

Real-Time Multitape Languages Transfer to Parsing Expression Grammars

via Persistent Stacks in Scaffolding Automata

Joshua Gay*

5 August 2026

Abstract

We prove that the reversal of every language recognized by a finite deterministic strict real-time multitape Turing machine has a parsing expression grammar. The construction represents each work tape by two persistent stacks. One newly appended scaffold node stores the simultaneous update of every stack root, using one named backward port per stack and distance two. This gives the containment

$$\text{rev}(\text{RT-MTTM}) \subsetneq \text{PEG}.$$

Properness follows from the online lower bound proved by Loff–Moreira–Reis. Combining the transfer with Slisenko’s real-time palindrome recognizer and the single-head simulation of Leong and Seiferas yields

$$\{ww^R : w \in \{0,1\}^*\} \in \text{PEG},$$

as a first major corollary, disproving Conjecture 7 of Loff–Moreira–Reis. The machine-to-scaffold simulation, the sufficient scaffold-to-PEG direction, and their strict-real-time containment composition have been formalized in Lean 4. In particular, Lean verifies the implication from an exact finite strict real-time multitape EPAL recognizer to $\text{EPAL}_2 \in \text{PEG}$; the source-verified classical results establish that recognizer premise.

1 Introduction

Parsing expression grammars (PEGs), introduced by Ford [1], are recognition-based grammars whose ordered choice operator gives them behavior quite different from context-free grammars. Loff, Moreira, and Reis introduced *scaffolding automata* and proved that they characterize PEG languages up to reversal [4, Theorem 16]. They also proposed the following test of the formalism’s power:

$$\text{EPAL}_2 := \{ww^R : w \in \{0,1\}^*\},$$

the language of even-length binary palindromes. Their Conjecture 7 states that $\text{EPAL}_2 \notin \text{PEG}$.

The key observation of this paper is that one scaffold node can act as a persistent cons cell for several logical stacks at once. Its finite label is a tuple of stack-top symbols, and its named backward ports point independently to the persistent predecessor of each stack. Thus one physical append operation can perform a fixed tuple of logical memory updates.

This yields the main transfer theorem.

*ChatGPT, Claude, and Codex were used for conjecture exploration, source checking, Lean formalization, computational auditing, and editorial assistance. The author is responsible for the mathematical claims and the final manuscript.

Theorem 1.1 (Real-time transfer). *Let L be recognized by a finite deterministic strict real-time multitape Turing machine with a fixed positive number of ordinary single-head work tapes. The machine receives one input symbol and makes one global transition per step, has no input-length advice, and performs no post-input computation. Then*

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

Together with the online lower bound already proved by Loff–Moreira–Reis, the transfer gives a proper class containment:

$$\text{rev}(\text{RT-MTTM}) \subsetneq \text{PEG}.$$

The proof has two parts. First, a fixed collection of stacks, each performing one push, keep, or pop operation per input symbol, is simulated letter-for-letter by a distance-two scaffolding automaton. Second, the usual two-stack zipper represents each Turing tape. The resulting scaffold has degree $2m$ for an m -tape machine.

Slisenko constructs a real-time multitape palindrome recognizer [6]. Leong and Seiferas give an ordinary single-head multitape implementation, using twenty tapes [3]. A parity bit restricts the recognizer to even lengths. Since $\text{EPAL}_2^R = \text{EPAL}_2$, theorem 1.1 gives the announced consequence.

Corollary 1.2. $\text{EPAL}_2 \in \text{PEG}$. *Hence Conjecture 7 of Loff–Moreira–Reis is false.*

The historical direction is worth noting. Loff, Moreira, and Reis explicitly investigated whether reversals of PEG languages admit efficient online Turing-machine recognition and refuted that proposed containment [4, Section 5.3]. The construction here travels from strict real-time machines to scaffolding automata and then applies their characterization.

2 Models and conventions

For a word $x = x_1 \cdots x_n$, write $x^R = x_n \cdots x_1$, and for a language L , write $L^R = \{x^R : x \in L\}$. All alphabets and state sets below are finite. Throughout, PEG is Loff–Moreira–Reis’s class of ordinary total PEGs: the start expression must succeed while consuming the complete input. This is complete-match PEG semantics, not a prefix-acceptance variant [4, Definitions 1–4].

2.1 Scaffolding automata

We use the model of Loff, Moreira, and Reis [4, Definitions 10–15]. A scaffold of degree d consists of a sequence of labelled nodes. Each node has d named ports, and every port is either missing or points to a node no later than its source. The scaffold begins with an unlabelled base node.

On each input symbol, the automaton appends exactly one labelled node. Its transition sees the state and the depth- k unfolding of the old top: a partial, labelled, d -ary tree that records labels along named port paths but not node identity. Each port of the new node may be left missing, may target the new node itself, or may target the endpoint of a port word of length at most k starting at the old top. The fixed constants d and k are the *degree* and *distance*.

Acceptance depends on the finite control state immediately after the last input symbol. The characterization we use is:

Theorem 2.1 (Loff–Moreira–Reis). *For every language K ,*

$$K \in \text{PEG} \iff K^R \text{ is decided by a scaffolding automaton.}$$

Lemma 2.2 (Fresh initial state). *Every finite scaffolding automaton A has an equivalent finite scaffolding automaton A^* whose initial state occurs only before the first input symbol. The construction preserves acceptance of the empty word and every nonempty word without inserting an input step.*

Proof. Let q_0 be the old initial state, introduce one fresh state q_* , and retain an injected copy $\iota(q)$ of every old state, including $\iota(q_0)$. The new initial state is q_* . On its first input symbol and observed neighborhood, q_* performs exactly the old q_0 -transition, with the returned state injected by ι . From every $\iota(q)$, perform the old q -transition and inject its returned state. Thus no transition returns q_* , while a transition that formerly returned q_0 now returns the ordinary recurrent copy $\iota(q_0)$.

Declare q_* accepting exactly when q_0 was accepting, and declare $\iota(q)$ accepting exactly when q was. The empty computation therefore has the same acceptance value. After the first symbol, the old and new computations have the same scaffold and corresponding control states; an induction over the remaining symbols preserves both facts. Hence all nonempty decisions also agree. Adding one state preserves finiteness and changes neither the scaffold alphabet nor its degree or distance. $\square$

This is the state duplication used without loss of generality in the sufficient direction of Loff–Moreira–Reis Theorem 16 [4, p. 23].

2.2 Strict real-time multitape machines

We use the letter-synchronous real-time convention of Rosenberg [5]: after each arriving input symbol, the machine performs exactly one transition and the resulting state reports the answer for the prefix then seen.

Definition 2.3. A *strict real-time m -tape machine* M , where $m \geq 1$, has a finite input alphabet Σ , finite tape alphabet T with a blank symbol, finite control Q , and m ordinary two-way work tapes with one head per tape. Its transition function has the form

$$\delta : Q \times \Sigma \times T^m \longrightarrow Q \times (T \times \{-1, 0, +1\})^m.$$

At each step exactly one new input symbol is supplied, one global transition is made, and every work-tape head writes and moves by at most one cell. Acceptance of a prefix is determined by the resulting control state.

The input is an arriving symbol, not a preloaded input tape. In particular, the transition receives neither the total input length nor an end-of-input work interval.

3 Parallel persistent stacks

We first isolate the mechanism responsible for the transfer. Consider a deterministic finite-control machine with $s \geq 1$ stacks over a finite alphabet A . On every input step it reads the s top symbols and performs one of

$$\text{push}(a), \quad \text{keep}, \quad \text{pop}$$

on each stack. Popping an empty stack leaves it empty.

We construct a scaffolding automaton of degree s and distance two. Reserve port j for stack j , and take the scaffold alphabet to be

$$\Gamma = (A \cup \{\perp\})^s.$$

For a scaffold node v , let $\lambda_j(v)$ denote component j of its label and let $e_j(v)$ be its port- j target. An unlabelled or missing node has component $\perp$.

The stack rooted at v is decoded recursively by

$$D_j(v) = \begin{cases} \varepsilon, & \lambda_j(v) = \perp, \\ a D_j(e_j(v)), & \lambda_j(v) = a \in A, \end{cases} \quad (1)$$

with $D_j(\emptyset) = \varepsilon$. The recursion terminates because the construction uses only strictly backward targets.

3.1 The update table

Let v be the old scaffold top and v^+ the new one. Component j of v^+ and port j are selected as follows:

operation	new component $\lambda_j(v^+)$	new target $e_j(v^+)$
push(a)	a	v , named by ε
keep	$\lambda_j(v)$	$e_j(v)$, named by j
pop	$\lambda_j(e_j(v))$	$e_j(e_j(v))$, named by jj

Every label read and every target named by this table occurs in the depth-two unfolding of v . The same newly appended node carries all s component and port choices simultaneously.

The empty-stack case requires one invariant. Extend $\lambda_j(\emptyset) = \perp$, and define

$$\text{CleanEmpty}(S) \iff \forall v, j, \lambda_j(v) = \perp \implies \lambda_j(e_j(v)) = \perp. \quad (2)$$

Thus a node that terminates stack j points, if anywhere, to another node that also terminates stack j . This prevents an empty pop from exposing a stale symbol through a historical port.

Lemma 3.1 (Persistent-stack update). *Suppose the old scaffold is strictly backward and satisfies CleanEmpty. After appending the node specified by the table, the decoded stack at v^+ is exactly the result of the selected stack operation. The new scaffold is strictly backward and again satisfies CleanEmpty.*

Proof. Fix a stack j .

For a push, (1) gives

$$D_j(v^+) = a D_j(v).$$

For a keep with a nonempty stack, the new component copies $\lambda_j(v)$ and the new port skips directly to $e_j(v)$, so the same decoded list is obtained. If the stack is empty, the copied component is $\perp$, and the result is empty.

For a pop of a nonempty stack, write

$$D_j(v) = a D_j(e_j(v)).$$

The new component is the first component of this tail and the new target is its predecessor, hence $D_j(v^+) = D_j(e_j(v))$. If the old stack is empty, **CleanEmpty** implies $\lambda_j(e_j(v)) = \perp$, so the new decoded stack remains empty.

All three targets are named by port words and therefore lie at or below the old top; none names the new node. Strict backwardness follows. It remains to check **CleanEmpty** at the new node. The push case is vacuous. In the keep case it is inherited from v . In the pop case it is inherited one node deeper, from $e_j(v)$. Old nodes are unchanged. $\square$

Theorem 3.2 (Parallel persistent stacks). *Every deterministic letter-synchronous finite-control machine with a fixed positive number s of stacks, performing at most one push, keep, or pop per stack and per input symbol, is simulated exactly by a degree- s , distance-two scaffolding automaton.*

Proof. Interpret the unlabelled base as the empty root for every stack. It is strictly backward and satisfies **CleanEmpty**. The scaffold state is the stack machine's finite control. On each input symbol, the transition reads the old top components, computes the stack machine's next control and operations, and appends the tuple-valued node specified above.

Induction on the input length, using lemma 3.1, shows that the controls agree and that every decoded scaffold stack equals its source stack after every prefix. Therefore the accepting states agree. $\square$

4 From work tapes to persistent stacks

Represent work tape i by a zipper

$$(L_i, c_i, R_i),$$

where c_i is the scanned symbol, L_i lists the cells strictly to the left of the head in nearest-first order, and R_i lists the cells strictly to the right in nearest-first order. The scanned symbols $(c_1, \dots, c_m)$ are stored in finite control. Blank cells beyond the finite lists are supplied by the fixed blank convention.

Suppose the machine writes b_i and then moves its i -th head. The zipper updates are:

$$\begin{aligned} \text{right: } (L_i, c_i, R_i) &\mapsto (b_i L_i, \text{head}_{\sqcup}(R_i), \text{tail}(R_i)), \\ \text{left: } (L_i, c_i, R_i) &\mapsto (\text{tail}(L_i), \text{head}_{\sqcup}(L_i), b_i R_i), \\ \text{stay: } (L_i, c_i, R_i) &\mapsto (L_i, b_i, R_i). \end{aligned} \tag{3}$$

Consequently each ordinary tape transition performs at most one operation on each of the two stacks L_i, R_i . All m tapes update in parallel.

Lemma 4.1 (Tape zipper). *Every strict real-time m -tape machine is simulated letter-for-letter by a finite-control machine with $2m$ stacks, performing at most one push, keep, or pop on each stack per input symbol.*

Proof. Use the representation (3). The source transition depends only on the finite control, the arriving input symbol, and the m scanned symbols. The new scanned symbols are determined by the written symbols, move directions, and the old top symbols of the $2m$ stacks. Hence they can be placed in the next finite-control state. Equation (3) assigns one stack operation to each zipper half. A direct induction shows equality of the source and decoded tape contents after every prefix. $\square$

Theorem 4.2 (Strict machine-to-scaffold simulation). *Let M be a strict real-time m -tape machine, $m \geq 1$. There is a scaffolding automaton A_M of degree $2m$ and distance two such that*

$$L(A_M) = L(M).$$

After every input prefix, the two persistent stacks associated with each tape, together with its scanned symbol in finite control, reconstruct the source tape exactly. In particular, the reconstructed and source tapes agree at every integer offset from the head, including their implicit blank tails, and the source control states agree.

Proof. Compose lemma 4.1 with theorem 3.2. If the source control and tape alphabet are Q and T , the augmented control is $Q \times T^m$, and the tuple label alphabet is $(T \cup \{\perp\})^{2m}$. Both are finite because Q , T , and m are finite. Since $m \geq 1$, the scaffold degree $2m$ is positive; its distance is the fixed constant two. Exactly one scaffold node is appended for each input symbol. $\square$

5 The real-time transfer

Theorem 5.1. *If L is recognized by a strict real-time multitape machine, then*

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

Equivalently,

$$\text{rev}(\text{RT-MTTM}) \subseteq \text{PEG}.$$

Proof. By theorem 4.2, a scaffolding automaton A decides L . If its initial state can recur, apply lemma 2.2; this preserves the language. Put $K = L^R$. Then $K^R = L$ is decided by a scaffolding automaton, so theorem 2.1 gives $K \in \text{PEG}$. $\square$

Lemma 5.2 (Strict real time is online constant time). *Every language recognized by a strict real-time multitape machine belongs to $\text{Online}(O(1))$ in the sense of Loff–Moreira–Reis Definition 22.*

Proof. Their lower-bound model is explicitly a multi-tape online Turing machine [4, p. 5]; the configuration count in their Theorem 25 proof records the plural tape-head positions and the bounded neighborhood around every head, with the fixed machine-dependent tape count absorbed by the O -constant [4, p. 33]. Their m there is the suffix length in the equivalence relation; the tape count is a separate constant factor.

Fix a strict machine M . Give the online machine the same finite control and fixed finite family of work tapes. Between successive advances of its one-way input head, it reads the current symbol and performs M 's one simultaneous work-tape transition. Initial alignment and the empty-input answer are finite-control cases. The auxiliary tape containing the exact input length in binary is left unread; declining to use an available tape does not place the simulator outside Definition 22. In fact, the authors state that their lower bound survives even arbitrary nonuniform contents on that auxiliary tape [4, footnote 10].

An induction on the consumed prefix shows equality of the control state, every work-tape content, and every work-tape head position. The fixed transition-and-advance schedule takes at most a machine-dependent constant c_M steps per symbol. The simulator therefore decides the same language and belongs to $\text{Online}(O(1))$. $\square$

Corollary 5.3 (Proper class containment).

$$\text{rev}(\text{RT-MTTM}) \subsetneq \text{PEG}.$$

Proof. Inclusion is theorem 5.1. Loff, Moreira, and Reis prove that there is a language $L \in \text{PEG}$ such that neither L nor L^R belongs to $\text{Online}(t(n))$ for any $t(n) = o(n/(\log n)^2)$ [4, Theorem 23]. Write W for this final tagged witness, to distinguish it from the proof’s intermediate scaffold language H . By lemma 5.2 and $O(1) = o(n/(\log n)^2)$, neither lower-bound orientation can contain the relevant strict machine language.

If $W \in \text{rev}(\text{RT-MTTM})$, then $W = X^R$ for some $X \in \text{RT-MTTM}$, so $X = W^R \in \text{Online}(O(1))$, contradicting Theorem 23. Hence $W \notin \text{rev}(\text{RT-MTTM})$, which makes the inclusion proper. $\square$

Remark 5.4 (Construction size). The Loff–Moreira–Reis proof is constructive, so theorem 5.1 effectively yields a finite PEG. Direct expansion is impractical. For the twenty-tape implementation used below, the scaffold degree is 40, and a depth-two neighborhood has $1 + 40 + 40^2 = 1641$ positions. Even with a two-symbol tape alphabet, the tuple label set has at least 3^{40} elements; the ambient family of fully labelled depth-two neighborhoods alone has more than 10^{31000} members. This size is a property of the direct generic compiler, not of the language itself.

6 Even-length palindromes

Let

$$\text{PAL}_2 = \{x \in \{0, 1\}^* : x = x^R\}.$$

The external machine result used in this section is deliberately stated at the interface at which it enters the proof.

Proposition 6.1 (Classical streaming reporter). *There is a fixed deterministic finite multihead tape-unit machine which starts from blank work storage, receives one binary input symbol per round, performs at most a fixed c ordinary controller/unit operations per round, continues for arbitrarily many rounds, and at the end of every nonempty round reports whether the prefix received through that round belongs to PAL_2 .*

Proof. Slisenko’s simplified paper concludes that a Turing machine recognizes palindromes in real time [6, pp. 123,139]. The paper’s operative convention immediately before Proposition 4.6 says that the next input letter arrives at every machine tick [6, p. 137]. No endmarker, preloaded input, total-length advice, or finishing interval occurs in that one-letter-per-tick convention. The theorem’s word “recognizes” is a two-sided language decision, not merely enumeration of positive instances.

For same-prefix timing, fix a prefix u and regard u itself as a legitimate finite input. The theorem decides u at tick $|u|$. Since the machine is deterministic and receives no endmarker or length advice, its configuration after $|u|$ ticks depends only on those $|u|$ letters and is identical whether the input observed is u or any extension uv . Thus at every tick the control state is the two-sided verdict for the prefix just read. The machine is a fixed deterministic Turing-machine program with fixed finite alphabets and a fixed finite family of heads and work units. One tick of that fixed program has a uniform finite controller/unit-operation bound c . Specializing its finite input alphabet to $\{0, 1\}$ gives the proposition.

Leong and Seiferas independently describe the required storage-model boundary: their deterministic command transducer returns the symbols under all logical heads without delay, and Corollary 3.6 replaces every fixed one-dimensional multihead unit by finitely many ordinary single-head tapes in real time [3, pp. 167,174]. They explicitly identify a twenty-tape ordinary implementation of Slisenko’s recognizer [3, p. 166]. Their role here is storage realization, not

palindrome correctness. Galil's later shorter construction corroborates the real-time result but reuses Slisenko's chain notion and four string lemmas, so it is not an independent premise [2]. $\square$

Lemma 6.2 (Fixed-round strictification). *Let C be a fixed deterministic streaming reporter built from a finite controller and a fixed finite collection of initially blank one-dimensional multihead tape units. Suppose one external round reads one new symbol, performs at most a fixed c ordinary controller/unit operations, and ends with a Boolean report $R_C(x)$ for the same prefix x . Suppose also that each unit is realized by the no-delay Leong–Seiferas simulation and its fixed-rate Hartmanis–Stearns normalization. Then there is a total strict real-time machine N in the sense of definition 2.3 such that, for every finite word x ,*

$$N(x) = R_C(x),$$

where the empty-word value is any prescribed initial report. In particular, the reporter of proposition 6.1 has a strict realization P with $P(x) = 1$ exactly when $x = x^R$, including $x = \varepsilon$.

Proof. First make each source round a fixed-shape schedule. Store the arriving symbol in finite control, add a finite program counter and a ‘done’ latch, and pad a branch that finishes in fewer than c operations with no-ops. All data-dependent branching is still determined by the source's finite control and currently scanned finite-alphabet symbols; padding neither changes the source configuration nor its end-of-round report.

For each logical h -head work unit, instantiate the deterministic Leong–Seiferas simulator. It consumes one unit command and returns the tuple of newly scanned symbols without delay. Give different logical units disjoint fixed families of simulator tapes and take the finite product of their controllers, the padded source controller, its program counter, and its report bit. During a selected-tape command, the target source transition performs the selected write and move; every untouched tape is instructed to rewrite its scanned symbol and stay. Parallel commands for disjoint unit simulators are combined in the same product cycle.

The resulting external round has a fixed machine-dependent bound on its ordinary cycles and reads at most its one stored external symbol. Leong and Seiferas state the Hartmanis–Stearns normalization in precisely this form: a fixed-rate composite step is simulated by a conventional Turing-machine step with an enlarged finite tape alphabet and *without time loss* [3, p. 167]. Choose one rate exceeding the padded source schedule and all simulator constants. The normalized transition therefore consumes exactly one new input symbol and finishes with the report for that same prefix; it introduces neither a fixed lag nor a post-input step.

For completeness, the round-boundary invariant after a prefix x asserts simultaneously:

- (a) the simulated controller and every logical tape-unit state equal those of C after x ;
- (b) every simulator decodes to the corresponding logical tape contents and all of its head positions;
- (c) the product program counter is at the round boundary and exactly $|x|$ external symbols have been consumed; and
- (d) the finite-control report bit is $R_C(x)$.

All simulator tapes are initially blank. Leong and Seiferas address the apparent startup marker problem explicitly: the finite controller initially pretends that the fixed ‘LEFT’, ‘RIGHT’, and ‘SYNCH’ markers lie beneath the blank-tape heads [3, p. 170]. Hence the invariant holds before

the first command. Their simulation and no-time-loss speedup prove its one-round preservation, including the data-dependent branches, so induction proves (a)–(d) after every prefix. There is no warm-up interval to absorb: the selected source theorem starts from the first command. Any transition entries outside this reachable invariant are filled arbitrarily, making the transition function total without changing a reachable run; these unreachable entries carry no proof obligation beyond totality.

Finally, if the source convention omits the empty report, add a fresh initial control state with the prescribed acceptance value. On the first symbol it executes exactly the old initial state's accelerated transition and enters the corresponding old product state; no transition returns to the fresh state. Thus every nonempty run is unchanged and the empty computation has the chosen answer. For palindromes choose that answer to be true. The state set, alphabets, number of tapes, padding bound, and speedup rate are all fixed and finite, which proves the strict compiler-source conclusion. $\square$

Lemma 6.3 (Even-palindrome identity). *For every binary word x ,*

$$x \in \{ww^R : w \in \{0,1\}^*\} \iff x = x^R \text{ and } |x| \text{ is even.}$$

The equivalence includes $x = \varepsilon$.

Proof. If $x = ww^R$, then

$$x^R = (ww^R)^R = (w^R)^R w^R = ww^R = x \quad \text{and} \quad |x| = 2|w|.$$

Conversely, suppose $x = x^R$ and $|x| = 2n$. Write $x = x_1 \cdots x_{2n} = wv$, where $w = x_1 \cdots x_n$ and $v = x_{n+1} \cdots x_{2n}$. For $1 \leq i \leq n$, reversal symmetry gives

$$v_i = x_{n+i} = x_{2n+1-(n+i)} = x_{n+1-i} = w_{n+1-i}.$$

Thus $v = w^R$ and $x = ww^R$. When $n = 0$, this gives $w = v = \varepsilon$ and hence $x = \varepsilon$, so the empty word is included. $\square$

Lemma 6.4 (Reversal invariance).

$$\text{EPAL}_2^R = \text{EPAL}_2.$$

Proof. Every member has the form ww^R , and $(ww^R)^R = (w^R)^R w^R = ww^R$. Thus reversal fixes every member pointwise; since reversal is an involution, the two languages are equal. This includes $\varepsilon = \varepsilon\varepsilon^R$. $\square$

Lemma 6.5 (Parity product). *Suppose a strict real-time machine P reports after every prefix x , including the empty prefix, whether $x = x^R$. Then a strict real-time machine P_{even} recognizes exactly EPAL_2 .*

Proof. Take the product of P 's finite control with one parity bit. Initialize the bit to even and flip it once in the same transition that consumes each new input symbol; leave P 's tapes and tape actions unchanged. Accept exactly when P 's report is true and the parity bit is even.

Induction on the consumed prefix shows simultaneously that the projected P -configuration is its original configuration after that prefix and that the bit is even exactly when the prefix length is even. Hence the product accepts x exactly when $x = x^R$ and $|x|$ is even, which is EPAL_2 by lemma 6.3. Initially both predicates are true, so the empty word is accepted without an extra transition. $\square$

Theorem 6.6. *The even-length binary palindrome language has a parsing expression grammar:*

$$\boxed{\text{EPAL}_2 = \{ww^R : w \in \{0,1\}^*\} \in \text{PEG}.}$$

Proof. Apply lemmas 6.2 and 6.5. This gives a strict real-time multitape machine for EPAL_2 . By theorem 5.1, $\text{EPAL}_2^R \in \text{PEG}$. Lemma 6.4 gives $\text{EPAL}_2^R = \text{EPAL}_2$, so the result follows. $\square$

Corollary 6.7. *Conjecture 7 of Loff, Moreira, and Reis [4] is false.*

Proof. Their conjecture states exactly that $\{ww^R : w \in \{0,1\}^*\} \notin \text{PEG}$ [4, Conjecture 7]. The language is exactly EPAL_2 , while theorem 6.6 proves $\text{EPAL}_2 \in \text{PEG}$. $\square$

7 Formal verification

The simulation in sections 3 and 4 and the sufficient scaffold-to-PEG direction used by theorem 5.1 have been formalized in Lean 4. The development models the operational core of scaffolding automata, finite ordinary complete-match PEG semantics, and the finite grammar construction. The complete Lean 4 sources are available in the public source directory. The development proves:

- (i) preservation of `CleanEmpty`;
- (ii) correctness of each push, keep, and pop, including pop on an empty stack;
- (iii) equality of all source stacks and decoded scaffold stacks after every input word;
- (iv) equality of each source tape and the tape reconstructed from its two decoded scaffold roots and scanned symbol, including equality at every integer offset;
- (v) equality of the strict multitape machine’s language and the compiled scaffold automaton’s language;
- (vi) exact state, label, path, and neighbourhood behavior for the generated grammar, including same-position `SELF` recursion;
- (vii) the sufficient direction of Loff–Moreira–Reis Theorem 16 for every finite distance-bounded scaffolding automaton; and
- (viii) the reverse-language PEG containment for strict real-time multitape machines, plus an EPAL implication conditional on a strict real-time multitape EPAL recognizer.

The principal formal endpoints are:

- `Automaton.run_top`: the scaffold top after every word is its length, proving one appended node per input symbol;
- `realtime_transfer`: the structural induction relating the source and compiled runs;
- `realtime_transfer_tape`: equality of each source tape with the tape reconstructed from the scaffold;
- `realtime_transfer_tape_contents`: agreement at every signed cell offset;

- `realtime_transfer_wellFormed_decides`: distance-two typing, absence of self-targets, and language equality;
- `scaffold_to_peg`: every finite distance-bounded scaffolding automaton, including one using `SELF`, has its reverse language in the finite PEG class;
- `revRT_multitape_subset_PEG`: the machine-checked strict-real-time containment used by theorem 5.1;
- `epal2_inPEG_of_multitape_recognizer`: the conditional EPAL implication.

The development uses Lean 4.30.0 without Mathlib and contains no admitted proofs or declared axioms. Its theorem dependencies are Lean’s standard logical axioms. The operational formalization is polymorphic over its types. At the PEG boundary, exhaustive finite carriers produce a genuinely finite nonterminal type using bounded port paths and transport it to a literal `Fin n`. The Loff–Moreira–Reis theorem is not claimed as a new mathematical result: this repository newly machine-checks its sufficient direction.

For the EPAL theorem, Lean verifies the complete implication from a finite strict real-time multitape machine together with an exact proof that it decides EPAL_2 , through the scaffold construction, to $\text{EPAL}_2 \in \text{PEG}$. The source-verified Slisenko theorem, Leong–Seiferas ordinary-machine implementation, and fixed-rate speedup establish that machine premise. The Loff–Moreira–Reis properness witness is likewise source-verified. These classical results are not re-mechanized here; that boundary records the division of assurance, not a defect in or withdrawal of the EPAL and proper-containment conclusions.

As a separate implementation check, a literal interpreter of the published scaffold transition model was tested with a transferred four-stack middle-symbol machine. Exhaustive tests through length 14, longer structured and random tests, and eight deliberately corrupted variants exercise the unfolding and port-word resolution independently of the Lean development.

8 Discussion

The transfer rests on a simple resource principle: the number of physical updates and the number of logical memory roots updated need not coincide. A scaffold node has a product label and a fixed tuple of named ports, so one append can advance every root in a fixed finite family. The two-stack zipper then converts local movement on each work tape into exactly this form.

This also explains why an explicit midpoint is unnecessary in the palindrome application. The scaffolding automaton does not discover the midpoint by a single stack turn. It simulates the distributed schedule already present in the classical real-time recognizer.

The degree is a fixed parameter of the scaffolding automaton, not a universal bound on the model. In the twenty-tape realization the construction uses degree 40 and distance two. No pointer-identity test is needed: the transition reads only labels along named paths in the published tree unfolding.

9 Conclusion

A fixed tuple of persistent stack roots can be updated by one appended scaffold node. Through the ordinary two-stack representation of a Turing tape, this gives an exact letter-synchronous simulation of strict real-time multitape computation. The Loff–Moreira–Reis characterization

therefore places the reversal of every such language in PEG. Applied to the classical real-time palindrome recognizer, the transfer yields a PEG for even-length binary palindromes.

The assurance boundary is exact. Lean machine-checks the compiler equality $L(A_M) = L(M)$, the sufficient scaffold-to-PEG construction, the strict-real-time containment, and the implication from an exact strict real-time multitape EPAL recognizer to $\text{EPAL}_2 \in \text{PEG}$. The source-verified Loff–Moreira–Reis, Slisenko, Leong–Seiferas, and fixed-rate speedup results supply the properness and recognizer premises. Thus the EPAL and proper-containment conclusions are established, while the classical source layer is not re-mechanized in this repository. The enormous generic PEG is not expanded here.

References

- [1] Bryan Ford. Parsing expression grammars: A recognition-based syntactic foundation. In *Proceedings of POPL 2004*, pages 111–122. ACM, 2004.
- [2] Zvi Galil. Palindrome recognition in real time by a multitape Turing machine. *Journal of Computer and System Sciences*, 16(2):140–157, 1978.
- [3] Benton L. Leong and Joel I. Seiferas. New real-time simulations of multihead tape units. *Journal of the ACM*, 28(1):166–180, 1981.
- [4] Bruno Loff, Nelma Moreira, and Rogério Reis. The computational power of parsing expression grammars. *Journal of Computer and System Sciences*, 111:58–72, 2020. <https://doi.org/10.1016/j.jcss.2020.01.001>.
- [5] Arnold L. Rosenberg. Real-time definable languages. *Journal of the ACM*, 14(4):645–662, 1967.
- [6] A. O. Slisenko. A simplified proof of the real-time recognizability of palindromes on Turing machines. *Zapiski Nauchnykh Seminarov LOMI*, 60:123–139, 1976. English translation in *Journal of Soviet Mathematics*, 15:68–77, 1981.