

Real-Time Multitape Languages Transfer to Parsing Expression Grammars

via Persistent Stacks in Scaffolding Automata

Joshua Gay*

5 August 2026

Abstract

We prove that reversing any language recognized by a deterministic machine with finite control, finite alphabets, and a fixed finite number of ordinary unbounded work tapes in strict real time yields a parsing expression grammar. Each tape becomes a two-stack zipper, and one appended scaffold node updates all persistent stack roots simultaneously. This gives

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

Properness uses the online lower bound of Loff–Moreira–Reis. Combining the transfer with the classical real-time palindrome theorem and its ordinary single-head multitape realization yields

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

disproving Conjecture 7 of Loff–Moreira–Reis. Lean 4 checks the new machine-to-scaffold construction, the sufficient scaffold-to-PEG direction, their containment composition, and the final implication from an exact strict-real-time EPAL recognizer. The classical recognizer premise and the properness premise are checked against the cited statements but are not re-mechanized.

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*

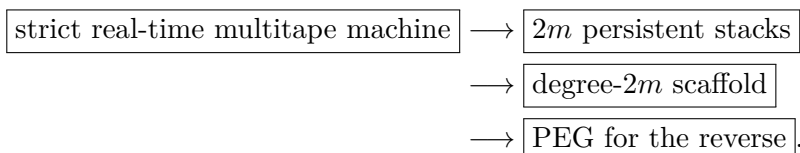
$$L^R \in \text{PEG}.$$

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.

The proof path is therefore



The construction is effective, although the generic PEG obtained by direct expansion is not expected to be compact or human-readable.

Slisenko constructs a real-time multitape palindrome recognizer [6]. Leong and Seiferas give an ordinary single-head multitape implementation [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 $L^R = \{x^R : x \in L\}$. For a language class $\mathcal{C}$, define

$$\text{rev}(\mathcal{C}) = \{L^R : L \in \mathcal{C}\}.$$

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. Adjoin a sentinel $\emptyset$ and set

$$e_j(\emptyset) = \emptyset, \quad \lambda_j(\emptyset) = \perp.$$

An unlabelled node also has component $\perp$. Following an invalid port path yields $\emptyset$, exactly as the missing-edge clause in Loff–Moreira–Reis Definition 13 requires.

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. Define

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

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.*

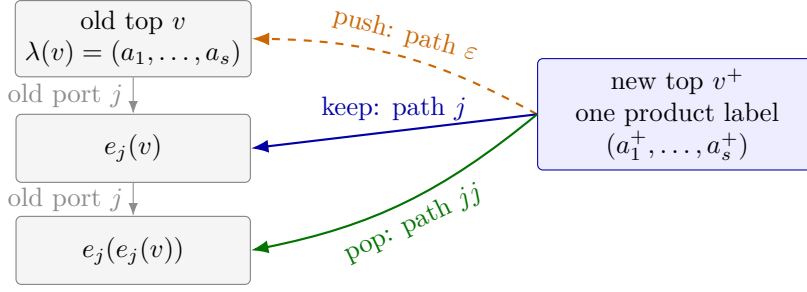


Figure 1: For one component j , the selected operation chooses one of three targets already visible within distance two of the old top. One physical node carries the independently selected component and port for every j .

Proof. The three rows are literal list equations. Push gives $D_j(v^+) = aD_j(v)$; keep copies the old head and tail; and a nonempty pop copies the head and tail of $D_j(e_j(v))$. For an empty pop, **CleanEmpty** makes that copied head $\perp$, so no stale symbol reappears. The same three cases preserve strict backwardness and **CleanEmpty**. Appendix A gives the sentinel-explicit calculation and a simultaneous two-stack example. $\square$

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. Here $\sqcup$ is an ordinary tape-alphabet symbol; $\perp$ means that a finite stack has no next element and is not a tape symbol. Initially $L_i = R_i = \varepsilon$ and $c_i = \sqcup$ on every tape.

For extensional comparison with the source tape, define

$$\text{cell}_{L,c,R}(z) = \begin{cases} c, & z = 0, \\ L_{-z}, & z < 0 \text{ and that entry exists,} \\ R_z, & z > 0 \text{ and that entry exists,} \\ \sqcup, & \text{otherwise.} \end{cases} \quad (3)$$

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

$$\begin{aligned} \text{right: } (L_i, c_i, R_i) &\mapsto (b_i L_i, \text{head}_{\sqcup}(R_i), \text{tail}(R_i)), \\ \text{left: } (L_i, c_i, R_i) &\mapsto (\text{tail}(L_i), \text{head}_{\sqcup}(L_i), b_i R_i), \\ \text{stay: } (L_i, c_i, R_i) &\mapsto (L_i, b_i, R_i). \end{aligned} \tag{4}$$

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 (4). 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 (4) assigns one stack operation to each zipper half. A direct calculation shows that right and left moves translate every reconstructed offset by one while recording the symbol just written, and that stay changes only offset zero. Induction over the input prefix therefore gives equality at every integer offset, including both implicit blank tails. The three extensional equations are written out in Appendix B. $\square$

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 . Every constant function c_M is $o(n/(\log n)^2)$. By lemma 5.2, neither lower-bound orientation can therefore 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: with m tapes the degree is $2m$, the tuple-label carrier grows exponentially in m , and the family of depth-two neighbourhoods grows exponentially again in that carrier. This blow-up belongs to the direct generic compiler, not necessarily to the smallest PEG for the language. We establish existence and effectiveness; we do not print the resulting grammar.

6 Even-length palindromes

Let

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

The only external premise needed here is stated directly in the source model of definition 2.3. It is a classical imported theorem: the proof below audits the source-model correspondence, while the source theorem itself is not re-mechanized here.

Theorem 6.1 (Classical palindrome interface (classical, imported)). *There are finite m, Q, T and a total strict real-time m -tape machine P such that, for every $x \in \{0, 1\}^*$, the state reached after exactly $|x|$ transitions is accepting if and only if $x = x^R$. The statement includes $x = \varepsilon$: the initial state is accepting.*

Proof. Slisenko proves real-time recognizability of palindromes with one new input letter per tick. Determinism and the absence of an endmarker or length advice make the report at tick $|x|$ a verdict for the prefix x , whether or not more symbols later arrive. Leong and Seiferas realize each fixed multihead tape unit by finitely many ordinary single-head tapes without delay, and their fixed-rate normalization compresses the uniformly bounded work of one source round into one conventional transition. The report bit is retained in finite control; a fresh initial state supplies the correct empty-word verdict.

Appendix C gives the clause-by-clause source correspondence, primary-source locations, same-prefix argument, and full fixed-round strictification invariant. It is part of the proof, not an appeal to terminology alone. $\square$

Lemma 6.2 (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

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

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 6.3 (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 6.4 (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 6.2. Initially both predicates are true, so the empty word is accepted without an extra transition. $\square$

Theorem 6.5. *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 theorem 6.1 and lemma 6.4. This gives a strict real-time multitape machine for EPAL_2 . By theorem 5.1, $(\text{EPAL}_2)^R \in \text{PEG}$. Lemma 6.3 gives $(\text{EPAL}_2)^R = \text{EPAL}_2$, so the result follows. $\square$

Corollary 6.6. *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 6.5 proves $\text{EPAL}_2 \in \text{PEG}$. $\square$

7 Formal verification

The reviewed Lean 4 source snapshot is commit [20ef2ef5f14274e37ac0dd40af0f944ff34885ce](https://gitlab.com/insight-forge/boxlab/fprd-lab/-/tree/20ef2ef5f14274e37ac0dd40af0f944ff34885ce)¹. The same source directory on the maintained branch is available here². It uses Lean 4.30.0 without Mathlib.

Layer	Conclusion	Assurance in this release
Machine to scaffold	$L(A_M) = L(M)$, exact tapes and one node per symbol	Machine-checked in Lean
Scaffold to PEG	LMR sufficient direction and $\text{rev}(\text{RT-MTTM}) \subseteq \text{PEG}$	Machine-checked in Lean; the LMR construction is formalized here, not claimed as a new theorem
Properness	$\text{rev}(\text{RT-MTTM}) \subsetneq \text{PEG}$	LMR lower bound checked against the cited classical statement but not mechanized
Even palindromes	$\text{EPAL}_2 \in \text{PEG}$	The implication from an exact strict recognizer is machine-checked; the recognizer interface is checked against the cited classical statements but not mechanized

The principal endpoints are:

- `realtime_transfer_tape_contents`;
- `scaffold_to_peg`;
- `revRT_multitape_subset_PEG`; and
- `epal2_inPEG_of_multitape_recognizer`.

The last is the conditional EPAL implication: no palindrome recognizer is introduced as an axiom. Exhaustive finite carriers and bounded port paths produce a literal finite nonterminal type, and the grammar handles automata that use `SELF`. Searches over the source find no `sorry`, `admit`, declared axiom, `unsafe`, `native_decide`, or `partial def`. For all four principal endpoints listed above, `#print axioms` reports only `propext`, `Classical.choice`, and `Quot.sound`. The occurrence of

¹<https://gitlab.com/insight-forge/boxlab/fprd-lab/-/tree/20ef2ef5f14274e37ac0dd40af0f944ff34885ce/publications/persistent-stacks-scaffolding/lean>

²<https://gitlab.com/insight-forge/boxlab/fprd-lab/-/tree/main/publications/persistent-stacks-scaffolding/lean>

`Classical.choice` comes from standard classical finite-carrier and transport constructions in the proof; it is not an added mathematical hypothesis about automata or languages.

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. An m -tape realization uses degree $2m$ 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 Loff–Moreira–Reis, Slisenko, Leong–Seiferas, and fixed-rate speedup results supply the properness and recognizer premises; they were checked against the cited classical statements but not mechanized. 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.

A The persistent update, literally

This appendix expands the compact proof of lemma 3.1. The sentinel conventions make every expression in the update table total:

$$e_j(\emptyset) = \emptyset, \quad \lambda_j(\emptyset) = \perp, \quad D_j(\emptyset) = \varepsilon.$$

If a port is missing, or a port word first becomes invalid, all remaining steps stay at $\emptyset$. Thus the expressions $e_j(v)$ and $e_j(e_j(v))$ have literal meanings even at the boundary.

Fix j . For a push, the new node has head a and tail root v , so

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

For keep, if $\lambda_j(v) = a$, the new node has head a and tail $e_j(v)$, hence reproduces $D_j(v)$. If $\lambda_j(v) = \perp$, both old and new decodings are empty. For pop, if $D_j(v) = aD_j(e_j(v))$, then the new head is $\lambda_j(e_j(v))$, its new tail is $e_j(e_j(v))$, and therefore

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

If $D_j(v) = \varepsilon$, **CleanEmpty** gives $\lambda_j(e_j(v)) = \perp$, including when $e_j(v) = \emptyset$, so the pop remains empty. Push cannot create an empty component. Keep inherits the clean-empty implication from v , and pop inherits it from $e_j(v)$. All named targets are at or below the old top, so they are strictly behind the newly appended node.

Here is the simultaneous two-stack case in one line. Suppose

$$D_1(v) = a\alpha, \quad D_2(v) = b\beta.$$

Push x on stack 1 and pop stack 2. The single new node has components x and $\lambda_2(e_2(v))$; its first port names v , while its second names $e_2(e_2(v))$. Consequently

$$(D_1(v^+), D_2(v^+)) = (xa\alpha, \beta).$$

The two logical updates are simultaneous because both components and both ports belong to the same appended node. If $\beta = \varepsilon$, the sentinel and **CleanEmpty** calculation above makes the second result empty.

B Extensional tape reconstruction

Let $Z = (L, c, R)$ be a zipper and let b be the symbol written before the head moves. Define the tape after the write but before the move by

$$\text{written}_{Z,b}(z) = \begin{cases} b, & z = 0, \\ \text{cell}_{L,c,R}(z), & z \neq 0. \end{cases}$$

If $Z_{\rightarrow}$, $Z_{\leftarrow}$, and $Z_{\circ}$ are the three zippers in (4), then for every $z \in \mathbb{Z}$,

$$\begin{aligned} \text{cell}_{Z_{\rightarrow}}(z) &= \text{written}_{Z,b}(z + 1), \\ \text{cell}_{Z_{\leftarrow}}(z) &= \text{written}_{Z,b}(z - 1), \\ \text{cell}_{Z_{\circ}}(z) &= \text{written}_{Z,b}(z). \end{aligned}$$

For the right-move equation, $z = -1$ reads the newly pushed head of bL , hence b ; $z = 0$ reads $\text{head}_{\sqcup}(R)$; positive offsets read the remaining tail of R ; and offsets below -1 read the old entries of L . These are exactly the four cases of the right-hand side. The left equation is the mirror image, and stay changes only offset zero. The default $\sqcup$ in $\text{head}_{\sqcup}$ and in (3) proves the equations even when either finite list is exhausted.

Initially $L = R = \varepsilon$ and $c = \sqcup$, so the reconstruction is blank at every integer offset. Induction on input length now maintains the following invariant for each work tape: the source control state and scanned symbol agree with the compiled control, and the source cell at every head-relative integer offset equals the cell reconstructed from the two decoded persistent stacks. This is the exact invariant asserted by theorem 4.2 and by the Lean theorem `realtime_transfer_tape_contents`.

C Classical palindrome interface audit

This appendix records how the classical statements discharge every clause of theorem 6.1. A longer source-audit note, including the rejected attempt to reconstruct the final theorem from positive-report sublemmas alone, is public with the paper.³

Compiler requirement	Classical evidence	Result
Deterministic computation	Leong–Seiferas model a tape unit by a deterministic command transducer and its real-time simulator by finite deterministic control; Slisenko gives a fixed Turing-machine algorithm.	Match
Fixed finite tapes and heads	Slisenko’s program has a fixed finite family of multihead units. Leong–Seiferas Corollary 3.6 replaces any fixed h -head one-dimensional unit by $4h - 4$ single-head tapes; their p. 166 twenty-tape count for Slisenko is corroborating, not needed quantitatively.	Match
Initial storage	Leong–Seiferas Theorem 3.1 and p. 170 start the simulator tapes blank; finite control only pretends that fixed LEFT, RIGHT, and SYNCH markers lie beneath the initial heads.	Match
Uniform work per arriving symbol	Slisenko uses one new input letter per tick and fixed linear speedup. Leong–Seiferas p. 167 uses a fixed rate k and the Hartmanis–Stearns normalization to replace a composite step by one conventional step without time loss.	Match after normalization
Answer for the prefix just read	Slisenko’s final, unnumbered recognition theorem and English summary (pp. 123 and 139), together with the one-letter-per-tick convention before Proposition 4.6 (p. 137), give the verdict at that tick. The same-prefix argument below removes any extension ambiguity.	Match
No unbounded finishing computation	A new letter arrives every tick and the verdict for the prefix is already present at its report point; there is no endmarker or post-input interval.	Match
Finite alphabets and control	The source programs, head sets, rates, tapes, and alphabets are fixed. The speedup enlarges the alphabet only by a fixed finite-radius encoding.	Match

The remaining source pins are Slisenko’s reporting conditions on p. 132 and Proposition 4.4 on p. 134, and Leong–Seiferas Theorems 3.1 and 3.5 and Corollary 3.6 on pp. 170, 173, and 174. Leong–Seiferas supply the storage realization; palindrome correctness comes from Slisenko’s final theorem.

Same-prefix timing

Fix a prefix u and regard u itself as a legitimate finite input. Slisenko’s real-time theorem decides it at tick $|u|$. The deterministic machine has received exactly the same symbols and no endmarker, length advice, or preloaded suffix after $|u|$ ticks on inputs u and uv . Its configuration is therefore identical in the two runs. Hence the report at each tick is the two-sided verdict for the prefix read through that tick. This argument uses the final recognition theorem; it does not infer rejection correctness from Slisenko’s positive-report propositions.

Fixed-round strictification

Consider a fixed deterministic streaming reporter whose round receives one symbol, performs at most c controller or tape-unit commands, and returns a Boolean report for that same prefix. Pad every branch to c program positions, retaining the arriving symbol, a program counter, branch results, and a done bit in finite control. Positions after completion are no-ops.

Instantiate the deterministic Leong–Seiferas simulator for each logical multihead unit, give different units disjoint finite tape families, and take the finite product of their controllers with the padded reporter. Corollary 3.6 supplies the fixed ordinary single-head simulations. The

³<https://gitlab.com/insight-forge/boxlab/fprd-lab/-/blob/main/docs/notes/epal-peg-proof-closure-20260731.md>

blank-tape initialization of Theorem 3.1 establishes the base configuration without a startup phase.

Choose one fixed speedup rate large enough for the padded schedule and every unit simulator. The no-time-loss normalization cited on Leong–Seiferas p. 167 compresses the product round into one conventional global transition over a fixed enlarged alphabet. A selected tape performs its prescribed write and move; every other tape rewrites its scanned symbol and stays. Put the report bit in the finite control.

After a prefix x , maintain simultaneously:

- (a) equality of the source and simulated controllers and logical unit states;
- (b) equality of every decoded tape content and logical head position;
- (c) a boundary program counter and exactly $|x|$ consumed symbols; and
- (d) equality of the finite-control report bit and the reporter’s answer for x .

Blank initialization proves the base case. The no-delay unit simulations, padding, and fixed-rate normalization prove one-round preservation, including data-dependent branches. Induction gives the same report after every prefix with no output lag or finishing computation. Fill unreachable transition-table entries arbitrarily to obtain a total transition function. If the historical convention omits an empty report, a fresh accepting initial state delegates its first-symbol transition to the old initial transition and is never re-entered. This proves exactly the strict interface in theorem 6.1.

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*, 68:123–139, 1977. English translation in *Journal of Soviet Mathematics*, 15:68–77, 1981. <https://doi.org/10.1007/BF01404109>.

Afterword

Real-time multitape languages transfer to parsing expression grammars

Methodological and contribution addendum. The paper's mathematical scope and review status remain as stated in the paper.

The Lab's approach

FPRD means **Finitely Presented Recognition Dynamics**. Its starting point is to distinguish a finite description, the possibly infinite process it generates, and what an observer can learn from that process. Equal answers need not come from equal internal structures. Conversely, an infinite state space may have a short, exact description and useful coordinates; failure to compress it into finitely many states need not end the investigation.

A small example: balanced parentheses

The grammar $S \rightarrow (S)S \mid \epsilon$ presents the language of balanced parentheses. To recognize a prefix, keep the number of unmatched opening parentheses, or a failure state if a closing parenthesis arrived too early. An opening parenthesis increments the count. A closing parenthesis decrements a positive count; at zero it enters failure. Failure persists. At the end, accept exactly when the count is zero.

The prefixes $($ and $(($ leave the same future task: any suffix that completes one completes the other. The prefix $(($ leaves a different task: $)$ completes the first two prefixes but not this one. These future tasks are **residuals**. There are infinitely many counts, yet a finite set of update rules generates their dynamics. This is a classical automata example, not a discovery of FPRD Lab. It illustrates the research question: which coordinates preserve all relevant futures, and what does updating or reading those coordinates cost?

From viewpoint to proof

In practice, the Lab fixes the observable and the allowed operations, compares presentations of the same problem, and seeks an exact translation or invariant. A failed shortcut is useful when a counterexample identifies the missing coordinate or hypothesis. Small computations test particular uncertainties; a proof must explain why the result holds beyond those tests. Costs, relation sides, boundary conditions, and the direction of each translation remain explicit.

AI-assisted exploration, source comparison, executable checks, and adversarial reconstruction support this work. Their agreement is not automatically independent evidence: shared assumptions must be exposed. A formal development certifies its stated formal theorem, not every interpretation attached to it. Each paper retains its own evidence and review status.

The approach is most useful when a poor representation hides a reusable construction, a compatibility condition, or a sharp obstruction. Outside recognition theory, it can guide the search for a structural reformulation without producing a literal residual dynamical system. The account below distinguishes that methodological influence from the ingredients of the proof.

Contribution to this publication

This addendum describes the current HTML reading edition. The preceding PDF pages retain their earlier edition; the afterword does not silently revise them.

FPRD's contribution to this paper

The key question was how much logical work one legal scaffold append can represent. Treating “one new physical node” as “one new logical cell” would impose a restriction that the source machine model does not require.

The Lab supplied an explicit coordinate representation: one node carries a finite tuple of stack-top symbols, with separately named ports for the corresponding persistent tails. Push, keep, and pop become bounded observations and port choices. A distance-two neighborhood suffices. The representation invariant also handles empty stacks, where a careless decoder could expose stale data.

The familiar two-stack representation of a work tape then becomes useful in this precise setting. Two persistent stacks per tape, together with the scanned symbol in finite control, translate each strict real-time global transition into one scaffold step. The substantive bridge is this letter-for-letter simulation with fixed resources, not the mere observation that tapes can be encoded as graphs.

The method also forced an interface audit at the other end. Applying a historical palindrome recognizer requires an answer for the prefix just read, fixed finite storage units, and no unbounded finishing computation. The paper spells out the source conventions and their normalization instead of treating every use of “real time” as interchangeable.

Inherited ingredients and the boundary of the contribution

Persistent stacks and the tape zipper are established constructions. Loff, Moreira, and Reis supply the exact scaffold/PEG characterization, including reversal. Slisenko supplies the real-time palindrome recognizer; Leong and Seiferas supply the ordinary single-head simulation used in the source-interface argument. The Lab's contribution is the explicit bounded scaffold realization and the checked composition of these interfaces.

Consequently, the theorem transfers the **reversal** of a strict real-time multitape language to PEGs. Even palindromes are an application because that language is reversal invariant. The paper does not infer a compact practical PEG, a classification of PEG languages, or unrestricted simulation of machines with post-input work.

The paper records a Lean development and distinguishes it from the historical source audit. This afterword adds no formal-verification or review credit. It identifies where the presentation viewpoint became an actual machine construction.

Where to inspect the contribution: the paper, Sections 3–7 and Appendices A–C, especially the update table, tape reconstruction, and same-prefix timing argument.