

Remembering information and reaching it

Research note extending Kim–Park’s PEG separation

FPRD Lab · Research directed by Joshua Gay

5 September 2026 · Revision 4

The grammar for our non-PEG language

This research note extends Kim–Park’s explicit separation [1]. It follows their lower-bound strategy and adapts their recursive parity rules; novelty of the extensions is not established. The grammar below uses a tree address to select one record. Its start symbol is **A**; **epsilon** denotes the empty string.

```
A  -> 0 A t | 1 A T t | # P1 [  
T  -> T T t | ] B [  
B  -> B 0 | B 1 | 0 | 1  
R  -> R t | R [ | R ] | R 0 | R 1 | epsilon  
  
P0 -> $ R ] | 0 P0 0 | 1 P0 0 | 0 P0 1 | 1 P1 1  
P1 ->          0 P1 0 | 1 P1 0 | 0 P1 1 | 1 P0 1
```

Call its language L . **This grammar is unambiguous, and no ordinary PEG recognizes L** , using Ko’s lower bound and the Loff–Moreira–Reis PEG–scaffolding theorem. The proof is given below in Proposition 1 and Corollary 6.

The proof studies the reversed language $U = L^R$, whose records arrive before its query. The grammar printed here reverses every right-hand side of the grammar for U in Proposition 1; thus $L = U^R$. Reversing derivation trees preserves unambiguity.

A question that arrives too late

Suppose you are given a collection of records. Each record is a row of zeros and ones. You can study the records and organize them however you like. Next, another row of bits arrives, one bit at a time. Finally, someone names a record and asks: **in how many positions do that record and the new row both have a one—is that number odd or even?**

For example, take the record 1011 and the arriving row 1101. They both have a one in the first and fourth positions. The answer is even, or zero. With the record 1001 and arriving row 1000, there is one such position. The answer is odd, or one. This calculation is called an *inner product over the two-element field*: multiply corresponding bits and add modulo two.

If the question names the record in advance, maintain one running parity bit. If it names the record only at the end, that strategy no longer suffices. Storing the arriving row still takes only one write per bit, but answering an arbitrary record’s question may require consulting much of it. Maintaining

every record’s answer during the updates moves the work earlier. Can a more ingenious organization avoid both costs?

This note connects that data-structure question to a question about grammars. Along the way we prove a limitation of one tempting way to classify recognition problems. Two families of instances can have the same number of distinguishable futures at every stage we measure, including after any initial update prefix has been fixed, while their access costs differ. The amount of information distinguished by future answers does not determine the cost of maintaining and consulting that information.

Our construction also gives an unambiguous context-free language outside ordinary parsing expression grammars, using Ko’s lower bound and the PEG-scaffolding theorem described below. The residual-count identities themselves need neither external theorem.

Ford’s question, and why ambiguity does not settle it

A context-free grammar describes strings by recursive construction rules. A familiar example is

$S \rightarrow (S) S \mid \text{epsilon}$

which generates balanced parentheses. A parsing expression grammar, or PEG, instead prescribes recognition operations. Its alternatives are ordered: a successful earlier alternative commits the parser at that choice. This changes what a rule means, even when its written form resembles a context-free rule.

Bryan Ford introduced the PEG framework in his 2004 paper [4]. He explained its practical appeal for describing syntax and established that PEGs can express languages beyond the context-free class. The other direction resisted proof: **is there a context-free language that no PEG recognizes?** Failure of one attempted PEG would not answer this. A language can have many grammars, so the question asks us to exclude every possible PEG for one language.

The distinction is particularly interesting for an *unambiguous* context-free language: one that has a context-free grammar giving every accepted word exactly one parse tree. A unique parse does not automatically provide a PEG. It says how many successful derivations exist, not how a recognizer can find the successful choices as input arrives.

Kim and Park [1, Section 4.3, equation (3)] already give an explicit grammar for a linear context-free language whose reversal is outside PEG. Their construction supplies the separation asked for by Ford. This note follows their lower-bound strategy and adapts their grammar: the eight recursive parity rules for P0 and P1 are the same, up to ordering, while the base case and surrounding record-selection rules change. Here a tree path selects at most one leaf, giving a proof of unambiguity. We also compare families with matching residual counts and different access costs. These are extensions within their approach; their novelty in the literature has not been established. The simulation below is proved directly, but it uses the same Ko lower bound and does not constitute an independent method of resolving Ford’s question.

What Ko contributes

Young Kun Ko studies dynamic data structures: information is prepared, updates arrive, and later queries must be answered [2]. His lower bound applies in the *cell-probe model*. Memory consists of cells holding words of a specified size. Reading or writing a cell costs one probe; computation using the values already read is free. Giving computation away makes an impossibility result in this model especially useful: it rules out clever encodings and arithmetic tricks as well as straightforward implementations.

The consequence we need concerns an arbitrary matrix with n^2 records of n bits each. With words of $\Theta(\log n)$ bits and polynomially many memory cells, an exact deterministic data structure cannot process each arriving coordinate in $O(1)$ probes and answer the final, previously unknown record query in $O(\log n)$ probes. Ko’s Theorem 1.1 and the coordinate-update accounting in his Section 3 supply that consequence. We call it **K** below. Ko’s proof is an external premise; our contribution here includes checking the interface and proving our own reduction to it.

Why should a grammar face a memory-access restriction? Loff, Moreira, and Reis give the bridge [3, Theorem 16]. They introduce a *scaffolding automaton*, which reads one input symbol at a time and appends one node to a permanent graph. The machine can inspect only a fixed-size neighborhood of its latest node. Old information persists, but reaching it takes navigation. Their characterization is

$$L \in \text{PEG} \iff L^R \in \text{SCA}.$$

The superscript R reverses every word in a language. Thus, to exclude a PEG for U^R , we can exclude a scaffold for U . Our direct simulation shows that a scaffold for our record-and-query language would violate K. The reversal is part of the theorem, not a convenient change of convention.

What a residual tells us

After a prefix u has arrived, its *residual* is the set of suffixes that would finish an accepted word:

$$\rho_L(u) = u^{-1}L = \{z : uz \in L\}.$$

Think of it as the complete answer sheet for every possible continuation. If two prefixes have different answer sheets, a deterministic recognizer must distinguish them. Otherwise it would give the same answer to a continuation that should distinguish them.

Here is a small example. Store two bits x_1, x_2 , and allow three final questions: ask for x_1 , for x_2 , or for their parity $x_1 \oplus x_2$. The four possible updates give these answer sheets:

Update	First bit	Second bit	Parity
00	0	0	0
01	0	1	1
10	1	0	1
11	1	1	0

There are four distinct residuals. This is **not the size of one residual**. Each residual consists of the questions answered one; in this example the residual for 00 is empty, while each other residual contains two questions. We count different answer sheets as the update varies.

Four possibilities require two bits to distinguish. But this count does not tell us what memory operations maintain the answer sheet, or how much of it a particular query must consult. Our theorem makes that distinction exact for an unbounded family. It compares canonical instances inside one language, not the global residual growth of two different languages.

A language with a uniquely selected leaf

Let $D = \{\mathbf{t}, [,], 0, 1\}$, with additional delimiters $\$$ and $\#$. A complete binary tree has prefix encoding

$T \rightarrow \mathbf{t} \ T \ T \mid [\ B \]$
 $B \rightarrow 0 \ B \mid 1 \ B \mid 0 \mid 1.$

Thus $\mathbf{t}$ introduces an internal node, and $[s]$ a leaf with a nonempty binary payload. These encodings are self-delimiting. Pending child slots start at one, increase by one on an internal node, and decrease by one after a leaf. The first return to zero uniquely identifies the end of a complete tree.

For an arbitrary data word $a \in D^*$ and path $p \in \{0, 1\}^*$, start at the beginning of a . Each path bit requires a $\mathbf{t}$. A zero descends immediately into the left child; a one skips exactly one complete tree and then descends into the right child. After all path bits, require a leaf $[s]$. If this succeeds, write s_p for its payload and put $p \in P(a)$.

Each path selects at most one leaf. The set $P(a)$ is finite, since each path step consumes a $\mathbf{t}$. The data after the selected leaf is unrestricted. We do not require the whole data word to be a complete tree. For example, $\mathbf{t}[1]\$1\#0$ will be accepted although its unvisited right subtree is missing.

The membership condition is

$$U = \{a\$x\#p^R : p \in P(a), |s_p| = |x|, \langle s_p^R, x \rangle = 1\}.$$

All inner products and ranks in this note are over $\mathbb{F}_2$. The query is a reversed path: the last query bit determines the first tree branch. The payload is reversed in the parity comparison for the same nested-matching reason.

Why every accepted word has one parse

Proposition 1.

The following CFG, with start symbol A , generates exactly U and is unambiguous. Its recursive P0 and P1 rules are adapted from Kim–Park [1, equation (3)]; the tree-addressed selection and parity base case give the language used here.

$A \rightarrow \mathbf{t} \ A \ 0 \mid \mathbf{t} \ T \ A \ 1 \mid [\ P1 \ \#$
 $T \rightarrow \mathbf{t} \ T \ T \mid [\ B \]$
 $B \rightarrow 0 \ B \mid 1 \ B \mid 0 \mid 1$
 $R \rightarrow \mathbf{t} \ R \mid [\ R \mid] \ R \mid 0 \ R \mid 1 \ R \mid \text{epsilon}$

$P0 \rightarrow] \ R \ \$ \mid 0 \ P0 \ 0 \mid 0 \ P0 \ 1 \mid 1 \ P0 \ 0 \mid 1 \ P1 \ 1$
 $P1 \rightarrow \quad \quad 0 \ P1 \ 0 \mid 0 \ P1 \ 1 \mid 1 \ P1 \ 0 \mid 1 \ P0 \ 1$

Proof. For $r \in \{0, 1\}$, induction gives

$$L(P_r) = \{s]z\$x : z \in D^*, |s| = |x|, \langle s^R, x \rangle = r\}.$$

The only terminal base is $P_0 \rightarrow]R\$$. A production $P_r \rightarrow bP_c e$ is present precisely when $r = c \oplus be$. It adds a payload bit on the left and an update bit on the right, changing the parity by be . Since P_1 has no terminal base, an accepting derivation has nonempty payload and update vector.

The two recursive A -productions respectively descend left and skip a complete left subtree before descending right. Appending the path bit on the right produces p^R . The base $A \rightarrow [P_1\#$ enforces parity one at the selected leaf. This proves equality with U .

To prove unambiguity, fix a terminal word. Its delimiters determine the data, update, and query fields. An empty query forces the base A -production; otherwise the last query bit forces the outer recursive production. A skipped tree has a unique boundary and derivation. At the selected leaf, the first closing bracket fixes the payload, while $\$$ fixes the ignored tail. Each P_r -production is determined by the first payload bit and last update bit. Finally R derives each tail uniquely. Induction leaves at most one derivation tree. $\square$

Reversing all production right-hand sides mirrors derivation trees bijectively. In particular, U^R is also an unambiguous CFL. Proposition 8 below proves that neither language is linear context-free.

Counting all possible futures

The formulas in this section concern full residuals in U , not a promised subset of continuations.

Proposition 2. Residual rank at an arbitrary update cut

Fix any $a \in D^*$ and update length $j \geq 0$. For each accessible leaf $p \in P(a)$, let $b_p = s_p^R$ and $\ell_p = |b_p|$. Let $M_j(a)$ contain the first j coordinates of every row b_p with $\ell_p \geq j$. Then

$$\rho_U(a\$v) = \left\{ y\#p^R : \begin{array}{l} p \in P(a), \quad \ell_p \geq j, \quad |y| = \ell_p - j, \\ \langle b_p[1:j], v \rangle + \langle b_p[j+1:\ell_p], y \rangle = 1 \end{array} \right\}.$$

Consequently, across all $v \in \mathbb{F}_2^j$, the number of residuals is $2^{\text{rank } M_j(a)}$.

Proof. Every accepting continuation must finish the binary update field, then supply $\#$ and a successful reversed path. Length equality and a split inner product give the formula. Two update prefixes v, v' have the same residual exactly when $M_j(a)(v + v') = 0$. If a row distinguishes them, choose its path and an all-zero remaining update: the acceptance values differ. Rank-nullity gives the count. A matrix without rows has rank zero and gives the single empty residual. $\square$

Now fix a binary matrix M of width $d \geq 1$ and $m = 2^h$ rows, numbered from zero. Let $\tau(M)$ be the complete balanced prefix tree with leaf i equal to $[M_i^R]$. Write $\text{code}_h(i)$ for the h -bit root-to-leaf address. Then

$$\tau(M)\$y\#\text{code}_h(i)^R \in U \iff \langle M_i, y \rangle = 1.$$

For a query prefix r , $|r| \leq h$, define its cylinder

$$I(r) = \{i : \text{code}_h(i)^R \text{ begins with } r\}.$$

This is one residue class modulo $2^{|r|}$: the query reveals low-order index bits first. If $\text{code}_h(i)^R = rz_i$, the full residual after $\tau(M)\$y\#r$ is exactly

$$\{z_i : i \in I(r), (My)_i = 1\}.$$

The remaining suffixes are distinct, so this residual identifies the answer vector on $I(r)$. No other continuations are possible: all leaf lengths equal d , and all successful addresses have length h .

Proposition 3. Conditional residual counts and probabilities

Define the update counts

$$u_M(j) = |\{\rho_U(\tau(M)\$v) : v \in \mathbb{F}_2^j\}|, \quad 0 \leq j \leq d.$$

For any fixed update prefix $v \in \mathbb{F}_2^j$ and query prefix r , define

$$Q_M(v, r) = |\{\rho_U(\tau(M)\$vz\#r) : z \in \mathbb{F}_2^{d-j}\}|.$$

Then

$$u_M(j) = 2^{\text{rank } M[:, 1:j]}, \quad Q_M(v, r) = 2^{\text{rank } M[I(r), j+1:d]}.$$

Under a uniform choice of z , these latter residuals are equiprobable. If the indicated rank is s , each occurs with probability 2^{-s} and has 2^{d-j-s} completions.

Proof. The first formula is Proposition 2. For the second, the possible answer vectors on $I(r)$ form the affine image

$$M[I(r), 1:j]v + M[I(r), j+1:d]z.$$

Its size is 2^s . Every fibre is a translate of the kernel and has size 2^{d-j-s} . The distinct suffixes identify the answer vector with the full residual. $\square$

Taking $j = 0$ gives the residual count across all update vectors at every query prefix. Taking $j = d$ gives count one: the update word is then completely fixed. The sole residual need not be the same in two compared instances.

How to hide a hard query inside matching counts

The next construction adds *guard records*: questions that expose individual update bits. Their job is to make the number of distinguishable answer sheets the same in both families. A final extra update bit, the *marker*, keeps that equality true even after earlier update bits have been fixed.

Consider just one cluster with two data bits x_1, x_2 and final marker b . In the hard cluster, the distinguished question asks for $x_1 \oplus x_2 \oplus b$. In the easy cluster, it asks for b . Both also contain guard questions for x_1 and x_2 , plus a question whose answer is always zero.

Update x_1x_2b	Hard answers: distinguished, two guards, zero	Easy answers: distinguished, two guards, zero
000	0000	0000
001	1000	1000
010	1010	0010
011	0010	1010
100	1100	0100

Update x_1x_2b	Hard answers: distinguished, two guards, zero	Easy answers: distinguished, two guards, zero
101	0100	1100
110	0110	0110
111	1110	1110

Both columns contain eight different answer sheets, each appearing once. The guards reveal the data bits; after those are known, the distinguished answer reveals the marker. The hard question has changed which update produces an answer sheet without changing the number of sheets.

This example alone is not a lower bound: two bits are easy to handle. For arbitrary record length, however, the distinguished questions contain an arbitrary matrix of inner products. We must also arrange the record addresses carefully. Early query bits select an original record while leaving its entire cluster available; later bits select a position within that cluster. The formal row indexing below achieves precisely this.

Putting the marker last matters. When we fix an initial part of the update, the marker remains free until all the data bits have arrived. It therefore contributes an independent dimension whenever the distinguished record is still among the possible questions. If we instead fixed the marker first, the distinguished question could collapse or depend on the data matrix in a way the easy question does not.

The exact matching statement

Theorem 4.

For every $n \geq 1$ and $A \in \mathbb{F}_2^{n^2 \times n}$, there are matrices $H(A)$ and E_n , both of width $d = n + 1$ and with the same $m = \Theta(n^3)$ rows, such that:

1. $u_{H(A)}(j) = u_{E_n}(j) = 2^j$ for every $0 \leq j \leq d$.
2. $Q_{H(A)}(v, r) = Q_{E_n}(v, r)$ for every update prefix v and every query prefix r .
3. Under uniform completions of v , the two distributions on residuals have the same multiset of probabilities.
4. The continuation problem for E_n has constant update and query probe cost. The family $H(A)$ contains arbitrary Multiphase Inner Product with constant overhead.

Consequently, under K, these matching profiles do not determine whether constant updates and logarithmic queries are attainable.

Construction. Put

$$N = 2^{\lceil \log_2(n^2) \rceil}, \quad G = 2^{\lceil \log_2(n+1) \rceil}, \quad m = NG.$$

Pad A with zero rows to N rows. Index the new rows as $i + Nc$, where $0 \leq i < N$ and $0 \leq c < G$. Define

$$H(A)_{i+Nc} = \begin{cases} (A_i, 1), & c = 0, \\ (e_c, 0), & 1 \leq c \leq n, \\ 0, & n < c < G, \end{cases}$$

where e_c is the c th basis vector of $\mathbb{F}_2^n$. Replace every distinguished row $(A_i, 1)$ by $(0^n, 1)$ to obtain E_n . The marker is the **last** update coordinate. This placement is what gives the conditional strengthening.

Update counts. The guard rows contain a basis for the first n columns. Adding the marker column raises the rank once more, since a distinguished row has marker one and guards have marker zero. Every initial j columns have rank j , in both matrices. Proposition 3 gives assertion 1.

Query counts after any fixed update prefix. Write $a = \log_2 N$ and fix r .

If $|r| \leq a$, its cylinder leaves the cluster coordinate c completely unrestricted. It contains an entire cluster and therefore all guard rows and a distinguished row. For every j , the selected rows restricted to columns $j + 1, \dots, d$ have full rank $d - j$ in both matrices.

If $|r| = a + t$, the query fixes one i and a residue class C_r of cluster positions modulo 2^t . For $j \leq n$, the surviving guards with $j < c \leq n$ remain distinct basis rows; guards with $c \leq j$ restrict to zero. If $0 \in C_r$, the distinguished row adds precisely one dimension because its final marker is still one. If $0 \notin C_r$, the selected rows are identical. Thus the rank in either matrix is

$$|\{c \in C_r : j < c \leq n\}| + \mathbf{1}_{\{0 \in C_r\}}, \quad j \leq n.$$

For $j = d$, no columns remain and both ranks are zero. Proposition 3 proves assertions 2 and 3, including the endpoints and every query branch.

Easy access. Write the update as $y = (x, b)$. Store each arriving coordinate. Given index $i + Nc$, return b if $c = 0$, return x_c if $1 \leq c \leq n$, and otherwise return zero. This costs one write per update and at most one read per query. Arithmetic on the given index is free in the cell-probe model.

Embedded hard problem. Given A , preprocess $H(A)$. Supply $y = (x, 0)$ and query row i with cluster position zero. The answer is exactly $\langle A_i, x \rangle$. The fixed final zero can be processed with the last coordinate update, at constant overhead. The transformation uses polynomial space, $d = n + 1$ update symbols, and $\log_2 m + 1 = O(\log n)$ query symbols. Hence excluded costs for Multiphase Inner Product would follow from the same costs for the whole hard family. $\square$

The lower bound is worst-case over the family: particular matrices such as $A = 0$ are easy. A single algorithm must handle arbitrary A , with the preprocessing information stored in its charged memory. A different uncharged program hardwired for each matrix is not the model under comparison.

What the theorem forgets, and what it preserves

The profiles preserve the number of distinct full residuals, including after any initial update prefix has been fixed. They also preserve the entropy and the multiset of residual probabilities under uniform completions. They do not preserve the residual sets themselves, their labels, or their transition maps.

For example, let $n = 1$ and $A = [1]$. On update $y = 10$, the distinguished hard row answers one and the easy row answers zero. After its complete query, the residuals are respectively $\{\varepsilon\}$ and $\emptyset$, although both conditional counts are one. Conditioning instead on a fixed **final** marker value while varying the preceding data is not covered by the prefix-conditioning theorem.

No claim is made that the easy canonical family itself forms a CFL, or that its cell-probe algorithm has a fixed-locality scaffold implementation. The theorem compares the precisely defined continuation problems embedded in U . It does not rule out classification using richer residual structure.

From a scaffold to a forbidden data structure

Proposition 5. From a fixed scaffold to cheap probes

If a fixed scaffolding automaton recognizes U , then Multiphase Inner Product admits the costs excluded by K.

Proof. Let the automaton have outdegree b , inspection radius k , a finite node-label alphabet, and finite control. Index its nodes by creation time. A node is a constant number of words containing its label and outgoing pointers. A constant-size memory header stores the control state, current top, and next free node index. All state surviving between operations is in charged memory.

One input symbol reads that header and follows every relevant path of length at most k from the top. Even if visits are repeated, this accesses at most a constant multiple of

$$1 + b + b^2 + \dots + b^k$$

node fields. The fixed transition table is evaluated for free. Its new node refers only to permitted inspected endpoints, itself, or absent pointers. Writing its fields and the header uses a constant number of further probes. Self-pointers and repeated endpoints cause no difficulty because the inspection depth is fixed. The simulation supplies exactly the prescribed labelled neighbourhood to the transition table.

During preprocessing, simulate the automaton on $\tau(H(A))\$$. This depends only on A and n . Process the bits of x one at a time; with the last bit also process the fixed marker zero. Each update uses constantly many probes. After index i is revealed, simulate $\#$ followed by its reversed h -bit address, and return the automaton's accepting-state bit. There are $h + 1 = O(\log n)$ query symbols.

The balanced tree has $m - 1$ internal nodes and m leaves of length $d + 2$, hence

$$|\tau(H(A))| = m(d + 3) - 1 = O(n^4).$$

All node addresses therefore fit in logarithmic words and the memory is polynomial. The update algorithms receive no queried row for free. The query arrives after all data updates. If a read-only final query is required, simulated writes can be kept in an operation-local overlay, since no later operation needs them. Correctness follows from the encoding equivalence and Theorem 4. $\square$

This simulation excludes arbitrary fixed scaffolding automata, regardless of their internal summaries, sharing, or preprocessing strategy. It is not a lower bound against one proposed data representation. The proof is included here in full, while its methodological source remains [1].

Corollary 6. An unambiguous CFL outside PEG

Under K, $U \notin \text{SCA}$ and

$$U^R \in \text{UCFL} \setminus \text{PEG}.$$

Proof. Proposition 5 contradicts K if U has an SCA. Proposition 1 gives the unambiguous grammar for U^R , and [3, Theorem 16] converts a hypothetical PEG for U^R into an SCA for U . $\square$

We have not proved whether U itself has a PEG. In particular, this paper does not fill both asymmetric PEG/reversal categories within UCFL.

Corollary 7. Failure in both orientations

Under K, $V = U \cup U^R$ is a self-reversing unambiguous CFL in neither PEG nor SCA.

Proof. Words of U start with $\mathfrak{t}$ or $[$. Words of U^R start with a bit, or with $\#$ when the path is empty. The languages are disjoint, so a new start symbol joining their unambiguous grammars remains unambiguous. Also $V^R = V$.

The regular filter selecting first symbols $0, 1, \#$ extracts exactly U^R . If V had a PEG, intersection with that regular language would give a PEG for U^R , contrary to Corollary 6. This regular-intersection closure follows directly by an anchored PEG test for the filter followed by the original recognizer [4]. Self-reversal and [3] also exclude SCA membership. $\square$

Failed conversions and finite evidence

Unambiguity of a CFG does not justify treating its alternatives as PEG ordered choices. For the grammar in Proposition 1, prioritizing $\mathfrak{t} \text{ A } 0$ before $\mathfrak{t} \text{ T A } 1$ rejects the valid word

$\mathfrak{t}[1]\mathfrak{t}[1][1]\$1\#01$

Prioritizing the right alternative rejects

$\mathfrak{t}\mathfrak{t}[1][1][1]\$1\#10$

An end-of-input guard inside each outer alternative still fails on nested instances: respectively $\mathfrak{t}\mathfrak{t}[1]\mathfrak{t}[1][1]\$1\#010$ and $\mathfrak{t}[0]\mathfrak{t}\mathfrak{t}[1][1][1]\$1\#101$. A nested call can commit before its caller checks the caller's trailing path bit. These counterexamples reject particular conversions; they are not proofs that $U \notin \text{PEG}$.

The accompanying original checker implements a CFG derivation counter and a separate semantic traversal. Its earlier run covered malformed data, ignored tails, varied tree depths, canonical encodings, and mutations. The new checker focuses on Theorem 4: it checks all binary 4×2 input matrices, both 1×1 matrices, and larger deterministic examples. It also forms actual residual answer signatures through the semantic recognizer and checks the conditional fibre multiplicities. All checks passed.

The tests corroborate index reversal, marker placement, rank formulas, and the explicit failures. Universal unambiguity, nonlinearity, and asymptotic lower bounds are supported by proofs, not by enumerating finite cases. The companion audit records the precise scope and reproduction commands.

The witness is not linear context-free

Proposition 8. Nonlinearity of the language

Neither U nor U^R is linear context-free.

Proof. Intersect U with the regular language described by

$\mathfrak{t} (\mathfrak{t} \mid [0])^* [1] \$ 1 \# 1.$

The final query takes one right turn. A word in the intersection is therefore exactly $te[1]\$1\#1$, where e is a complete prefix tree whose leaves are all $[0]$. The selected $[1]$ is the unique such leaf and the regular restriction allows no tail after it.

Apply the homomorphism sending $\mathfrak{t}$ to a , 0 to b , and all other symbols to ε . Its image is aE , where $E \rightarrow aEE \mid b$. These are precisely the nonempty words with total a -minus- b balance zero and strictly positive balance at every nonempty proper prefix.

The linear-language pumping lemma [5, Lemma 6] gives a constant P and, for each sufficiently long word, a factorization $uvwxy$ with $|uvxy| \leq P$, $|vx| > 0$, and uv^iwx^iy in the language for every $i \geq 0$.

Choose $N > P$ and

$$z = a^{N+1}b^Na^Nb^{N+1} \in aE.$$

The bounded end factors force v to contain only initial a 's and x only final b 's. Put $r = |v|$, $s = |x|$. Pumping down with $r \neq s$ destroys total balance. If $r = s$, both are positive, and the prefix through the first b -block has balance $1 - r \leq 0$, with a nonempty second a -block still following. This also leaves aE . Thus aE is not linear.

Regular intersection and homomorphism preserve linearity: one can annotate the single nonterminal by DFA entry/exit states, and replace its terminal contexts by their homomorphic images. Hence linearity of U would contradict that of its extracted image. Reversal also preserves the one-nonterminal production property, so U^R is not linear either. $\square$

What remains to classify

The theorem isolates a loss of information in the proposed classifier. Counts retain quotient sizes while discarding how individual update coordinates and query branches act on those quotients. Equal size, equal conditional size, and equal residual-probability multisets leave room for different access costs.

A stronger classification must retain enough structure to ask whether one representation supports all required updates and late queries within the available local work. Passing the present obstruction does not imply SCA or PEG membership. An intrinsic scaffold obstruction independent of cell-probe lower bounds remains open. So do a PEG construction for U and a witness retaining both linearity and unambiguity.

The reusable contribution is the terminal-marker construction and its exact comparison inside an explicit UCFL. Rank-nullity, pumping, and the external lower bound retain their established provenance. The resulting separation is a focused obstruction to cardinality-based classification, not a classification of all CFLs.

References

1. Jungyeom Kim and Jihyeok Park. *Separating Parsing Expression Grammars using Cell-Probe Lower Bounds*. arXiv:2608.29592v1, 2026. [Article](#). Methodological source for the late-query strategy.
2. Young Kun Ko. *An $\Omega((\log n / \log \log n)^2)$ Cell-Probe Lower Bound for Dynamic Boolean Data Structures*. arXiv:2603.25914v1, 2026, Theorem 1.1 and Section 3. [Article](#). External lower-bound premise; source inspected 5 September 2026.
3. Bruno Loff, Nelma Moreira, and Rogério Reis. *The computational power of parsing expression grammars*. Journal of Computer and System Sciences **111** (2020), 1–21. [DOI](#); [author version](#), [Theorem 16](#).
4. Bryan Ford. *Parsing Expression Grammars: A Recognition-Based Syntactic Foundation*. POPL 2004, 111–122. [Paper](#); [DOI](#).
5. Géza Horváth and Benedek Nagy. *Pumping lemmas for linear and nonlinear context-free languages*. Acta Universitatis Sapientiae, Informatica **2**(2) (2010), 194–209. [Paper](#), Lemma 6.