

Persistent Receipts and Simultaneous Records: A Direct Scaffold for Even Palindromes

Joshua Gay
FPRD Lab, Insight Forge

21 August 2026

Abstract

Loff, Moreira, and Reis conjectured that the language $\text{EPAL}_2 = \{ww^R : w \in \{0,1\}^*\}$ has no parsing expression grammar. That conjecture has already been refuted by an FPRD Lab transfer through strict real-time multitape machines. Here we give an independent direct route and make explicit the persistent memory discipline executed by the scaffold.

The recognizer maintains the longest even-palindromic suffix and the reversed context of its current occurrence. Typed receipts name historical suffix-link occurrences directly; persistent receipt ropes update that context by a bounded number of sequence operations. The construction then encounters a genuine same-step cycle: a fresh palindrome owns a fresh rope whose terminal receipt points back to that palindrome. To express this without smuggling in mutation, we define bounded simultaneous record transducers. Their fixed finite fresh-batch graphs compile into single scaffold nodes using SELF together with finite slot tags. Kaplan–Tarjan real-time catenable steques supply the sequence operations, and an opaque-atom staging theorem shows why their acyclic structural allocation can be tied to the fresh owner only at the simultaneous encoding boundary.

The resulting finite scaffolding automaton recognizes EPAL_2 , and the Loff–Moreira–Reis correspondence yields $\text{EPAL}_2 \in \text{PEG}$ without a Turing machine or Slisenko’s palindrome theorem. The paper is written for formal-language, automata, parsing, and adjacent mathematics researchers; it separates the word recurrence, persistent representation, physical scaffold, and executable checks. The package is released for specialist review. The membership result is not first, and no novelty claim is made pending broader literature review.

Contents

I	Why this proof looks different	3
1	The question and the direct route	3
1.1	The general aim: presentations, dynamics, and observations	4
1.2	Assurance boundary	5
2	A reader’s map	5
3	A running palindrome trace	6

II	Persistent palindrome state	7
4	The occurrence-context recurrence	7
5	Typed mirror receipts	9
6	Why simpler representations fail	10
6.1	Why an immutable type table is not enough	10
6.2	Why raw persistent words are not enough	11
6.3	Why bounded bridge chasing is not enough	11
7	Receipt ropes and shadow transport	12
III	Turning bounded records into a scaffold	15
8	The cyclic allocation obstacle	15
9	Bounded Simultaneous Record Transducers	16
9.1	The scaffolding-automaton interface	16
9.2	The corrected source model	17
9.3	Tuple packing	17
10	Opaque-atom staging	19
11	A direct palindrome scaffold	20
IV	Result, meaning, and evidence	22
12	What the direct route contributes	22
12.1	Comparison of the two proof routes	22
12.2	Why the negative representation results matter	23
13	Modern source calibration	23
14	Executable evidence and limits	24
A	Notation and a closer reading of the running trace	25
B	Fixed allocation inventory	26

C Claim and reproducibility map

26

Part I

Why this proof looks different

1 The question and the direct route

Guiding question. Can a device that adds only one bounded-local graph node per input letter recognize a language whose obvious description compares two unbounded halves?

A parsing expression grammar (PEG) is a deterministic recognition formalism whose ordered choice and predicates differ fundamentally from the usual nondeterministic reading of a context-free grammar. Loff, Moreira, and Reis introduced scaffolding automata and proved that a language is recognized by a PEG exactly when its reversal is recognized by a finite scaffolding automaton [2]. They conjectured that

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

does not belong to PEG.

For a reader coming from automata theory, there are two useful cautions. A PEG is not a context-free grammar equipped with a preferred parsing algorithm: ordered choice is semantic, and failure may be used by predicates. Conversely, the theorem used here does not require us to synthesize a readable grammar rule by rule. It is enough to construct a finite scaffolding automaton. That automaton reads from left to right, appends one persistent node at each step, and bases its next transition on a fixed-radius view of the current top. The Loff–Moreira–Reis theorem then converts this finite recognition process into a PEG for the reversed language.

FPRD Lab has already refuted that conjecture by compiling strict real-time multitape machines into scaffolding automata and using a classical real-time palindrome theorem. The result here is not a first refutation. Its purpose is to remove the Turing-machine detour and identify the persistent data structure that the scaffold itself executes.

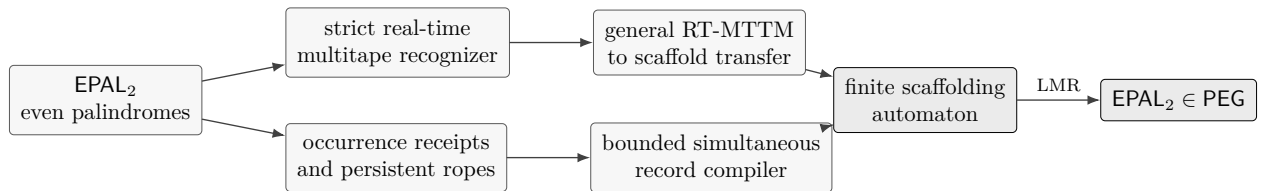


Figure 1: Two proofs of the same membership statement. The upper route is the earlier transfer proof. This paper develops the lower route. Its point is the direct persistent representation, not a new first refutation.

The lower route in Figure 1 produces two mathematically separable results. The application is a direct finite scaffold for even palindromes. The reusable intermediate result is a compiler: any transition which observes only bounded old structure and specifies a bounded finite graph of fresh records—even a graph with cycles among those fresh records—can be encoded by one scaffold node. The palindrome proof forces precisely the cycle that makes this strengthening necessary.

1.1 The general aim: presentations, dynamics, and observations

This paper builds directly on the framework introduced by Loff, Moreira, and Reis in *The Computational Power of Parsing Expression Grammars* [2]. Their scaffolding automata turn the history generated by a finite transition table into an explicit persistent graph, and their correspondence turns acceptance of that graph process into a PEG language. Our question is therefore not only whether EPAL_2 has some PEG. It is which finitely presented recognition dynamic makes palindrome comparison visible to that correspondence.

That question is representative of what we call *finitely presented representation dynamics* (FPRD). The basic factorization is

$$\text{finite recognition presentation} \longrightarrow \text{recognition dynamics} \longrightarrow \text{language or suffix observation}.$$

FPRD first asks what exact state space, named actions, topology, finite observations, and boundary behavior a finite presentation generates. It then asks which part of that richer object a chosen observation exposes. Changing the observation may merge states without changing the presentation; changing the presentation may change internal actions without changing the accepted language. Both phenomena are kept visible rather than quotiented away at the start.

The deterministic foundation separates four layers.

1. *Semantic algebra*: exact recognition behaviors and their constructors, quotiented only by exact behavior equality.
2. *Observational completion*: finite interfaces—for example, agreement on all inputs through a bounded length—their quotient maps, and compatible inverse threads.
3. *Presentation-generated dynamics*: a distinguished initial state and named actions extracted from, or supplied by, one finite presentation.
4. *Coordinate realization*: an injective state encoding, coordinates for every named action, and a uniform readout recovering recognition behavior from coordinates and input.

An implication between layers is a theorem obligation. A universal product embedding is not automatically a useful coordinate realization; a state encoding without a uniform readout is incomplete; language equality does not imply equality of labelled continuation actions. Conversely, two models may share a coordinate topology and action monoid while exposing different remaining-suffix observations.

In this paper the exact word recurrence and receipt equations supply the semantic behavior; bounded old-record and scaffold unfoldings are the finite interfaces; input letters generate the persistent occurrence dynamic; and the tuple-packed compiler supplies both an injective scaffold encoding and its uniform decoder. Acceptance is the final observation that the reversed context is empty. The companion RT-MTTM route studies a different presentation-generated dynamic with the same language observation. The two routes therefore complement one another even though both end at $\text{EPAL}_2 \in \text{PEG}$.

Main application (FPRD-T128; proved in Section 11). There is a finite scaffolding automaton recognizing the binary even-palindrome language. Consequently $\text{EPAL}_2 \in \text{PEG}$. This construction uses neither a strict real-time multitape machine nor Slisenko’s real-time palindrome theorem.

Stating the result now gives the reader a fixed target: all intervening definitions exist to explain

what information the single new scaffold node must contain and how that information is obtained locally.

1.2 Assurance boundary

The proof has four layers.

1. The longest even suffix and its reversed left context obey an exact three-case recurrence.
2. Typed mirror receipts and bridge-shadow identities turn that recurrence into a fixed collection of persistent rope operations.
3. Kaplan and Tarjan’s purely functional real-time catenable steque [1] supplies those rope operations with worst-case bounded structural work.
4. A bounded simultaneous record transducer compiles letter-for-letter into one finite scaffolding automaton.

The written arguments are mathematical proofs. The accompanying programs are finite corroborations and regressions, not proofs of the infinite statements. The later OCaml/Rocq implementations of Viennot et al. [3] are used as implementation calibration, not as the functional-correctness premise. No Lean reconstruction of this paper is claimed or required. External specialist review and novelty evaluation remain open.

2 A reader’s map

The argument moves through three representations of the same online state. Keeping them separate prevents most of the apparent paradoxes.

Word state.

For the prefix S already read, we factor $S = CP$ where P is its longest even-palindromic suffix and retain $\ell = C^R$. This level contains ordinary words and exact identities.

Persistent record state.

Palindrome occurrences, typed suffix-link receipts, and receipt ropes expose exactly the historical objects needed by the next transition. Sharing occurs here; a stored receipt is a pointer to a particular record, not merely the name of a palindrome string.

Physical scaffold state.

A single physical node tuple-packs all logical records born at one input time. A logical pointer is decoded as a physical historical node plus a finite slot tag.

The distinction between *type* and *occurrence* is especially important. The word 0110, for example, is one palindrome type. During an online run it may occur at many positions and under different surrounding contexts. A conventional mutable index may attach transitions to the shared type. An immutable scaffold cannot retroactively add a pointer to an old record, so it transports receipts associated with the occurrence that made a transition available.

We assume ordinary familiarity with words, palindromes, finite-state control, persistent immutable records, and inductive proofs of simulation. We do not assume prior familiarity with scaffolding automata, PEG semantics, palindromic trees, catenable steques, or the FPRD research program. Definitions are given at the first point where they carry proof weight. In diagrams a solid arrow is a logical pointer or implication; a dashed arrow is a decoded or represented relationship rather than an additional heap edge.

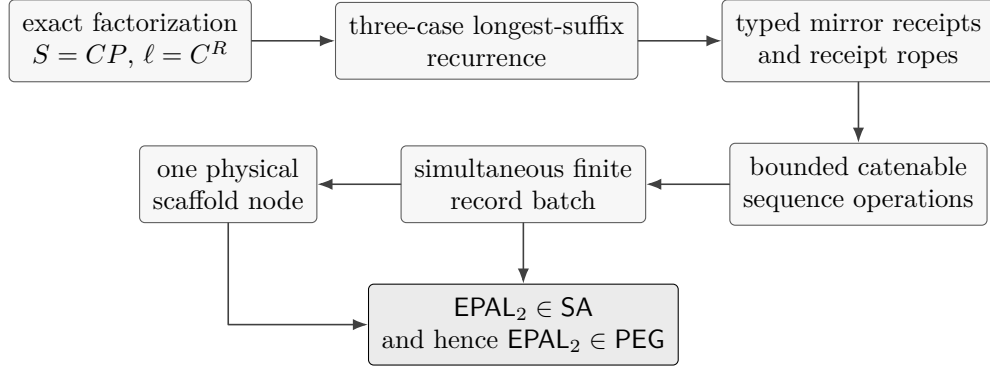


Figure 2: The proof dependency ladder. The upper row is word combinatorics; the lower row is representation and compilation. Executable checks in the public package test individual arrows but are not used as proofs of them.

3 A running palindrome trace

Before introducing the record machinery, we follow one word. For every input prefix S , let $P = P(S)$ be its longest even-palindromic suffix, write $S = CP$, and let $\ell = C^R$. The pair (P, ℓ) determines acceptance: $\ell = \epsilon$ exactly when the entire prefix is an even palindrome. It also contains enough word-level information to determine the next pair.

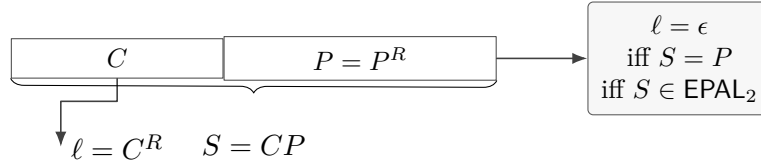


Figure 3: The occurrence-context factorization. The context is reversed so that the letter immediately preceding the current occurrence of P is at the front of ℓ .

Table 1 gives the complete checked trace for 00011001. The empty word is displayed as ϵ . “External” means that the new letter matches the first letter of the stored context, so it wraps the entire current palindrome. “Internal” means that the full P cannot be wrapped, but a proper even-palindromic suffix of P can. “Reset” means that no nonempty even suffix can be wrapped, so the new longest even suffix is empty.

Two rows are worth pausing over. After the second letter the word is 00, so $P = 00$ and the context is empty: the recognizer would accept at that point. The next 0 cannot externally wrap 00, because there is no letter to its left. Nevertheless the empty proper suffix can be internally wrapped, giving the new longest even suffix 00 at the last two positions and context 0. At the final row, the current $P = 001100$ is preceded by 0, not by the arriving 1. The proper suffix 00 is preceded inside P by 1, so it is wrapped to 1001; the remainder of the old palindrome is transferred into the context. That apparent nonlocal jump is the update that the receipt machinery must make bounded-local.

The case sequence

reset, external, internal, reset, external, external, external, internal

will reappear when we interpret the general recurrence and gives concrete names to the otherwise

i	old S	a	old P	old ℓ	case	new P	new ℓ
1	ϵ	0	ϵ	ϵ	reset	ϵ	0
2	0	0	ϵ	0	external	00	ϵ
3	00	0	00	ϵ	internal	00	0
4	000	1	00	0	reset	ϵ	1000
5	0001	1	ϵ	1000	external	11	000
6	00011	0	11	000	external	0110	00
7	000110	0	0110	00	external	001100	0
8	0001100	1	001100	0	internal	1001	1000

Table 1: Exact occurrence-context trace for 00011001. The accompanying checker computes the same eight case labels from the definitions. The table illustrates the recurrence; it is not evidence in place of its proof.

abstract objects.

Part II

Persistent palindrome state

4 The occurrence-context recurrence

Guiding question. *What is the smallest exact word state from which one new letter determines the next longest even-palindromic suffix?*

Maintaining a longest suffix is reminiscent of a failure-link automaton: when the current candidate cannot be extended, one tries a proper suffix. The difference here is that we cannot follow an unbounded chain during one input step. We therefore separate the combinatorial recurrence from the later question of how its selected transition is exposed by one bounded pointer. This section proves the recurrence with ordinary words only.

Let S be the input prefix read so far. Let $P(S)$ be its longest even-palindromic suffix and write

$$S = CP, \quad \ell = C^R.$$

The word ℓ is the left-context stack of the current occurrence; its first letter lies immediately to the left of P .

For an even palindrome P and a letter a , let $D_a(P)$ be the longest proper even-palindromic suffix Q for which

$$P = ZaQ,$$

and put $\omega_a(P) = Z^R$. Both are undefined when no such factorization exists.

The pair $(D_a(P), \omega_a(P))$ should be read as a factored transition. Its first coordinate is the longest proper suffix of P that the arriving a can wrap. Its second coordinate is exactly the material displaced from the palindrome into the reversed context when that transition is taken. It is important that $D_a(P)$ is selected by an occurrence of a immediately before the suffix, not merely by equality of palindrome types.

external match	internal direct	reset
$S = C'aP$ $Sa = C'aPa$ $\ell = a\ell_0 \mapsto \ell_0$	$P = ZaD_a(P)$ $P' = aD_a(P)a$ $\ell \mapsto Z^R\ell$	$D_a(P)$ undefined $P' = \epsilon$ $\ell \mapsto aP\ell$

Figure 4: The exhaustive three-case update at the word level. The external case consumes one context letter; the internal case prepends a selected chunk; the reset case moves the entire failed candidate into the context.

Theorem 4.1 (Occurrence-context factorization). *After reading a , the new longest even suffix P' and reversed context ℓ' are exactly*

$$(P', \ell') = \begin{cases} (aPa, \text{tail}(\ell)), & \ell = a\ell_0, \\ (aD_a(P)a, \omega_a(P)\ell), & \ell \notin a\Sigma^*, D_a(P) \text{ defined}, \\ (\epsilon, aP\ell), & \ell \notin a\Sigma^*, D_a(P) \text{ undefined}. \end{cases}$$

Moreover,

$$S \in \text{EPAL}_2 \iff \ell = \epsilon.$$

Proof. Every nonempty even-palindromic suffix of Sa has the form aQa , where Q is an even-palindromic suffix of S immediately preceded by a .

If the current longest suffix P is externally preceded by a , then $Q = P$ is maximal. Removing that external letter removes the first symbol of ℓ . Otherwise every eligible Q is a proper suffix of P , so maximality selects $D_a(P)$. Writing $P = ZaD_a(P)$ shows that the new context is CZ and hence has reversal $Z^R C^R = \omega_a(P)\ell$. If no eligible proper suffix exists, the longest even suffix is empty and the reversed full prefix is $aP\ell$.

Finally ℓ is empty exactly when C is empty, which is exactly when the whole prefix equals its longest even-palindromic suffix. $\square$

In Table 1, rows 2, 5, 6, and 7 are external matches; rows 3 and 8 are internal-direct updates; rows 1 and 4 are resets. The final equivalence in the theorem explains why row 2 is accepting and the final row is not. This interpretation also shows why keeping only the length of P would be insufficient: the internal case must identify a particular suffix and the chunk that precedes it.

The chunks $\omega_a(P)$ themselves satisfy a finite-alphabet recursion. If $L = \lambda(P)$ is the longest proper even-palindromic suffix and

$$P = RbL = LbR^R,$$

write $B_P = R^R$. Then

$$D_b(P) = L, \quad \omega_b(P) = B_P,$$

and for $a \neq b$,

$$D_a(P) = D_a(L), \quad \omega_a(P) = \omega_a(L)bB_P,$$

with the same definedness on both sides. Indeed, every proper even-palindromic suffix of P is a suffix of L ; the bridge letter b selects L itself, while every other letter delegates to L and adds the displayed intervening block.

This recursion is the first sign of persistent self-similarity. For the one letter b immediately before the suffix link, the desired transition is available at the current record. Every other alphabet coordinate is inherited from the suffix-link record and receives the same intervening block. Because the alphabet is fixed, the entire transition table has fixed width. The remaining difficulty is not the number of coordinates; it is transporting the correct historical record and the potentially long intervening word without walking through either one.

5 Typed mirror receipts

Theorem 4.1 reduces online recognition to catenating type-indexed chunks. It does not yet say how a fresh occurrence obtains its suffix-link record without a search. The needed object is a typed receipt.

A palindrome *type* is a word such as 001100. A palindrome *occurrence* is that word at a specified interval of the input history. The distinction is normally inexpensive in an online palindromic tree: a mutable dictionary may map a type and a letter to another type whenever the transition is discovered. An append-only scaffold has no such global update. The record available at the current top must already carry a bounded route to the historical occurrence that will certify its next suffix link.

The word “typed” in *typed receipt* does not mean a programming-language type annotation. It means that the receipt identifies the exact palindrome record required by the proof. A raw time coordinate says where some relevant substring ended; a typed receipt says which immutable record at that history node represents the required suffix-link occurrence.

Theorem 5.1 (Typed mirror receipt). *Suppose an update selects a longest suffix occurrence Q of the old input that is immediately preceded by a , and creates $Y = aQa$. Then*

$$\lambda(Y) = \begin{cases} aD_a(Q)a, & D_a(Q) \text{ defined,} \\ \epsilon, & D_a(Q) \text{ undefined.} \end{cases}$$

The palindrome $\lambda(Y)$ has a complete occurrence in the old input and a pre-existing birth record. One old typed pointer can therefore install the suffix-link field of Y .

Proof. Every proper nonempty even-palindromic suffix of $Y = aQa$ is aRa , where R is a proper even-palindromic suffix of Q immediately preceded in Q by a . Maximizing R gives the displayed formula. Since Y is a palindrome, $\lambda(Y)$ is also a proper prefix of Y ; that prefix ends before the newly appended final a , so the complete occurrence lies in the old input.

At the first occurrence of any nonempty palindrome H , it is the longest palindromic suffix of the prefix ending there. Otherwise a longer palindromic suffix would contain H as a proper prefix as well as a suffix, giving an earlier occurrence of H . The online construction has therefore already made a birth record for $\lambda(Y)$. The empty case uses one distinguished empty record. $\square$

Working picture. Reflect the proper suffix of the new palindrome across its center. Its mirror copy finishes before the newly appended last letter, so it is wholly old. The online construction

has already created a record when that palindrome first appeared. The receipt transports a pointer to that record rather than asking the next transition to rediscover it.

The word “typed” is essential. A raw endpoint does not identify the needed record at bounded depth.

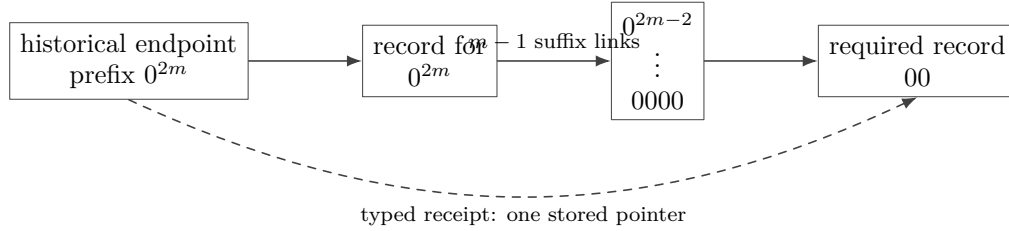


Figure 5: Endpoint information and typed information are different. The endpoint identifies a historical prefix, but its longest-palindrome record may be arbitrarily high above the required type.

Proposition 5.2 (Endpoint-only escape). *For $m \geq 2$, let $S_m = 0^{2m}1100$. Its longest even-palindromic suffix is 001100 and the suffix link is 00. The mirror occurrence of 00 ends at the historical prefix 0^{2m} , whose longest suffix-link chain is*

$$0^{2m}, 0^{2m-2}, \dots, 00, \epsilon.$$

Recovering 00 from that untyped endpoint takes exactly $m - 1$ suffix-link steps.

Proof. The suffix 001100 is an even palindrome, and no longer suffix of S_m can be a palindrome because any such suffix begins inside the initial zero block but contains its unmatched 11 asymmetrically. Its longest proper even suffix is 00. The prefix copy of that 00 selected by the mirror argument ends at time $2m$. At that time the input is 0^{2m} , whose longest even suffix is the whole prefix and whose proper even suffixes decrease by two zeros at each link. Starting at 0^{2m} reaches 00 after $m - 1$ links. $\square$

This is a representation-specific escape theorem, not a language or scaffold lower bound. It says that the positive construction must transport the typed record pointer itself.

6 Why simpler representations fail

The receipt construction was not guessed in its final form. Several natural representations preserve enough extensional information to describe the next palindrome but do not expose the needed immutable pointer within a bounded number of old-field traversals. Recording these failures serves two purposes. It explains why the positive representation carries apparently redundant occurrence data, and it keeps the propositions in their proper scope. None of them is a lower bound for PEGs, scaffolding automata, or palindrome recognition in general.

6.1 Why an immutable type table is not enough

If one record is created at the first birth of each palindrome type, it is tempting to copy a fixed-alphabet transition table from the suffix-link record, as a mutable palindromic tree would do. The target of a later transition, however, need not exist when the source type is born.

Proposition 6.1 (Future transition after type birth). *For $r \geq 1$, put $A_r = 0^{2r}$,*

$$P_r = A_r 1 1 A_r, \quad T_r = 1 A_r 1.$$

Then $D_1(P_r) = A_r$, so the transition selected by 1 has target $T_r = 1D_1(P_r)1$. The type P_r is born by the end of the prefix P_r , but T_r is not a factor of that prefix. It becomes available only in a later history such as $P_r 1 P_r$. Thus the immutable first-birth record of P_r cannot already point to every future transition out of P_r .

Proof. The word A_r is the longest proper even-palindromic suffix of P_r that is immediately preceded by 1, giving $D_1(P_r) = A_r$. Any occurrence of $T_r = 1A_r1$ contains two 1 symbols separated by $2r$ zeros. In P_r the only adjacent 1 symbols lie at its center, so no such factor occurs. In $P_r 1 P_r$, the separator 1 together with the initial block $A_r 1$ of the second copy gives an occurrence of T_r . This event is strictly later than the birth record of P_r . $\square$

The forced design decision is modest but decisive: transitions must be transported by occurrence-indexed receipts, or by some representation of equal strength. A canonical type node remains useful for explaining suffix relations, but it is not by itself a closed immutable update interface.

6.2 Why raw persistent words are not enough

A persistent rope can share the letters of a context perfectly. Receipts, however, depend on the palindrome around which those letters are interpreted.

Proposition 6.2 (Anchor dependence). *For the fixed one-letter block $w = 0$ and anchors $A_r = 0^{2r}$,*

$$\lambda(0A_r0) = A_r.$$

Consequently the same raw rope cell 0 requires infinitely many distinct typed receipt targets as its anchor varies.

Proof. The word $0A_r0 = 0^{2r+2}$ is an even palindrome. Its longest proper even palindromic suffix is $0^{2r} = A_r$. These target types are distinct as r varies. Hence a receipt attached only to the raw letter block cannot be correct for all anchors. $\square$

This proposition does not forbid structural sharing of the underlying word. It says that the positive object must be an *anchored* receipt path, and that later reuse must justify why the receipt shadow survives a legal change of physical anchor. Lemma 7.1 supplies exactly that justification.

6.3 Why bounded bridge chasing is not enough

One might defer the missing transition and repeatedly follow the suffix link, testing the bridge letter at each record. The following family makes the number of failed tests arbitrary.

Proposition 6.3 (Bridge-chasing escape). *For $r \geq 1$ and $d \geq 0$, let*

$$P_{r,d} = A_r(11A_r)^d, \quad A_r = 0^{2r}.$$

These are even palindromes. For $d > 0$,

$$\lambda(P_{r,d}) = P_{r,d-1} \quad \text{and} \quad b_{P_{r,d}} = 1,$$

whereas the bridge of A_r is 0. The search for the 0-transition from $P_{r,d}$ therefore makes exactly d failed bridge tests before reaching the variable target A_r .

Proof. The word $P_{r,d} = A_r 11 A_r 11 \cdots 11 A_r$ is unchanged by reversal, so it is an even palindrome. The suffix beginning after the first block $A_r 11$ is $P_{r,d-1}$ and is palindromic. No longer proper suffix is palindromic: a suffix beginning inside the first A_r has fewer initial zeros than terminal zeros and meets a 1 while its reversal is still inside the terminal zero block; a suffix beginning in the following 11 starts with 1 and ends with 0. Hence $P_{r,d-1}$ is the longest proper even-palindromic suffix. The letter immediately before it is the second 1 of the removed 11 block.

Thus each of the first d records rejects the requested bridge letter 0 and delegates to its suffix link. At $P_{r,0} = A_r$, the bridge letter before its longest proper even suffix is 0, so the recursion stops. There are exactly d failures, and d is unbounded. $\square$

The proposition refutes bounded bridge chasing for this representation. It does not refute recursively stored direct transitions: the final construction materializes the same recursion persistently, so the transition selected in a later step is already at bounded depth.

Attempt	What it preserves	Escape family	Information retained finally
Historical endpoint	where the mirror copy ends	$0^{2m}1100$: target 00 lies $m - 1$ links below the historical maximum	direct typed pointer to the required record
Immutable type-birth table	canonical suffix-link type	$P_r = 0^{2r}110^{2r}$: a needed target is born later	occurrence-indexed transition receipt
Anchor-free word rope	persistent raw context letters	the block 0 over anchors 0^{2r} needs different receipts	headered anchored receipt path
Bounded bridge chase	correct recursive selector equation	$P_{r,d}$ forces exactly d failures	persistently materialized direct/reset rope

Table 2: Representation lessons. Each escape concerns the interface in the first column, not the expressive power of the target automaton model.

These eliminations narrow the task to a positive one: define an anchored path whose cells carry typed receipts, and prove that the parts shared after an online update retain the same receipt trace even when their concrete palindrome anchors change. That is the purpose of receipt ropes.

7 Receipt ropes and shadow transport

Guiding question. *How can a persistent sequence share an arbitrarily long old tail when the palindrome at the tail's central anchor has changed?*

A receipt rope is not merely a word rope with annotations. It represents a walk through palindrome occurrences. Starting from an anchor A , each context letter wraps the current palindrome and each cell records the suffix link of the newly wrapped palindrome. The record at the front is a *header*: if a

tail operation makes the first wrapped palindrome current, its suffix-link receipt is already exposed. This promotion is what an ordinary list of letters lacks.

For an even palindrome A and word $w = c_1 \cdots c_k$, define the anchored path $\Pi(A, w)$ by

$$A_0 = A, \quad A_i = c_i A_{i-1} c_i.$$

Cell i stores $(c_i, \lambda(A_i))$. Its receipt trace is

$$\Lambda(A, w) = (\lambda(A_1), \dots, \lambda(A_k)),$$

and its headered form is

$$\hat{\Pi}(A, w) = (\lambda(A); (c_1, \lambda(A_1)), \dots, (c_k, \lambda(A_k))).$$

The header is the receipt required if the anchor becomes current; taking a tail promotes the first cell's receipt to the new header. Anchored paths obey the literal catenation law

$$\Pi(A, uv) = \Pi(A, u) \cdot \Pi(\text{wrap}_u(A), v).$$

The catenation law carries an important guard: the second path is anchored at the result of wrapping A by u . Two identical raw words do not therefore represent interchangeable receipt ropes at arbitrary anchors. Proposition 6.2 is the one-letter instance of this fact.

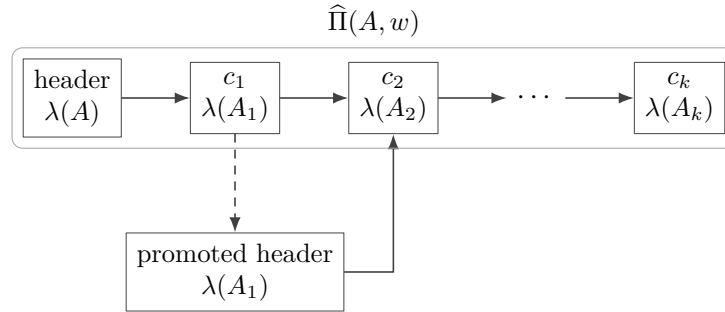


Figure 6: A headered receipt rope. Tail removes the first cell and promotes its stored receipt; the persistent remainder is shared. The positive proof must show that legal online reanchoring leaves the receipts in that shared remainder unchanged.

The physical palindrome states in a retained context tail can change after a prepend or reset. The next lemma shows that the typed receipt trace does not.

Lemma 7.1 (Bridge-shadow identity). *Suppose $H = RbP$, $\lambda(H) = P$, and P is the longest even-palindromic suffix of $w^R P$. If $w = \epsilon$, or if the first letter of w differs from b , then*

$$\Lambda(H, w) = \Lambda(P, w).$$

Working picture. The anchors H and $P = \lambda(H)$ begin with the same suffix-link choice as soon as the first context letter is not the bridge b . If their receipt traces first diverged later, the unequal bridge exposed at that first divergence could be reflected through the context already consumed. The reflection would form an even-palindromic suffix strictly longer than P in $w^R P$, contradicting the hypothesis that P is longest. The proof below adds indices and a strict length bound to this picture.

Proof. For a letter c and even palindrome X , put $G_c(X) = \lambda(cXc)$. Every proper even-palindromic suffix of X is a suffix of $\lambda(X)$. Hence

$$G_c(X) = \begin{cases} c\lambda(X)c, & c \text{ is the bridge before } \lambda(X), \\ G_c(\lambda(X)), & \text{otherwise.} \end{cases} \quad (1)$$

Write $w = c_1 \cdots c_k$, put $u_i = c_1 \cdots c_i$, and define

$$X_i = u_i^R H u_i, \quad Y_i = u_i^R P u_i, \quad N_i = |P| + i.$$

Both X_i and Y_i end in the same word $T_i = P u_i$, of length N_i . We prove for every $i \geq 1$ that

$$\lambda(X_i) = \lambda(Y_i) = K_i \quad \text{and} \quad |K_i| < N_i. \quad (2)$$

For $i = 1$, the hypothesis $c_1 \neq b$ and (1) give $G_{c_1}(H) = G_{c_1}(P)$. This common palindrome is a proper suffix of $c_1 P c_1$, so its length is at most $|P| < N_1$.

Assume (2) at $i < k$. Since K_i is shorter than the common suffix T_i , the letter immediately before its terminal occurrence lies inside T_i . It is therefore the same bridge letter in X_i and Y_i . Equation (1), with $c = c_{i+1}$, now gives the same next receipt on both sides: it is either $cK_i c$ when c is that common bridge, or $G_c(K_i)$ otherwise.

It remains to retain the strict length bound. In the second case $|G_c(K_i)| \leq |K_i| < N_i < N_{i+1}$. In the first case the only way $|cK_i c|$ could reach N_{i+1} is, by parity, that N_i is odd and $|K_i| = N_i - 1$. Then the common suffix has the form $T_i = cK_i$. Because K_i is a palindrome,

$$(u_i c)^R P = c u_i^R P = c T_i^R = c K_i c$$

is an even-palindromic suffix of $w^R P$ strictly longer than P : the word $u_i c$ is a prefix of w . This contradicts the maximality hypothesis. Thus (2) follows by induction, and its receipt equalities are precisely $\Lambda(H, w) = \Lambda(P, w)$. $\square$

The length inequality in (2) is not bookkeeping decoration. It ensures that the bridge inspected at the next step lies inside the suffix shared by both physical anchors. The parity subcase is the only point at which equality could occur; resolving it by constructing the forbidden longer suffix is what makes the shadow statement global rather than merely a one-cell observation.

For nonempty Y with suffix link L , write

$$Y = RbL = LbB.$$

When $D_c(Y)$ is defined, define the direct headered rope

$$\Omega_c(Y) = \hat{\Pi}(cD_c(Y)c, \omega_c(Y)).$$

When it is undefined, define the reset rope

$$Z_c(Y) = \hat{\Pi}(\epsilon, cY).$$

Theorem 7.2 (Receipt-rope recurrence). *Let ρ_b be the header receipt of $\Omega_b(Y)$. For $c \neq b$,*

$$\Omega_c(Y) = \begin{cases} \text{undefined,} & \Omega_c(L) \text{ undefined,} \\ \Omega_c(L) \cdot (b, \rho_b) \cdot \text{body } \Omega_b(Y), & \text{otherwise,} \end{cases}$$

and, whenever $Z_c(Y)$ is defined,

$$Z_c(Y) = Z_c(L) \cdot (b, \rho_b) \cdot \text{body } \Omega_b(Y).$$

In words, every off-bridge direct rope and every reset rope consists of three pieces already visible at bounded depth: the corresponding rope at the suffix link L , one bridge cell, and the body of the single bridge-matching rope at Y . The bridge-shadow identity is what makes the third piece literally shareable; without it the equation would describe only equal raw words and would not be a persistent pointer construction.

Proof. The chunk recursion above gives the raw-word factorizations. The initial $\Omega_c(L)$ or $Z_c(L)$ portion is literal. At its endpoint, the next b cell has the bridge receipt ρ_b . The remaining raw word is B . Lemma 7.1 identifies every receipt along that retained B tail with the body of $\Omega_b(Y)$. Definedness of the off-bridge direct rope is the same at Y and L because every proper suffix of Y lies under L . $\square$

Theorem 7.3 (Full abstract receipt closure). *Over a fixed alphabet, a fresh occurrence record, its complete direct/reset rope table, and the updated live context are obtained from old receipts by a fixed number of head, tail, singleton, inject, and catenation operations.*

The adjective “abstract” marks a clean interface boundary. The theorem has already eliminated suffix-link search, mutation, global interning, and random access to the old input. It assumes only a persistent sequence supporting a fixed collection of operations in worst-case bounded structural work. The next part supplies that sequence and explains how its fresh structural records are represented by the scaffold.

Proof. Suppose the update creates $P = aQa$ and Theorem 5.1 supplies $L = \lambda(P)$. Select the old direct rope $\Omega_a(Q)$ when $D_a(Q)$ is defined and the old reset rope $Z_a(Q)$ otherwise. Its body is nonempty; write it $(b, H)\mathcal{T}$. The fresh bridge data are

$$b_P = b, \quad \Omega_b(P) = (H; \mathcal{T}(a, P)). \quad (2)$$

The last receipt is P itself by the mirror-prefix factorization. The bridge entry (2), Theorem 7.2, and the finite alphabet construct every other direct or reset rope by a constant-size table update.

Let $\mathcal{K}(P, \ell) = \widehat{\Pi}(P, \ell)$ be the live context rope. The three cases of Theorem 4.1 give

$$\mathcal{K}' = \begin{cases} \text{tail } \mathcal{K}(P, \ell), & \ell = a\ell_0, \\ \Omega_a(P) \cdot \text{body } \mathcal{K}(P, \ell), & D_a(P) \text{ defined,} \\ Z_a(P) \cdot \text{body } \mathcal{K}(P, \ell), & D_a(P) \text{ undefined.} \end{cases} \quad (3)$$

Lemma 7.1 makes the old body in the mismatch cases literally reusable even though its concrete anchor changes. Equations (2)–(3) use a fixed number of rope primitives and old fields. $\square$

Part III

Turning bounded records into a scaffold

8 The cyclic allocation obstacle

Guiding question. *If every rope operation performs only bounded work, why can we not simply allocate its fresh immutable records in topological order?*

The obstruction is already visible in equation (2) of the closure proof. A fresh palindrome occurrence record P stores the root of its bridge rope $\Omega_b(P)$. The final fresh cell of that rope stores the typed receipt P . Thus

$$P \longrightarrow \Omega_b(P) \longrightarrow \text{terminal receipt } P.$$

Both edges are semantic requirements of the same input transition. Neither endpoint can be replaced by an old record, and neither ordering of the two fresh objects makes all pointers backward.

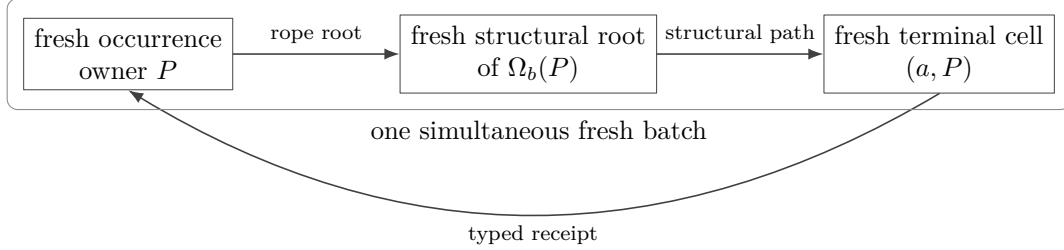


Figure 7: The owner/rope same-step cycle. The steque’s structural records are acyclic when the payload P is treated as an opaque atom. Substitution of the fresh owner pointer creates the cycle only at the batch boundary.

This is not a request for a recursively defined infinite object. After the input letter and bounded old observation are fixed, there are finitely many fresh slots and a finite adjacency table saying which slot each field targets. The graph may contain a cycle, exactly as an ordinary finite directed graph may. Decoding the graph means recovering that adjacency table; it does not mean recursively unfolding the cycle into a tree.

A previously considered source interface allowed a fresh field to name only an old object or an earlier fresh slot. That interface silently excludes Figure 7; proving closure inside it would prove a theorem about the wrong program. The repair is to permit arbitrary references within one uniformly bounded fresh batch and then show that this stronger source step still compiles to the target model.

Three simpler cyclic plans help fix the semantics. A self-loop has one fresh slot x with field $f(x) = x$. A forward reference has two slots ordered syntactically as x, y but field $f(x) = y$. A mutual cycle has $f(x) = y$ and $g(y) = x$. All three are finite labelled adjacency tables. The palindrome batch is larger but no different in kind.

9 Bounded Simultaneous Record Transducers

We now define the source machine and the target scaffold precisely. This section is independent of palindromes: a reader may regard its input as an arbitrary bounded persistent-record update.

9.1 The scaffolding-automaton interface

A scaffolding automaton has finite input, control, and working alphabets, a fixed positive degree d , and a fixed observation distance k . Its initial scaffold is one unlabelled base node. On each arriving letter, the transition sees the depth- k labelled tree unfolding of the old top and appends exactly one new labelled node. Each new port is either missing, follows a port word of length at most k from the old top, or is **SELF**, which names the node being created. Acceptance is determined by the

finite control after the last input letter. The unfolding carries no node identity, so the transition cannot test whether two observed paths alias.

The single-node-per-letter rule is the apparent tension with a persistent program that allocates several logical records. Tuple packing resolves it: the one physical node represents all logical records born at that time, and a finite slot coordinate distinguishes them. The physical graph remains a legal scaffold; “record” is an interpretation placed on its ports and label.

We use only the following direction of the PEG correspondence.

Theorem 9.1 (Loff–Moreira–Reis [2]). *If a finite scaffolding automaton decides a language L , then L^R belongs to ordinary complete-match PEG.*

The existing FPRD Lean development independently reconstructs this sufficient direction, including automata that use SELF. The present paper uses the published theorem; it does not use or claim the converse direction. No result about the full class of scaffold-recognizable languages is needed.

9.2 The corrected source model

Definition 9.2 (Bounded simultaneous record transducer; FPRD-D36). Fix constants s, A, q, r . A bounded simultaneous record transducer has s current roots, allocates at most A active slots per input symbol, gives each record q pointer fields, and follows at most r old-record fields from its current roots in one step. Its control states, record tags, and nonpointer payloads are finite.

After its bounded old observation, one transition chooses a finite labelled batch graph. Every new root and field targets one of:

$\emptyset$, an old address of at most r dereferences, any active slot of the fresh batch.

The last case includes self-loops, forward references, and mutual cycles. The whole finite adjacency table is specified simultaneously. No fresh record is inspected while the plan is chosen.

“Simultaneous” constrains dependencies rather than prescribing a physical allocation primitive. The plan may depend on finite control and bounded old structure, but it may not branch on a fresh record as though that record had already been evaluated. The adjacency table is output data of the transition. The initial roots may be missing; any fixed finite initial object can be absorbed into finite control and materialized in the first batch.

9.3 Tuple packing

Suppose a physical node v_t represents the batch born at time t . A logical pointer is decoded as (v_t, i) , where $i < A$ is a finite slot number. The slot tag is stored in the label associated with the incoming physical port. Old logical field traversal alternates between a physical port and the finite slot tag found at its target. A fresh pointer instead uses SELF to return to v_t and carries the desired fresh slot in the new label.

Theorem 9.3 (Simultaneous batch compiler; FPRD-T126). *Every transducer of Definition 9.2 compiles exactly and letter-synchronously into a finite scaffolding automaton of degree*

$$d = s + Aq$$

and observation/update distance at most $r + 1$.

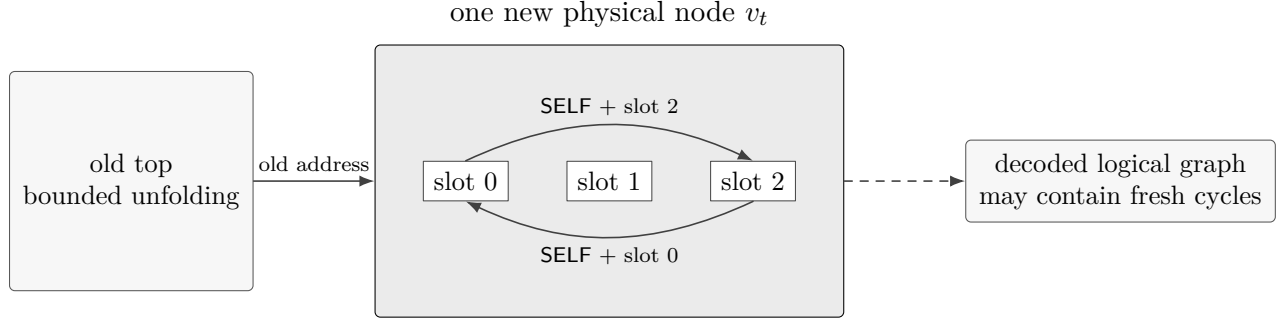


Figure 8: Tuple packing. Several logical fresh records occupy finite slots of one physical scaffold node. Every intra-batch edge has the same physical target **SELF**; its finite tag selects the logical slot.

Proof. One physical scaffold node represents the complete logical batch at one input time. Its finite label records the active mask, every tag and payload, and the finite target-slot tag associated with each present root or field.

Ports $0, \dots, s-1$ represent the next roots. Port $s+iq+j$ represents field $j < q$ of slot $i < A$. Thus a decoded logical pointer is a pair (v, i) of a physical scaffold node and a finite slot number.

Consider an old logical address that starts at root u and follows fields $j_1, \dots, j_m$, with $m \leq r$. The current labelled unfolding reveals the slot reached after every prefix. Compile the address to the raw port word

$$u, s + i_0q + j_1, \dots, s + i_{m-1}q + j_m,$$

where i_h is the slot tag at the preceding logical pointer. This word has length at most $r+1$ and reaches exactly the old target. The extra one in the distance bound is the initial root port; each logical dereference contributes one field port thereafter.

Use a missing descriptor for $\emptyset$ and the compiled word for an old target. For every fresh target use physical **SELF** and record the intended logical slot in the new finite label. This rule makes no distinction between an earlier target, a later target, a self-loop, or one edge of a mutual cycle.

Assume that the old top decodes to the source roots and old heap. Induction on the length of an old logical address proves that its raw word decodes correctly: the base root port reaches the physical node named by that root and supplies its slot; each subsequent field port is selected using the current slot and supplies the next slot. Missing decodes to missing. A fresh descriptor reaches the new physical node by **SELF** and then its target-slot tag selects exactly the requested fresh logical record.

Applying these facts independently to every root and field makes the decoded finite labelled adjacency table identical to the source batch. Notice that the induction is on an *old address*, never on the unfolding of a fresh cycle. Comparing two finite adjacency tables requires no fixed point. Tags, payloads, control, and acceptance are copied directly.

The missing-root base case and this one-step commuting square give the full run invariant by induction on input length. All parameter ranges, masks, labels, and plans are finite, so the compiled automaton is finite. $\square$

For the three sample batches from Section 8, the compiler emits one physical node in every case. A self-loop is **SELF** tagged by its own slot; a forward reference is **SELF** tagged by the later slot; and

the two edges of a mutual cycle are two SELF ports with opposite slot tags. This explicit check is useful because it isolates the sole new semantic feature from all palindrome and steque details.

10 Opaque-atom staging

***Guiding question.** How does a worst-case constant-time functional sequence operation become a bounded fresh-record plan, and where may the cyclic payload be tied in?*

The receipt proof has reduced the online step to a constant number of catenable-sequence primitives. We now justify that these primitives expose a bounded record plan without pretending that an ordinary immutable functional value can tie the cycle in (2).

An ordinary persistent singly linked list provides constant-time head, tail, and insertion at the front, but not catenation or insertion at the rear. A naive catenation must copy one operand or defer work in a thunk; copying is unbounded in the current step, while an unevaluated thunk merely moves that unbounded work to an unspecified later step. Receipt closure needs the following stronger abstract interface.

Receipt-rope need	Sequence operation	Required bound
inspect current receipt	head / constructor test	worst-case constant
promote first cell	tail or pop	worst-case constant
make one bridge cell	singleton / push	worst-case constant
append terminal receipt	inject	worst-case constant
share factored ropes	catenation	worst-case constant

Kaplan and Tarjan’s regular catenable steque supplies these operations in a strict purely functional model. The point of citing their computational model explicitly is not to import implementation folklore: it ensures that one operation really is a bounded decision tree of constructor observations and fresh constructors, rather than a unit-cost instruction hiding an unbounded counter, array, or delayed computation.

Theorem 10.1 (Kaplan–Tarjan [1]). *There is a purely functional regular catenable steque whose push, pop, inject, and catenation operations take worst-case constant time and preserve the representation.*

For their paper, Kaplan and Tarjan define “purely functional” to mean built only from `car`, `cons`, and `cdr`; memoization is explicitly excluded. This strict model is stronger for our purpose than a bare unit-cost RAM claim.

Theorem 10.2 (Opaque-atom staging; FPRD-T127). *Let a strict purely functional operation use finite constructor tests and a worst-case bounded number of `car`, `cons`, and `cdr` operations, without inspecting, comparing, hashing, or identity-testing its stored elements. With new elements replaced by symbolic atoms, its branch and fresh structural allocation graph are determined by old structure alone. The graph is finite, acyclic, and uniformly bounded.*

If a symbolic element denotes a fresh owner record that stores one resulting root, the owner and skeleton form one bounded simultaneous batch. Substituting the owner slot for the atom is sound, even though it creates a cycle.

Proof. Fix the old structural roots. A worst-case constant bound gives a uniform number K of primitive evaluation steps. Replace each new element by a distinct symbol and follow the evaluation.

A constructor test, `car`, or `cdr` reads an old structural record or a structural record allocated at an earlier step. By hypothesis it never follows an element field. Its result and next branch are therefore independent of the symbols. A `cons` allocates one fixed-arity record whose fields are old pointers, earlier fresh structural pointers, or symbolic elements. Induction over the at most K steps yields a bounded acyclic structural allocation graph independent of the symbolic values.

Now add the fresh owner slot and replace the designated symbol by a pointer to that slot. The resulting cycle crosses from the owner to a structural root and returns through an opaque element field. Structural evaluation never follows the return edge, so neither the sequence denotation nor the resource bound changes. Theorem 9.3 compiles the whole simultaneous batch, including that feedback edge, with `SELF`. $\square$

The theorem separates two notions that are easy to conflate. The steque allocates an acyclic *structural skeleton*; this follows from ordinary strict functional evaluation, because a fresh constructor can point only to old or earlier structural constructors. Some fields of that skeleton are *payload positions*. The sequence implementation neither follows nor compares them. Replacing a symbolic payload atom by the simultaneously fresh owner pointer may therefore make the combined semantic graph cyclic without changing any branch or allocation in the steque calculation.

Equivalently, opaque-atom staging is a substitution lemma. Run the structural program with a variable x in each place where the new receipt is needed. The old structure determines a finite graph expression $G(x)$. The owner record contains a root of $G(x)$, and the simultaneous batch substitutes the owner slot for x . No recursive evaluation of G is performed after the substitution. Figure 7 depicts precisely this boundary.

Theorem 10.1 satisfies the hypotheses for the rope operations in Theorem 7.3. Receipt pairs are generic steque elements. A singleton is a push onto empty; body removal is pop; terminal insertion is inject; and rope composition is catenation. Thus every input update determines a bounded simultaneous record plan.

This argument is deliberately semantic rather than source-code specific. The load-bearing premise is Kaplan–Tarjan’s theorem in its stated strict model. Section 13 later examines a modern OCaml realization as a hostile calibration and records two places where a literal program may fail a bounded-record interface even though its high-level operation is called constant time.

11 A direct palindrome scaffold

Guiding question. *Do the word recurrence, receipt closure, catenable sequence, and simultaneous compiler now meet with uniform constants in one machine?*

We assemble the construction. Finite control distinguishes the empty initial state and the three update cases. The persistent roots name the current occurrence record and live headered context rope. Each occurrence record has, for the two input letters, one direct or reset rope root, together with its suffix-link and bridge data. This is a fixed record layout.

On input a , inspect the live context header and the a -entry of the current record. Theorem 4.1 selects pop, internal prepend, or reset. Theorems 5.1 and 7.2 provide only old typed receipts.

Theorem 7.3 expresses the fresh record, its two-entry table, and the new live context by a constant number of catenable-steque operations. Theorems 10.1 and 10.2 turn those operations into one bounded simultaneous record batch. Theorem 9.3 appends that batch as the one scaffold node required for the arriving letter.

All constructors, letters, table coordinates, and slot tags range over finite sets. The input alphabet is binary and the number of rope operations per step is fixed, so the roots, slots, fields, old observation radius, and fresh allocation bound are constants of one machine. Acceptance is the finite empty-body test from Theorem 4.1.

For completeness, one concrete inventory is as follows. Keep two current roots: the occurrence record and the live headered-rope value. A binary occurrence record has one suffix-link field, one finite bridge-letter payload, and two tagged rope entries. A rope value has a header receipt and a steque body root. Expanding (2)–(3) uses, in the worst branch, one source-body pop, one terminal inject, one singleton push, two catenations for the off-bridge table entry, and one pop or catenation for the live context: at most six steque calls. If K bounds the primitive steps and allocations of one Kaplan–Tarjan call, the simultaneous batch therefore needs at most $6K + 4$ fresh slots: $6K$ structural slots, one occurrence owner, two table-rope values, and one live-rope value. Its old observation depth is at most $6K + 3$, and its uniform field bound is the maximum of the fixed occurrence, rope, and steque-record arities. These conservative bounds, rather than their numerical optimization, are all the compiler requires.

The constants deserve one conceptual check. K is not an input-dependent complexity estimate: it is the maximum number of primitive structural steps in one fixed steque operation. The alphabet has two letters and the receipt equations call at most six such operations. Therefore $6K + 4$ is one number built into the automaton, not a counter maintained at runtime. Similarly, $6K + 3$ is a fixed radius of the old scaffold unfolding. The construction may retain an unbounded history, but every next transition observes only this fixed neighborhood and appends exactly one physical node.

The run invariant can now be stated without implementation shorthand. After reading a prefix $S = CP$:

1. the decoded occurrence root names the record for the current occurrence of the longest even-palindromic suffix P ;
2. the decoded live-rope root denotes $\hat{\Pi}(P, C^R)$;
3. every direct/reset table entry in every reachable occurrence record has the word and receipt semantics of Section 6 and Theorem 7.2; and
4. finite control marks acceptance exactly when the live rope has empty body.

The empty initial prefix satisfies the invariant using the distinguished empty record and empty rope. One application of the word recurrence chooses the right abstract update; receipt closure proves the new decoded objects have the required semantics; opaque staging and the compiler prove that the single new physical node decodes to those objects. This is the induction performed in the final theorem.

Theorem 11.1 (Direct palindrome scaffolding; FPRD-T128). *The binary even-palindrome language is recognized by a finite scaffolding automaton:*

$$\boxed{\text{EPAL}_2 \in \text{SA.}}$$

Proof. The construction above gives one letter-synchronous finite scaffolding automaton. Its decoded record state follows the exact recurrence by induction on the input length. It accepts exactly when the decoded left context is empty, which by Theorem 4.1 is exactly EPAL_2 . $\square$

Corollary 11.2 (Direct PEG theorem).

$$\boxed{\text{EPAL}_2 \in \text{PEG.}}$$

This proof route uses neither a strict real-time multitape machine nor Slisenko’s real-time palindrome theorem.

Proof. Apply Theorem 9.1 to Theorem 11.1. Since the reversal of ww^R is again of the form uu^R , the language EPAL_2 is invariant under reversal. $\square$

Part IV

Result, meaning, and evidence

12 What the direct route contributes

The corollary settles no membership question that was still open: the earlier FPRD transfer paper already proved $\text{EPAL}_2 \in \text{PEG}$. The contribution is a new factorization of that fact. It exposes the recognition state as a persistent palindrome algorithm and isolates two interfaces that can be reused or challenged independently.

First, the palindrome side replaces global mutable type transitions with typed occurrence receipts. The longest-suffix recurrence is represented by headered ropes whose shared tails retain a provably invariant receipt shadow. The endpoint, type-birth, anchor-dependence, and bridge-chasing families explain why each extra coordinate is present. This is a more informative outcome than merely knowing that some real-time machine exists: it identifies the exact historical information a direct immutable recognizer carries.

Second, the compiler side shows that a bounded persistent update need not be topologically ordered within one time step. A fixed finite cyclic adjacency table is no harder for a scaffold node to encode than an acyclic one, because all logical fresh records occupy slots of that node and **SELF** supplies the common physical target. Opaque-atom staging gives a disciplined way to use this fact with ordinary strict functional data structures: compute their acyclic skeleton first, and tie any semantic payload cycle only in the final simultaneous batch.

These results motivate questions about other online algorithms, simultaneous-batch normal forms, and receipt replacements for mutable canonical indexes. Here we prove one compiler theorem and one palindrome application; we do not classify PEGs, scaffolding automata, persistent algorithms, or palindromic data structures.

12.1 Comparison of the two proof routes

The transfer route and the direct route should be viewed as complementary explanations.

Coordinate	Earlier transfer route	Direct receipt route
Inherited theorem	Slisenko strict real-time palindrome recognition	Kaplan–Tarjan real-time catenable steques
Intermediate machine	strict real-time multitape Turing machine	bounded simultaneous persistent records
Palindrome state exposed	tape-level state of an external recognizer	longest suffix, occurrence context, typed receipts
Scaffold compilation	general RT-MTTM transfer	tuple-packed finite fresh batch
Final statement	$\text{EPAL}_2 \in \text{PEG}$	$\text{EPAL}_2 \in \text{PEG}$

Agreement at the final statement is useful assurance, but the routes are not independent proofs of every intermediate lemma: both eventually use the Loff–Moreira–Reis scaffold-to-PEG direction. Their internal palindrome arguments and scaffold source models are different. A defect in the direct receipt construction would not retract the earlier membership result; it would invalidate the claimed persistent explanation and compiler application.

12.2 Why the negative representation results matter

The four escape families are small mathematical results with deliberately narrow conclusions. Their value is diagnostic. Each converts an intuitive but underspecified implementation proposal into an exact doctrine and then exhibits an unbounded parameter that doctrine fails to expose. The residue of the failure becomes the next positive invariant: endpoint escape motivates typed pointers; future type transitions motivate occurrence receipts; anchor dependence motivates anchored ropes; bridge escape motivates recursively materialized direct ropes.

This failure-to-invariant progression is also where the FPRD method affected the proof most visibly. Instead of treating a convenient representation as the recognition dynamic, the investigation repeatedly projected the same word semantics into competing persistent coordinates and asked which observations remain bounded. The final coordinate system is not asserted to be canonical. It is the first one in this chain for which every semantic update, persistent operation, and scaffold address closes with an explicit uniform bound.

13 Modern source calibration

Viennot, Wendling, Gueneau, and Pottier give a later OCaml implementation and a separate Rocq implementation of Kaplan–Tarjan catenable dequeues [3]. Their paper clearly separates the evidence: the OCaml code is tested but not verified; the Rocq code has machine-checked functional correctness; there is no formal correspondence between the two; and the Rocq extraction does not establish the same constant-time bound because runtime indices are retained.

We therefore use Kaplan–Tarjan Theorem 10.1 as the mathematical premise. The modern source is a hostile implementation calibration. At the pinned commit, its operational core is strict and nonrecursive, but the audit found two exact mismatches with an opaque finite-record interface:

1. four classifiers read an exact unbounded buffer length; and
2. the noncatenable empty test uses OCaml polymorphic structural equality.

The first is extensionally replaced by fixed persistent end probes; no classifier uses more than twelve primitive root observations. The second is replaced by matching the outer empty constructor. On

valid inputs the original equality already stops at that constructor, before any element is inspected, but the replacement makes opacity explicit. These repairs are checked through length 512. This evidence is not described as verification of the OCaml program and is not needed for Theorem 11.1.

This separation matters because “functionally verified” and “satisfies the bounded-record cost interface” are different coordinates. The Rocq implementation offers strong evidence about the denotation of its own program; the paper explicitly does not establish the same extraction-time cost bound. The strict OCaml implementation offers a concrete constant-time design and an excellent source for hostile interface inspection; its authors explicitly do not claim a formal correspondence with the Rocq program. Neither coordinate alone is silently promoted into the conjunction needed here. The mathematical proof instead uses the original Kaplan–Tarjan theorem and the staging lemma.

14 Executable evidence and limits

The public package contains five independent Python programs. They check the three-case context recurrence, typed mirror receipts, receipt-rope closure, simultaneous cyclic batch decoding, and the modern catenable-interface calibration. The current complete run includes:

Finite audit	Cases or checks
Occurrence-context word updates	262,142 exhaustive; 100,000 random
Typed mirror updates	131,070 exhaustive; 20,000 random
Legal receipt-tail updates	65,534 exhaustive; 20,000 longer random
Typed old-address checks	5,119,788
Simultaneous compiler squares	250,000 plus three cyclic witnesses
Catenable threshold classifiers	3,063; maximum 12 observations

The programs use exact finite words and pointer graphs. They corroborate the displayed identities and protect against regression, but bounded computation is not the proof of a universal theorem.

The checks are intentionally divided at the same seams as the proof. A word checker has no implementation of scaffolds; it can therefore falsify an error in the recurrence without agreeing by construction with the compiler. The compiler checker generates bounded record heaps and cyclic batches without palindrome semantics. The source-interface checker inspects the pinned catenable implementation and its threshold replacements without assuming the receipt equations. This separation is not complete logical independence, but it reduces the chance that one shared encoding mistake makes every check agree.

Three boundaries should remain visible.

1. The conclusion $\text{EPAL}_2 \in \text{PEG}$ was already proved by the earlier FPRD real-time transfer route. This paper contributes a direct persistent record route.
2. Endpoint-only, type-birth-only, and bounded bridge-chasing failures are representation-specific. They are not lower bounds against all scaffolding automata.
3. External specialist review and a broader novelty search remain open. A complete written proof is not the same coordinate as novelty or review status, and no novelty claim is made here.

The reusable result beyond palindromes is Theorem 9.3: a scaffold node can execute one bounded

simultaneous persistent record step, including finite semantic cycles. The direct palindrome construction is the first application studied here, not a claimed classification of all bounded persistent programs or all PEG languages.

A Notation and a closer reading of the running trace

Table 3 collects the symbols whose meanings cross more than one proof layer. A symbol naming a word is not silently identified with the record that represents an occurrence of that word.

Symbol	Meaning
$S = CP$	input prefix, factored at its longest even-palindromic suffix P
$\ell = C^R$	reversed occurrence context; its head is immediately left of P
$\lambda(P)$	longest proper even-palindromic suffix of nonempty P
$D_a(P)$	longest proper even suffix of P immediately preceded by a
$\omega_a(P)$	reversed word displaced into context when $D_a(P)$ is wrapped
$\Pi(A, w)$	anchored palindrome path obtained by wrapping A through w
$\hat{\Pi}(A, w)$	headered receipt rope for that anchored path
$\Omega_a(P)$	direct receipt rope when $D_a(P)$ exists
$Z_a(P)$	reset receipt rope when $D_a(P)$ does not exist
SELF	physical scaffold descriptor for the node currently being appended
(v, i)	decoded logical record: physical node v and finite slot i

Table 3: Notation crossing the word, record, and scaffold layers.

The two internal-direct rows of Table 1 show both a base case and a nontrivial jump. At row 3 the old palindrome is $P = 00$ and the context is empty. For the arriving 0, the only eligible proper even suffix is $D_0(P) = \epsilon$. Writing $P = Z0D_0(P)$ gives $Z = 0$ and $\omega_0(P) = 0$. Hence

$$P' = 0\epsilon 0 = 00, \quad \ell' = 0\epsilon = 0.$$

The palindrome type has not changed, but its occurrence has: it now occupies the last two positions of 000 rather than all of the preceding prefix. This is a small example of why occurrence context cannot be discarded.

At row 8 the old state is

$$S = 0001100 = 0 \cdot 001100, \quad P = 001100, \quad \ell = 0.$$

The arriving letter is 1, so the external case fails. The factorization

$$P = 001 \cdot 1 \cdot 00$$

shows $D_1(P) = 00$ and $\omega_1(P) = 100$. The recurrence gives

$$P' = 1 \cdot 00 \cdot 1 = 1001, \quad \ell' = 100 \cdot 0 = 1000.$$

The extra 1 in $Z = 001$ is easy to omit when the factorization is read by eye; reversal then makes that omission visible at the front of the new context. The checked table fixes the intended indexing, while the theorem supplies the general argument. The example is therefore useful both pedagogically and as a regression witness.

B Fixed allocation inventory

The proof needs uniform finiteness, not optimized constants. Nevertheless an explicit inventory prevents the phrase “constant number of operations” from hiding an input-dependent loop.

Let K simultaneously bound the primitive observations and fresh structural constructors performed by any one specialized Kaplan–Tarjan operation used by the algorithm. Such a maximum exists because the relevant operations have a finite strict decision tree and the element payloads are opaque. In the worst binary branch, the receipt update invokes at most six sequence operations:

1. one pop to expose the body of the source direct or reset rope;
2. one inject to add the terminal receipt carrying the fresh owner;
3. one singleton construction for the new bridge cell;
4. two catenations for an off-bridge direct or reset table entry; and
5. one pop or catenation for the new live context.

Charging K fresh structural slots to each call gives $6K$ slots. Four additional logical values suffice at the receipt layer: the fresh occurrence owner, two finite-alphabet table-rope values, and the live-rope value. Thus $6K + 4$ is a conservative simultaneous allocation bound. Sharing may make the actual number smaller, and an optimized layout may combine some values, but no such optimization is used by the proof.

Old observation is bounded similarly. Inlining at most six operation trees and the fixed occurrence/rope selectors gives the conservative radius $6K + 3$. Every record constructor has fixed arity; let q be the maximum after padding smaller constructors with missing fields. With two current roots and $A = 6K + 4$ slots, Theorem 9.3 yields physical degree

$$d = 2 + (6K + 4)q$$

and a fixed update distance determined by the inlined radius. These are machine constants. They do not grow with the prefix length, the current palindrome, the receipt-rope length, or the number of historical occurrences.

C Claim and reproducibility map

The publication package separates mathematical statements from the mechanisms used to challenge them. FPRD-D36 is Definition 9.2. FPRD-T125 is the receipt package comprising Theorems 4.1, 5.1, 7.2, and 7.3, with the bridge-shadow lemma. FPRD-T126 is Theorem 9.3. FPRD-T127 is Theorem 10.2. FPRD-T128 is Theorem 11.1 and Corollary 11.2.

The five public checkers correspond respectively to occurrence-context words, mirror receipts, receipt closure and shadow reuse, simultaneous record compilation, and the catenable-sequence interface. Their canonical output is stored with the paper so that a rerun can be compared byte for byte. The package verifier rebuilds the PDF, checks its reader-facing anchors and size guardrails, reruns the focused mathematical audits, and checks the central claim/publication registries.

Primary papers and external source checkouts are intentionally not copied into the public repository. The source audit records titles, versions, links, local checksums, and exactly which theorem or source property is used. This keeps the public artifact limited to original work while making the inherited premises inspectable. A successful checker run is evidence about the tested finite instances or pinned

source; it does not change the mathematical status of a theorem, certify novelty, or substitute for specialist review.

Acknowledgment of inherited work

Kaplan–Tarjan catenable dequeues and the Loff–Moreira–Reis scaffold/PEG correspondence are inherited mathematical results. The Viennot et al. implementations are external work used for calibration. FPRD Lab contributes the occurrence-context formulation, typed receipt system, representation escape audits, simultaneous cyclic source model, tuple-packed compiler, opaque-atom staging bridge, direct composition, and the adversarial correction that exposed the same-step cycle before publication. Detailed attribution is recorded in the accompanying `FPRD-Lab-Contribution.md` and `SOURCE-AUDIT.md`.

References

- [1] Haim Kaplan and Robert E. Tarjan. Purely functional, real-time dequeues with catenation. *Journal of the ACM*, 46(5):577–603, 1999. <https://doi.org/10.1145/324133.324139>.
- [2] Bruno Loff, Nelma Moreira, and Rogerio Reis. The computational power of parsing expression grammars. 2020. <https://arxiv.org/abs/1902.08272v2>.
- [3] Jules Viennot, Arthur Wendling, Armael Gueneau, and Francois Pottier. Verified purely functional catenable real-time dequeues. 2025. <https://arxiv.org/abs/2505.07681v3>.