n+2))$ additional bits beyond the stored path.

We give explicit parameters for the packed interpolation in [6, Appendix A]. Set $D = 3$ and supply zero-valued dummy leaves at gates of smaller arity, with those arguments ignored. Let K be a field of characteristic two and choose

$$\ell = \lceil \log_2\big(8D(W+1)\big) \rceil, \qquad |K| = 2^{2\ell}, \qquad t = \left\lceil \frac{W}{\ell} \right\rceil.$$

Using the polynomial-basis representation of K, let S be the subset whose last ℓ binary coordinates are zero. Thus $|S| = 2^\ell$, and an element of S encodes ℓ bits. Extend each gate to words of $t\ell$ bits by ignoring the extra input bits and zeroing the extra output bits. Its j-th output block defines a total function $g_{v,j} : S^{Dt} \to S$, for $1 \le j \le t$.

For $a \in S$, define the Lagrange polynomial

$$L_a(z) = \prod_{c \in S \setminus \{a\}} \frac{z-c}{a-c}.$$

All denominators are nonzero. An extension of the output-block function is

$$\widetilde{g}_{v,j}(z_1, \dots, z_{Dt}) = \sum_{\vec{a} \in S^{Dt}} g_{v,j}(\vec{a}) \prod_{h=1}^{Dt} L_{a_h}(z_h). \tag{4.2}$$

It agrees with $g_{v,j}$ on S^{Dt}, and its total degree is at most

$$Dt(2^\ell - 1) \le D(W+1)(2^\ell - 1) \le \frac{2^\ell(2^\ell - 1)}{8} < |K| - 1.$$

This is the degree condition needed in the Cook–Mertz averaging step. Indeed, for any P of total degree less than $|K|-1$,

$$\sum_{\substack{\lambda\in K\\ \lambda\neq 0}} P(\lambda\vec{x}+\vec{y}) = P(\vec{y}).$$

To check the identity, expand in λ. The positive powers have zero sum over $K^{\times}$, and the constant term is repeated an odd number of times in characteristic two. Thus this choice of S and extension supports the same packed evaluation procedure.

Each of the $D+1$ shared registers holds t field elements and occupies $2\ell t = O(W)$ bits. A recursion frame stores a counter over field elements, a child choice, and a constant-size program counter in $O(\log(W+2))$ bits. Field arithmetic can be implemented in $O(\log|K|)$ space [5, Proposition 2]. A polynomial representation and a primitive element can also be found by enumeration within this space. In particular, the interpolation parameters do not require storing a table of field elements.

To compute (4.2), enumerate $\vec{a}$, store one tuple in $Dt\ell = O(W)$ bits, and accumulate its interpolation weight by looping over h and c. Decode the tuple, evaluate the named Boolean gate, and extract its j-th block. Lemma 3.1 bounds this gate computation by $O_G(W+\log(n+2))$ scratch bits even on arbitrary decoded inputs. All remaining arithmetic uses $O(\log(W+2))$ scratch bits. Leaf words obey the same bound by (3.1).

These local queries make no recursive calls to other tree nodes. Their scratch is reused and cleared before each recursive call of the evaluator, while the shared registers and the recursion stack are preserved. Consequently it contributes only one additive $O_G(W+\log(n+2))$ term to (4.1), establishing the lemma. □

Proof of Theorem 1.1. Lemmas 3.2 and 4.1, together with $W = c_G b^2$, give working space

$$O_G\left(b^2 + \left(\frac{n}{b}+1\right)\log(b+2) + \log(n+2)\right). \tag{4.3}$$

Choose

$$b = \left\lceil \left((n+1)\left\lceil \log_2(n+2)\right\rceil\right)^{1/3}\right\rceil.$$

This integer can be computed using logarithmic working space, for example by testing successive integer cubes against the displayed integer target. Substitution into (4.3) gives $O_G(n^{2/3}(\log(n+2))^{2/3})$ for all sufficiently large n. The finitely many remaining lengths and the empty word are handled in finite control. The root bit decides membership. □

Proof of Corollary 1.2. For each fixed $0<\varepsilon<1/3$,

$$(\log(n+2))^{2/3} = O(n^{1/3-\varepsilon}).$$

Apply Theorem 1.1. Taking $\varepsilon = 1/4$ gives the stated common exponent. □

5. Relation to earlier work

The binary normal form and the interval recurrence are established parts of Boolean-grammar theory [1, 3]. The known deterministic linear-space recognition bound is discussed in [2, 4]. The proposed improvement uses a different organization of the recurrence, with local completion of the endpoint blocks and a distance-ordered aggregation of the intermediate blocks.

Grandjean, Grente, and Terrier characterize Boolean languages by SCYK automata [7] and give corresponding characterizations for conjunctive and context-free subclasses. This supplies a parallel view of the substring recurrence. The result here instead organizes that recurrence into bounded endpoint blocks and controls the dependency height of their aggregation in order to obtain the sequential working-space bound of Theorem 1.1.

The evaluator itself is due to Cook and Mertz [5]. The proof uses the implicit-tree implementation explained by Williams [6]. No new tree-evaluation theorem is needed.

References

[1] A. Okhotin, *Boolean grammars*, Information and Computation **194** (2004), no. 1, 19–48. doi:10.1016/j.ic.2004.03.006.

[2] A. Okhotin, *Nine open problems on conjunctive and Boolean grammars*, Bulletin of the EATCS **91** (2007), 96–119. Publisher PDF.

[3] A. Okhotin and C. Reitwießner, *Parsing unary Boolean grammars using online convolution*, Dagstuhl Seminar Proceedings **10501** (2011), 1–11. doi:10.4230/DagSemProc.10501.3.

[4] M. Barash and A. Okhotin, *Linear-space recognition for grammars with contexts*, Theoretical Computer Science **719** (2018), 73–85. doi:10.1016/j.tcs.2017.11.006.

[5] J. Cook and I. Mertz, *Tree evaluation is in space* $O(\log n \cdot \log\log n)$, Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC 2024), 1268–1278. doi:10.1145/3618260.3649664.

[6] R. R. Williams, *Simulating time with square-root space*, Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC 2025), 13–23. doi:10.1145/3717823.3718225. Expanded author version, 25 February 2025, arXiv:2502.17779v1 (including Appendix A).

[7] É. Grandjean, T. Grente, and V. Terrier, *Context-free, conjunctive and Boolean grammars and SCYK automata*, in M.-P. Béal and P. Caron (eds.), Developments in Language Theory (DLT 2026), Lecture Notes in Computer Science **16578**, Springer, 2026, 139–152. doi:10.1007/978-3-032-28404-4_11.

Email address: howew@purdue.edu

Email address: huffma97@purdue.edu

Email address: jenwin@purdue.edu

Email address: kruerl@purdue.edu