

Persistent Receipts and Simultaneous Records

Draft 4 applicability correction

10 September 2026

Status	The direct palindrome route is a candidate with an open load-bearing data-structure bridge. It is not presently claimed as a closed theorem.
Still proved	The simultaneous-record compiler, including the virtual-initial-record repair, and the conditional opaque-atom staging lemma.
Withdrawn	The claim that Kaplan-Tarjan Theorem 5.1 alone closes the finite-record/opacity bridge, and the formulas $A = 6K + 4$ and $r = 6K + 8$ with their derived d, k .
Unaffected	The separate persistent-stack proof that binary even palindromes belong to PEG.

What the source establishes

Kaplan and Tarjan's Theorem 5.1 states worst-case constant cost for push, pop, inject and catenation on regular steques, with regular results. Catenation assumes two regular inputs. Their paper defines a strict purely functional model using car, cons and cdr, with memoization excluded.

The described steque algorithm appears to move elements of the arbitrary base set without examining them. But the theorem does not state an opacity theorem, present code, define a formal operational cost semantics, or give a uniform bound on cons allocations or old-address depth. Those are separate requirements of the FPRD application.

Why the current proof does not bridge the gap

The conditional staging lemma begins with an explicit program that has finite constructor tests, bounded car/cons/cdr work, and opaque base elements. Kaplan-Tarjan supply a high-level prose construction and an asymptotic theorem. The manuscript had treated the stronger interface as if it were a clause of the cited theorem. That inference is now withdrawn.

A later paper gives nonrecursive OCaml code and a separate Rocq verification. The OCaml code is tested rather than verified; the Rocq development does not establish the same constant-time extraction; and the paper states that there is no formal connection between them. Their conjunction cannot be assumed.

The allocation inventory

The six-call list correctly describes the constructive rope updates in the worst binary branch: source pop, terminal inject, singleton push, two catenations for the off-bridge table entry, and one live pop or catenation.

It is not a complete cost inventory. The program must first determine whether the live body is empty and, when it is not, inspect its first cell. In the external branch this may share work with the live pop. In internal and reset branches the sixth counted operation is a catenation, so the front/empty observation has not been charged or proved allocation-free. The prose radius count likewise omits a literal path account for these observations and returned-cell fields.

Safe conditional conclusion

If a concrete strict sequence implementation satisfies the staging interface, then the fixed binary client program has some finite allocation bound A^* and old-address radius R^* . With maximum record arity q , the corrected compiler would give

$$d = \max\{1, 2 + A^* q\}, k = R^* + 1.$$

No numerical A^* or R^* is established here. Restoring explicit constants requires a branch-by-branch program and address inventory, including empty/front observations, every returned-cell field, every fresh structural record, and the inline-storage convention for fixed-shape cells.

What would close the bridge

One path is to reconstruct the Section 5 steque program and prove sequence correctness, regularity preservation, base-element opacity, finite branching, allocation and old-address bounds. Another is to prove that one later concrete implementation satisfies that complete conjunction. Finite tests and unconnected proofs about different implementations are not enough.

Evidence boundary

Two separate AI-assisted reviews addressed different failure modes: exact source scope and client allocation/address accounting. They share model dependencies and are not external specialist review. This is not a full audit of the receipt recurrence or the complete palindrome construction.

Primary source: Haim Kaplan and Robert E. Tarjan, Purely Functional, Real-Time Deques with Catenation, JACM 46(5), 577-603 (1999), DOI 10.1145/324133.324139; author-hosted final PDF inspected through Theorem 5.1 and its supporting steque construction.

Comparison: Jules Vienne, Arthur Wendling, Armael Gueneau and Francois Pottier, Verified Purely Functional Catenable Real-Time Deques, arXiv:2505.07681v3 (2026), especially Sections 1.1-1.3.

Persistent Receipts and Simultaneous Records

A Direct Scaffold for Even Palindromes

Draft 3 correction folio - 10 September 2026

What changed	The fixed initial record graph is represented by a permanent finite decoder table, rather than being charged to the first fresh batch.
Result	The original degree bound $d = \max\{1, s + A q\}$ and distance $k = r + 1$ are preserved. Finite labels and control may depend on the fixed initial graph.
Scope	This repairs the initialization case of Theorem 10.2. It does not recheck the receipt recurrence, Kaplan-Tarjan premise, allocation inventory, or the complete palindrome theorem.

How to read this PDF

This folio is part of the manuscript. It precedes the preserved Draft 2 body and its contribution afterword. Where the body says that a fixed initial object is materialized in the first batch, or that the empty record is created in that batch, this folio supersedes that wording. The corrected web edition incorporates the replacement argument at the theorem and application sites.

Corrected theorem

Theorem 10.2 (corrected initialization case). Consider a bounded simultaneous record transducer with finite control and record alphabet, s roots, q immutable pointer fields per record, at most A fresh records per letter, and rooted observation depth at most r . Suppose its initial heap is fixed, finite, immutable and pointer-closed; transitions use only bounded labelled rooted unfoldings and do not test pointer identity; and acceptance is in finite control. Then it has a letter-synchronous scaffolding-automaton simulation with

$$d = \max\{1, s + A q\}, \quad k = r + 1,$$

preserving decoded source configurations up to record renaming and preserving acceptance after every prefix, including the empty prefix.

The virtual-initial-record encoding

Name the fixed initial records $v_0, \dots, v_{(H-1)}$. Their labels and fields form a finite table in the compiled machine. The decoder includes them as distinct logical records at every time; they never occupy physical scaffold nodes.

Logical pointer	Label tag	Physical edge
Missing	MISSING	missing
Initial record v_h	VIRTUAL(h)	missing
Later slot i at node u	DYNAMIC(i)	edge to u

Address lemma. A rooted address containing $m \leq r$ field dereferences is determined by the physical $(r+1)$ -neighbourhood. Its endpoint is missing, a virtual identifier, or a dynamic slot reached by a physical path of length at most $m+1$.

For a dynamic root, follow its root port and read its slot tag. At a dynamic record, the slot chooses the field port and that node label supplies the next tag. Each dynamic continuation adds one physical edge. A virtual continuation switches to the fixed table. It cannot return to dynamic storage because the initial graph is immutable and pointer-closed. Induction on address length proves the lemma. A wholly dynamic walk uses one root edge and at most r field edges.

One-step simulation

Recover the permitted old observations with the address lemma and apply the source transition table. Encode missing targets with MISSING, virtual old targets with VIRTUAL(h), dynamic old targets with their bounded path and DYNAMIC(i), and fresh targets with SELF and the fresh slot. Only the original at-most-A fresh records occupy the new node. Fresh self-loops, forward references and mutual cycles are finite adjacency entries.

Initialization and induction

LMR starts from an unlabelled physical node, so use a separate initialization flag and a fixed initial-root table. At time zero the decoder returns the source initial configuration and copies its empty-input acceptance. The first input transition reads the fixed table directly, emits only its new dynamic batch, and clears the flag permanently. Later steps use the address lemma. The decoder adjoins the virtual graph to every dynamic heap, including unreachable initial records when whole-heap preservation is claimed. Induction on input length gives the run invariant.

Boundary cases and assumptions

Zero degree

If $s + A q = 0$, add one unused physical port to meet the target model's positive-degree convention.

No allocation

If $A = 0$, appended nodes have no active dynamic slots; virtual roots still work.

No fields

If $q = 0$, active slots and finite record labels still distinguish records.

No roots

If $s = 0$, no rooted old observation exists; globally decoded fresh batches remain possible.

Empty initialization

If $H = 0$, this is the ordinary tuple-packing construction.

Cycles

Cycles wholly inside the initial graph use its fixed table. Fresh cycles use slot-tagged SELF. A dynamic record may point to a virtual record; a virtual record cannot point to later dynamic storage.

Identity

The construction reconstructs allowed labelled unfoldings. It does not supply a new identity or alias oracle. The source restriction against identity-dependent branching remains essential.

Small cyclic example

Choose $r = 2$. Let $u.f = v$, $v.f = u$, and $\text{root} = u$ in a two-record fixed initial graph. On the first letter create w with $w.f = v$ and make w the root. The new root is SELF tagged DYNAMIC(0); $w.f$ is a missing physical edge tagged VIRTUAL(v). The logical address $\text{root}.f.f$ reaches u through one physical edge and finite-table traversal. On the next letter, a fresh z can point back to dynamic w by a bounded old path. The initial cycle requires no fresh slots.

Application to the direct palindrome construction

Section 12 uses two roots. Its fixed logical initial graph contains the distinguished empty occurrence, both reset ropes and their finite singleton representations, and the initial live empty rope, with sharing preserved. These are virtual from time zero. They are not "created in the first batch."

The steady-state inventory $A = 6K + 4$ counts dynamic records allocated by each input step. Subject to the separate allocation inventory and other proof premises, Theorem 10.2 therefore retains

$$d = 2 + (6K + 4)q, \quad k = 6K + 9.$$

The empty-prefix invariant is decoded from the initialization state. Empty-input acceptance is copied from source finite control. Theorem 11.2 is a one-step staging statement and needs no added initial allocation. Theorem 12.2 and its PEG corollary inherit the corrected compiler case but are not otherwise revalidated here.

Evidence boundary

The proof was reconstructed from Draft 2, pages 26-34 and 37-38, and checked against Loff-Moreira-Reis arXiv:1902.08272v2, Definitions 10-15. Two separate AI review passes attacked the encoding and its application. They share model dependencies and are not external expert review. No proof-assistant artifact or compiler checker was run for this correction.

The earlier A+H materialization is still a valid fallback construction. It is unnecessary under the virtual encoding, but the existence of this encoding does not show that the original missing-root proof handled the case.

This correction does not establish the Kaplan-Tarjan implementation bridge, the receipt recurrence, the full allocation inventory, or the complete palindrome theorem. Those obligations retain their existing status.

Primary target source: Bruno Loff, Nelma Moreira and Rogerio Reis, The computational power of Parsing Expression Grammars, arXiv:1902.08272v2, Definitions 10-15; journal version JCSS 111 (2020), 1-21.

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

Joshua Gay
FPRD Lab, Insight Forge

Draft 2, 22 August 2026

Abstract

A parsing expression grammar (PEG) is a grammar read as a deterministic recognition procedure: alternatives are tried in a fixed order and the first success is committed to. Loff, Moreira, and Reis proved that a language has a PEG exactly when its *reversal* is decided by a *scaffolding automaton*, a machine that reads its input once, left to right, and appends one bounded-degree node to a permanent graph at every letter. They conjectured that the language of even-length binary palindromes $\text{EPAL}_2 = \{ww^R : w \in \{0,1\}^*\}$ has no PEG.

That conjecture is false, and this paper gives a direct proof: we build a finite scaffolding automaton that recognizes EPAL_2 and then apply the Loff–Moreira–Reis correspondence. The automaton maintains, for the input read so far, its longest even-palindromic suffix and the reversed context to the left of that suffix. Because a scaffold can never modify an old node, the usual mutable palindrome index (a “palindromic tree”) is unavailable; we replace its pointers by *typed receipts*—immutable pointers to the particular historical record a later step will need—and transport them in *persistent receipt ropes* built from Kaplan and Tarjan’s catenable steques. The construction meets one genuine obstacle: a fresh palindrome record must own a fresh rope whose last cell points back to that record, a cycle no ordinary allocate-then-link discipline can build. We define *bounded simultaneous record transducers*, whose one-step update is an arbitrary finite graph of fresh records, and prove that every such transducer compiles into a scaffolding automaton, using the scaffold’s self-reference port together with finite slot tags. A staging theorem explains why the steque’s acyclic allocation can be tied to the fresh owner only at that boundary.

The membership conclusion is not new: an earlier paper from the same project proves it by compiling a classical real-time Turing-machine recognizer. What is new here is the direct persistent representation and the compiler theorem, which does not depend on palindromes. The paper is written to be checkable from its own text plus the cited literature; it defines grammars, PEGs, scaffolding automata, persistence, and every structure it uses. It is released for specialist review and makes no novelty claim.

Contents

I	Why this proof looks different	4
1	The question and the direct route	4
1.1	The language and the conjecture	4

1.2	What this paper does	4
1.3	How to read this paper	5
2	Background: words, grammars, parsing expression grammars, and scaffolds	5
2.1	Words, languages, palindromes	5
2.2	Context-free grammars: the generative reading	6
2.3	Parsing expression grammars: the recognition reading	6
2.4	Why comparing the two classes is not a syntactic matter	7
2.5	Scaffolding automata	7
2.6	The exact correspondence with PEGs, and why reversal appears	8
2.7	Persistent data structures	9
2.8	The mutable palindrome index, and why a scaffold cannot have it	9
3	A reader's map	10
3.1	The viewpoint behind the paper	10
3.2	Three representations of one state	10
4	A running palindrome trace	11
II	Persistent palindrome state	12
5	The occurrence-context recurrence	12
5.1	Definitions	12
5.2	The three-case update	13
5.3	The transition table has fixed width	14
6	Typed mirror receipts	15
7	Why simpler representations fail	17
7.1	Why an immutable type table is not enough	17
7.2	Why raw persistent words are not enough	17
7.3	Why bounded bridge chasing is not enough	18
8	Receipt ropes and shadow transport	18
8.1	What a receipt rope represents	19
8.2	Definitions	19
8.3	The bridge-shadow identity	20
8.4	Direct and reset ropes, and the mirror-prefix lemma	21

8.5	The receipt-rope recurrence	22
8.6	Full receipt closure	23
III	Turning bounded records into a scaffold	25
9	The cyclic allocation obstacle	25
9.1	Records, fields, and batches	25
9.2	The cycle the palindrome construction forces	26
10	Bounded simultaneous record transducers	26
10.1	The source model	26
10.2	Tuple packing	27
11	Opaque-atom staging	29
11.1	Why an ordinary list is not enough	29
11.2	The inherited theorem and its model	29
12	A direct palindrome scaffold	31
12.1	The record layout	31
12.2	One step	31
12.3	Uniform constants	32
12.4	The run invariant and the theorem	33
IV	Result, meaning, and evidence	34
13	What the direct route contributes	34
13.1	Comparison of the two proof routes	34
13.2	Why the negative representation results matter	35
14	Assurance, provenance, and the viewpoint	35
14.1	What kind of evidence supports what	35
14.2	Provenance and AI assistance	36
14.3	The viewpoint, stated more fully	36
A	Notation	37
B	Fixed allocation inventory	37

C Claim and reproducibility map	38
--	-----------

D Modern source calibration	38
------------------------------------	-----------

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?*

1.1 The language and the conjecture

Write w^R for the word w read backwards and EPAL_2 for the set of binary words of the form ww^R : the even-length palindromes ϵ , 00 , 11 , 0110 , 1001 , 001100 , and so on. As a *generative* object the language is as simple as they come: three rewriting rules produce it (Section 2.2). Loff, Moreira, and Reis (hereafter LMR) studied how much computational power the *recognition-based* grammars called parsing expression grammars have, and found both surprising positive examples and a natural negative conjecture [7, Conjecture 7, p. 10]:

The language of even-length palindromes has no PEG, i.e. $P = \{ww^r \mid w \in \{0,1\}^\} \notin \text{PEG}$.*

The intuition behind the conjecture is that a PEG cannot “find the middle” of its input. The conjecture is false. An earlier paper of this project [5] refuted it by compiling strict real-time multitape Turing machines into LMR’s scaffolding automata and invoking the classical theorem that palindromes are recognizable in real time by such a machine [9, 4]. That route proves membership but explains nothing about *what a scaffold must remember* to compare two halves.

1.2 What this paper does

This paper gives a second, direct proof. The reader needs no Turing machine and no palindrome theorem: only words, finite graphs, and one inherited data structure. Figure 1 shows the two routes.

The lower route produces two separable results.

- **The application.** A finite scaffolding automaton recognizes EPAL_2 (Theorem 12.2); by LMR’s theorem and the fact that EPAL_2 is its own reversal, $\text{EPAL}_2 \in \text{PEG}$ (Corollary 12.3).
- **The reusable tool.** Any one-step update that reads a bounded amount of old persistent structure and creates a bounded finite graph of fresh records—even a graph with cycles among the fresh records—can be performed by one scaffold node (Theorem 10.2). The palindrome construction is exactly what forces the cyclic case.

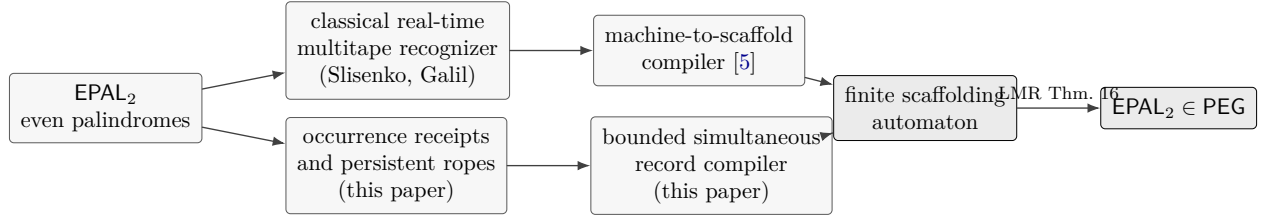


Figure 1: Two proofs of the same membership statement. The upper route is the earlier transfer proof. This paper develops the lower route. Both end with the same inherited theorem of Loff, Moreira, and Reis.

1.3 How to read this paper

Section 2 defines everything the argument uses: words, context-free grammars, PEGs, scaffolding automata, persistence, and the mutable palindrome index that the scaffold cannot have. A reader who knows these objects may skim it, but should read Definition 2.6 and Theorem 2.7, whose exact form matters. Section 3 fixes three layers of description that the proof keeps separate. Section 4 follows one input word by hand. Part II proves the combinatorics of the longest even-palindromic suffix and turns it into operations on persistent ropes. Part III defines the compiler, proves it, and assembles the automaton. Part IV says what was proved, what is inherited, and what remains open.

Proofs state their strategy before their details. Every example says which of its features generalize. Theorem statements are complete in this paper; the only external premises are LMR’s Theorem 16 (sufficient direction) and Kaplan–Tarjan’s Theorem 5.1, both quoted with their hypotheses.

2 Background: words, grammars, parsing expression grammars, and scaffolds

2.1 Words, languages, palindromes

An *alphabet* Σ is a finite set of letters; in this paper $\Sigma = \{0, 1\}$ unless stated otherwise. A *word* is a finite sequence of letters, ϵ is the empty word, Σ^* is the set of all words, and $|w|$ is the length of w . Juxtaposition uv is concatenation. A *language* is any subset of Σ^* . The *reversal* of $w = c_1c_2 \cdots c_n$ is $w^R = c_n \cdots c_2c_1$; reversal of a language is $L^R = \{w^R : w \in L\}$.

A word u is a *suffix* of w if $w = xu$ for some x , and a *proper* suffix if moreover $u \neq w$; prefixes are defined symmetrically. The empty word is a proper suffix of every nonempty word. A *palindrome* is a word w with $w = w^R$; an *even palindrome* is a palindrome of even length, equivalently a word of the form ww^R . The empty word is an even palindrome. Thus

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

and $\text{EPAL}_2^R = \text{EPAL}_2$ because reversing ww^R gives ww^R again.

When a palindrome P occurs as a suffix of a word $S = CP$, the letter of C immediately to the left of that occurrence will matter constantly; we say the occurrence is *preceded by* that letter. The empty suffix of a nonempty word is preceded by the word’s last letter; the empty suffix of ϵ is preceded by no letter.

2.2 Context-free grammars: the generative reading

A *context-free grammar* consists of a finite set of terminal symbols (our alphabet), a disjoint finite set of non-terminal symbols, a start non-terminal, and finitely many *rewriting rules* $A \rightarrow \varphi$ in which A is one non-terminal and φ is a word over terminals and non-terminals [1, pp. 120–122]. A *derivation* starts from the start symbol and repeatedly replaces one occurrence of a non-terminal by the right-hand side of one of its rules, in any order, until only terminals remain. The language *generated* by the grammar is the set of terminal words obtained by some derivation.

Example 2.1 (even palindromes are context-free). With start symbol S and rules

$$S \rightarrow 0S0, \quad S \rightarrow 1S1, \quad S \rightarrow \epsilon,$$

the derivation $S \Rightarrow 0S0 \Rightarrow 01S10 \Rightarrow 0110$ produces 0110, and in general the generated language is exactly EPAL_2 : each rule adds one matching letter at each end, and every ww^R arises by choosing the letters of w in order. The reading is *existential*: a word belongs to the language if *some* sequence of choices produces it. Nothing in the definition says how a machine would discover those choices.

2.3 Parsing expression grammars: the recognition reading

Parsing expression grammars, introduced by Ford [3, §3], use rule shapes that look like context-free rules but give them an operational meaning. We follow the formalization of LMR [7, Definitions 1–4], which is the one their theorem is stated for.

Definition 2.2 (parsing expressions and grammars). Fix disjoint finite sets Σ of terminals and NT of non-terminals. *Parsing expressions* are built from the atoms $a \in \Sigma$, $A \in NT$, ε (accept the empty word) and **FAIL** by three constructions: *sequence* e_1e_2 , *ordered choice* e_1/e_2 , and the *predicates* $!e$ and $\&e$. A *parsing expression grammar* G assigns to each non-terminal A one expression $R(A)$, written $A \leftarrow R(A)$, and names a start non-terminal S .

The meaning of an expression is a *recognition procedure* $\text{Rec}_G(e, x)$ that reads an input word x from the left and either *fails* or *succeeds consuming a prefix* x' of x [7, Definition 3 and Algorithm 1]:

- ε succeeds consuming nothing; **FAIL** fails; the terminal a succeeds consuming a if x begins with a , and fails otherwise.
- Sequence e_1e_2 : run e_1 on x ; if it succeeds consuming y_1 , run e_2 on the remainder; succeed consuming y_1y_2 if both succeed, otherwise fail.
- Ordered choice e_1/e_2 : run e_1 ; *if it succeeds, that is the answer*; only if it fails is e_2 run on the same input.
- Predicates: $!e$ succeeds consuming nothing if e fails, and fails if e succeeds; $\&e$ succeeds consuming nothing if e succeeds.
- A non-terminal A runs $R(A)$.

The procedure is deterministic: at every point exactly one thing happens. It may loop forever (for example $A \leftarrow A$); a grammar is *total* if it never does. Following LMR, a total grammar *recognizes* the language

$$L(G) = \{x \in \Sigma^* : \text{Rec}_G(S, x) = x\},$$

the words on which the start expression succeeds *consuming the whole input*, and PEG is the class of languages recognized by total PEGs [7, Definition 4]. We call this *complete-match* recognition; it is the ordinary notion, not a variant.

Example 2.3 (order is semantic). Take $S \leftarrow a/ab$ and the input ab . The first alternative a succeeds consuming a , so the choice commits to it: $\text{Rec}(S, ab) = a \neq ab$, and $ab \notin L(G)$ although the second alternative would have consumed it. As a context-free grammar, $S \rightarrow a \mid ab$ generates $\{a, ab\}$. Ordered choice is not a parsing strategy for a context-free grammar; it changes the language. (Ford proves the corresponding algebraic fact: sequence distributes over ordered choice on one side only [3, p. 7].)

Example 2.4 (predicates see ahead without consuming). The expression $!(0/1)$ succeeds exactly at the end of the input, consuming nothing, because it succeeds only where no letter can be read. Predicates let a PEG test a property of the remaining input and then parse it again. It is known that the non-context-free language $\{a^n b^n c^n\}$ has a PEG [7, p. 3, citing Aho and Ullman], and LMR use predicates to recognize the palindromes of power-of-two length [7, Theorem 8, p. 10].

2.4 Why comparing the two classes is not a syntactic matter

Example 2.1 shows EPAL_2 is context-free by three rules. One cannot read those rules as a PEG and hope: the choice $S \leftarrow 0S0/1S1/\varepsilon$ commits to $0S0$ whenever the input begins with 0, and then must find the matching closing 0 at a position it has no way to locate—a PEG consumes left to right and cannot “count to the middle”. That is the content of LMR’s conjecture. Conversely, PEGs recognize languages, such as $\{a^n b^n c^n\}$ [7, p. 3], that no context-free grammar generates. Neither class contains the other by any syntactic argument; the comparison must go through a machine model. LMR supplied that model.

2.5 Scaffolding automata

LMR’s machine builds a permanent graph. We give their definition [7, Definitions 10–15, pp. 14–16] in words first, then precisely, then with a picture.

In words: the machine reads one input letter at a time. Its memory is a growing directed graph whose nodes are numbered $0, 1, 2, \dots$ in the order of creation; node 0 is an unlabelled base, and every later node carries a label from a finite *working alphabet* and a fixed number d of *ports* (outgoing edges), each pointing to an older node, to the node itself, or to nothing. Old nodes are never changed. At each letter the machine looks only at the nodes within a fixed distance k of the most recently created node (the *top*), following ports; based on what it sees, on the letter, and on its finite control state, it creates one new node, chooses its label, and chooses where its ports point—necessarily among the nodes it just looked at, or the new node itself. After the last letter it accepts or rejects by its control state.

Definition 2.5 (scaffold). Let $d \geq 1$ and let Γ be a finite alphabet. A (d, Γ) -scaffold with top t has nodes $0, 1, \dots, t$, a label $L(v) \in \Gamma \cup \{\emptyset\}$ for each node ($L(0) = \emptyset$: the base is unlabelled), and for each node v and each port $i \in \{0, \dots, d-1\}$ a target $e_v(i) \in \{0, \dots, v\} \cup \{\emptyset\}$ (*edges point backwards or to the node itself*; $\emptyset$ means the port is missing). A *port word* $p = p_1 \dots p_m$ over $\{0, \dots, d-1\}$ is followed from a node by taking port p_1 , then port p_2 of the node reached, and so on; it is *valid* from v if no missing port is met, and then it has an *endpoint*. The *k-neighbourhood* $N_k(v)$ of a node is the labelled tree of depth k obtained by unfolding ports from v : it records the label of every node reached by a port word of length at most k , and where a port is missing. The tree records labels, not node numbers, so it does not reveal whether two port words reach the same node.

Definition 2.6 (scaffolding automaton). A scaffolding automaton is a tuple $A = \langle \Sigma, d, \Gamma, k, Q, \delta, q_0, F \rangle$: an input alphabet Σ , a degree $d \geq 1$, a working alphabet Γ , a distance $k \geq 0$, a finite set of control states Q with initial state q_0 and accepting states $F \subseteq Q$, and a transition function

$$\delta : Q \times \Sigma \times N_k(d, \Gamma) \longrightarrow Q \times \Gamma \times (\{0, \dots, d-1\}^{\leq k} \cup \{\text{SELF}, \emptyset\})^d,$$

where $N_k(d, \Gamma)$ is the set of possible k -neighbourhoods. The computation on $x = \sigma_1 \cdots \sigma_n$ starts from the scaffold consisting of the base node alone and the state q_0 . At step j the machine computes $(q', \gamma, p_0, \dots, p_{d-1}) = \delta(q, \sigma_j, N_k(t))$, where t is the current top, and appends node $t+1$ with label γ and ports

$$e_{t+1}(i) = \begin{cases} \text{the endpoint of } p_i \text{ followed from } t, & p_i \text{ a valid port word,} \\ \emptyset, & p_i = \emptyset \text{ or } p_i \text{ invalid,} \\ t+1, & p_i = \text{SELF.} \end{cases}$$

The new state is q' . The word is accepted if the state after the last letter lies in F ; $L(A)$ is the set of accepted words, and we write $L \in \text{SA}$ when some scaffolding automaton decides L .

Three points deserve emphasis. The descriptor ϵ (the empty port word) names the old top itself. The descriptor **SELF** lets a new node point to itself; it is part of LMR’s model, not an extension. And the machine cannot test whether two observed nodes are the same node: it sees a tree.

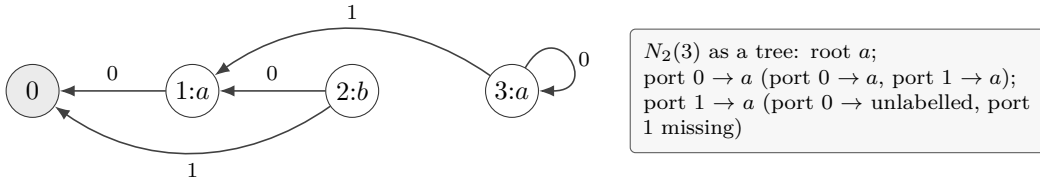


Figure 2: A scaffold of degree 2 after three letters. Node 1 was created with descriptors $(\epsilon, \emptyset)$ from top 0; node 2 with $(\epsilon, 0)$ from top 1 (port 1 follows port word 0 from node 1 and reaches node 0); node 3 with $(\text{SELF}, 0)$ from top 2. The depth-2 neighbourhood of node 3 is the tree on the right: it does not record that port 0 returns to node 3 itself.

2.6 The exact correspondence with PEGs, and why reversal appears

Theorem 2.7 (Loff–Moreira–Reis [7, Theorem 16, p. 21]). *A language $L \subseteq \Sigma^*$ is in PEG if and only if its reversal L^R is decided by some scaffolding automaton. In particular, if a finite scaffolding automaton decides L , then L^R is recognized by a total complete-match PEG.*

We use only the second sentence (the “sufficient direction”, whose proof in LMR begins on p. 23). The reversal is not an accident of notation. A scaffolding automaton consumes the input left to right and commits at each letter; LMR’s grammar for the reverse language re-enacts the automaton’s run from the grammar’s point of view, which reads the other way. For a language equal to its own reversal, such as EPAL_2 , the distinction disappears, and this is exactly why Corollary 12.3 mentions reversal invariance. The paper does not use the other direction of the theorem and makes no statement about the class **SA** as a whole.

2.7 Persistent data structures

A data structure is *ephemeral* if an update destroys the old version, and *persistent* if every old version remains available after updates; it is *fully* persistent if old versions can also be updated [2, §1]. The scaffold of Definition 2.5 is persistent by construction: nothing is ever overwritten. The data structures of this paper are persistent in the strongest sense, which Kaplan and Tarjan call *purely functional*: built only from the three LISP primitives `cons` (make a pair), `car` and `cdr` (read its two components), with no mutation and no memoization [6, p. 578, footnote 1].

The key consequence is *structural sharing*. If a list is represented as a chain of pairs, then `cons( $x$ ,  $\ell$ )` creates one new pair pointing at the old chain ℓ ; the old list and the new list share everything but that pair, and neither can tell (Figure 3). A structure that needs to “copy” an unbounded old part can instead point to it.



Figure 3: Structural sharing. Both lists exist; the new one reuses the old chain.

A *steque* (stack-ended queue) is a sequence supporting *push* (add at the front), *pop* (remove the front element), and *inject* (add at the rear); a *catenable* steque also supports concatenating two steques [6, pp. 578–579]. A data structure is *real-time* when every operation takes worst-case constant time—a constant number of primitive steps regardless of the sequence’s length [6, p. 579]. (The reader should keep this *data-structure* sense of “real-time” apart from the *machine* sense used in the earlier transfer paper, where a real-time Turing machine is one that performs a bounded number of steps per input letter.) Section 11 quotes the theorem of Kaplan and Tarjan that supplies a purely functional real-time catenable steque.

2.8 The mutable palindrome index, and why a scaffold cannot have it

Algorithms that maintain all palindromes of a growing word use a *palindromic tree* (the *eertree* of Rubinchik and Shur [8, §2.2]): one node per distinct palindrome, a *suffix link* from each node to its longest proper palindromic suffix, and, added whenever it is first discovered, an edge labelled c from the node of P to the node of cPc . Discovering that edge means walking suffix links from the current node until a node is found whose occurrence is preceded by c ; the walk can be long, but its cost is amortized and the edge is then stored *at the old node*, so it is never needed again.

A scaffold can do neither thing. It cannot walk an unbounded chain within one step (it sees distance k), and it cannot add an edge to an old node. The present paper shows what must be stored instead: not a pointer from a palindrome *type* to a type, but a pointer carried by each *occurrence* to the particular historical record that the next step will need. Section 7 records, with small counterexamples, why each cheaper idea fails.

3 A reader’s map

3.1 The viewpoint behind the paper

This paper builds directly on LMR’s framework. 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* EPAL₂ has some PEG, but *which internal recognition dynamic* makes palindrome comparison visible to that correspondence.

Finitely Presented Recognition Dynamics, abbreviated FPRD, is the research viewpoint used in this work: a finite grammar or machine description is treated as a *presentation* that generates an internal recognition *dynamic*, and the accepted language is treated as one *observation* of that dynamic. FPRD Lab is the AI-assisted research project in which the present arguments were developed; nothing in this paper requires trust in that project, its tools, or its records, and Section 14 states plainly what was checked by whom. The viewpoint has one concrete role in the proofs: we keep the *word-level* recurrence (what is true of the input), the *record-level* representation (what a persistent program stores), and the *scaffold-level* encoding (what the machine physically appends) as three separate layers, and we prove each implication between them rather than identifying them.

3.2 Three representations of one state

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 attaches 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. One consequence should be said at once, because it resolves an apparent tension later: the *content* stored in a record (its suffix link, its bridge letter, its ropes) is a function of the palindrome type alone—the receipt traces of Section 8 are defined from words—so any record of the right type would do *semantically*; what depends on the occurrence is only *which* physical record is reachable within the machine’s fixed distance.

Definitions are given at the first point where they carry proof weight, and Appendix A collects the notation. 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.

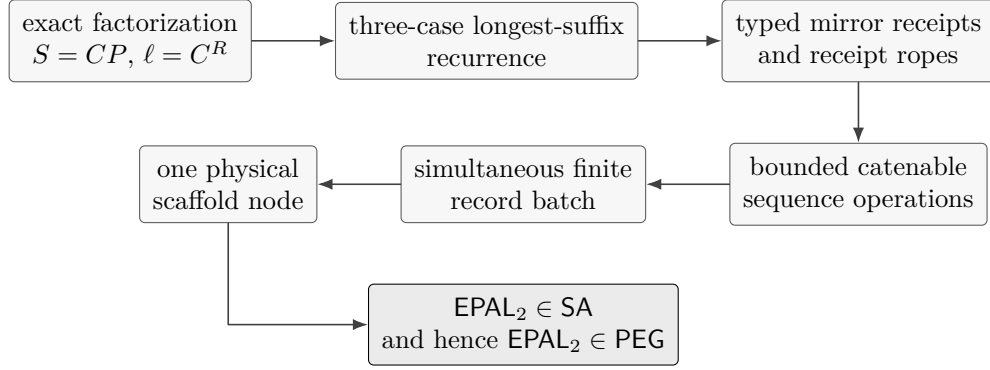


Figure 4: The proof dependency ladder. The upper row is word combinatorics; the lower row is representation and compilation. Executable checks in the public package test individual arrows but are not used as proofs of them.

4 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 (it exists, since ϵ is one), 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.

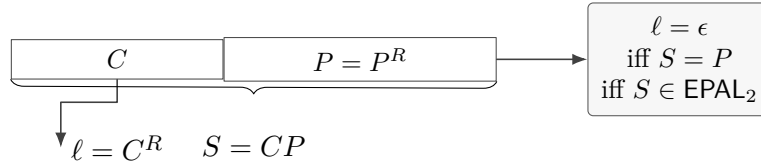


Figure 5: The occurrence-context factorization. The context is reversed so that the letter immediately preceding the current occurrence of P is at the front of ℓ .

Table 1 gives the complete 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 even suffix at all—not even the empty one—can be wrapped, so the new longest even suffix is empty. Recall that the empty suffix of a nonempty word counts as preceded by the word’s last letter; that is why row 3 is an internal update and not a reset.

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

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.

will reappear when we interpret the general recurrence. This word will be followed through every layer of the paper: its row 8 becomes the rope update of Example 8.10 and the scaffold batch of Example 12.1. What generalizes from it is the three-way case split and the use of the empty suffix; what is incidental is the particular word.

Part II

Persistent palindrome state

5 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 (a string-matching automaton whose “failure” transitions fall back to the longest matching proper suffix): 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.

5.1 Definitions

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 .

Definition 5.1 (selected suffix and displaced chunk). For an even palindrome P and a letter a , let $D_a(P)$ be the longest proper even-palindromic suffix Q of P for which

$$P = ZaQ$$

for some word Z (that is, the longest proper even-palindromic suffix *preceded by a* inside P), 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.

Definition 5.2 (suffix link, bridge, bridge block). For a nonempty even palindrome P , let $\lambda(P)$ be its longest proper even-palindromic suffix (possibly ϵ), called its *suffix link*. Write $P = RbL$ with $L = \lambda(P)$; the letter b is the *bridge* of P and $B_P = R^R$ is its *bridge block*. Because P and L are palindromes, $P = P^R = L^R b R^R = L b R^R$, so L is also a prefix of P and

$$P = RbL = LbB_P.$$

At the word level we set $\lambda(\epsilon) = \epsilon$ by convention. At the record level, the typed receipt for this empty word is a distinguished empty record $\perp$, which has no suffix-link field or bridge. Thus an occurrence of $\lambda(\epsilon)$ in a word equation means ϵ , while the same value stored in a receipt slot is represented by $\perp$.

Example 5.3. For $P = 001100$: $\lambda(P) = 00$, the bridge is $b = 1$ (the letter before the suffix occurrence of 00), $R = 001$, and $B_P = 100$; indeed $P = 001 \cdot 1 \cdot 00 = 00 \cdot 1 \cdot 100$. For $P = 0110$: $\lambda(P) = \epsilon$, $b = 0$ (the last letter), $R = 011$, $B_P = 110$. For $P = 00$: $\lambda(P) = \epsilon$, $b = 0$, $B_P = 0$.

5.2 The three-case update

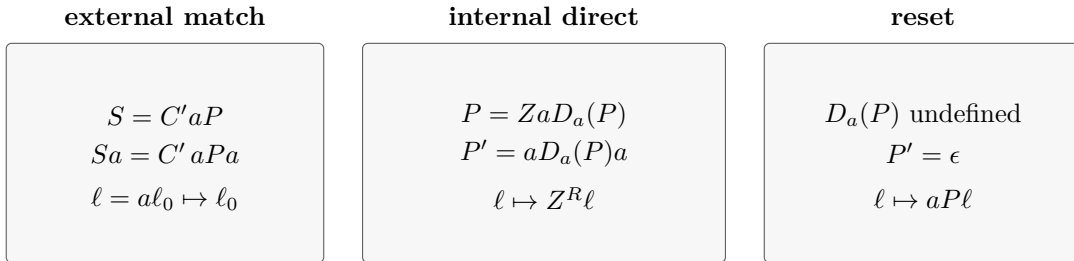


Figure 6: The exhaustive three-case update at the word level. The external case consumes one context letter; the internal case prepends a selected chunk; the reset case moves the entire failed candidate into the context.

Theorem 5.4 (Occurrence-context factorization). *After reading a , the new longest even suffix P' and reversed context ℓ' are exactly*

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

Moreover,

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

Proof. Strategy. Every new even-palindromic suffix must end with the new letter and therefore begin with it; peel the two copies of a off and what remains is an old even-palindromic suffix preceded by a . Then maximize.

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. The theorem 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.

Example 5.5 (the two internal rows). 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 $P' = 0\epsilon 0 = 00$ and $\ell' = 0\epsilon = 0$. 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 $S = 0001100 = 0 \cdot 001100$, $P = 001100$, $\ell = 0$. 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 $P' = 1 \cdot 00 \cdot 1 = 1001$ and $\ell' = 100 \cdot 0 = 1000$. 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.

5.3 The transition table has fixed width

Lemma 5.6 (chunk recursion). *Let P be a nonempty even palindrome with suffix link $L = \lambda(P)$, bridge b , and bridge block B_P . Then*

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

and for every letter $a \neq b$,

$$D_a(P) = D_a(L), \quad \omega_a(P) = \omega_a(L) b B_P,$$

with the same definedness on both sides (for $L = \epsilon$ the right-hand sides are undefined).

Proof. Every proper even-palindromic suffix of P is a suffix of L , since L is the longest one. The suffix L itself is preceded by b , so $D_b(P) = L$ and the displaced chunk is $R^R = B_P$. For $a \neq b$, the suffix L is not eligible, so every eligible suffix is a proper suffix of L preceded by a inside L , which is the definition of $D_a(L)$; writing $L = Z'aD_a(L)$ gives $P = RbZ'aD_a(L)$, so $\omega_a(P) = (RbZ')^R = Z'^R b R^R = \omega_a(L) b B_P$. $\square$

Example 5.7. For $P = 001100$ ($L = 00$, $b = 1$, $B_P = 100$): $D_1(P) = 00$, $\omega_1(P) = 100$; and $D_0(P) = D_0(00) = \epsilon$, $\omega_0(P) = \omega_0(00) \cdot 1 \cdot 100 = 0 \cdot 1 \cdot 100 = 01100$. Directly, $P = 00110 \cdot 0 \cdot \epsilon$ and $(00110)^R = 01100$.

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.

6 Typed mirror receipts

Theorem 5.4 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 inexpensive in the mutable palindromic tree of Section 2.8: a 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 a record of exactly the palindrome type 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 6.1 (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 even palindrome H , it is the longest even-palindromic suffix of the prefix ending there. Otherwise a longer even-palindromic suffix would contain H as a proper prefix as well as a suffix, giving an earlier occurrence of H . The online construction creates a record for the longest even-palindromic suffix at every step, so it has already made a birth record for $\lambda(Y)$. The empty case uses the 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.

What this does and does not say. The theorem says that the required record *exists* in the old history. It does not yet say which old field holds a pointer to it; that is Theorem 8.9(iv): in the external case it is the receipt stored in the first cell of the live context rope, and in the internal case it is the header of the old rope entry selected by the letter. Nor does the theorem say that the pointer reaches the *birth* record; any record of the right type serves (Section 3).

The word “typed” is essential. A raw endpoint does not identify the needed record at bounded depth. Proposition 6.2 makes this precise for the following *endpoint-only* discipline: store, instead of a typed pointer, the time at which the mirror copy of the required palindrome ends, and recover the record by starting from the longest even-palindromic suffix of the prefix ending there and following suffix links downward.

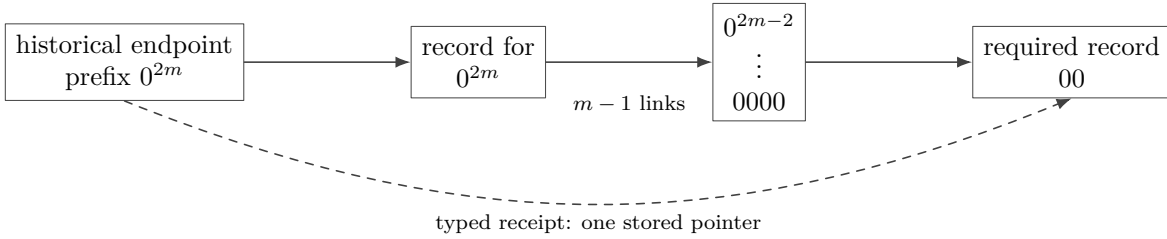


Figure 7: Endpoint information and typed information are different. The endpoint identifies a historical prefix, but its longest-palindrome record may be arbitrarily high above the required type.

Proposition 6.2 (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. $\square$

What this does and does not say. 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. What generalizes: any discipline that recovers a record by descending a suffix-link chain from an endpoint is unbounded. What is incidental: the particular family $0^{2m}1100$.

7 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.

7.1 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 the mutable palindromic tree would do: an entry for letter a pointing to the record of the *target* type $aD_a(P)a$. The target of a later transition, however, need not exist when the source type is born.

Proposition 7.1 (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. The reader may ask why the per-letter entries that the final construction *does* store (Section 12) escape this proposition. They do because they point to the record of the *selected suffix* $D_a(P)$, which exists when P is born (it is a suffix of P), and not to the record of the target $aD_a(P)a$, which may not.

7.2 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 7.2 (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 8.3 supplies exactly that justification.

7.3 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 7.3 (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 $P_{r,d} = A_r11A_r11 \cdots 11A_r$ is unchanged by reversal, so it is an even palindrome. The suffix beginning after the first block A_r11 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 (for $r = 1$, $\lambda(00) = \epsilon$ is preceded by the last letter 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.

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.

8 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?*

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.

8.1 What a receipt rope represents

Consider the current state (P, ℓ) with $\ell = c_1 c_2 \cdots c_k$. If the next input letters happen to be $c_1, c_2, \dots$ in that order, the external case of Theorem 5.4 fires each time: the palindrome becomes $c_1 P c_1$, then $c_2 c_1 P c_1 c_2$, and so on, while the context loses one letter each time. Each of those future palindromes will need its suffix link at the moment it becomes current (Theorem 6.1), and each is determined by P and a prefix of ℓ . A *receipt rope* stores the context as a sequence of cells, one per context letter, where cell i carries both the letter c_i and the typed receipt for the palindrome obtained after i external matches. Popping a cell (an external match) then exposes, in the next cell, exactly the receipt the new current palindrome needs. The rope is not merely a word with annotations: it is a walk through palindrome occurrences, *anchored* at P .

The record at the front is a *header*: the receipt of the anchor itself. If a pop makes the first wrapped palindrome current, its suffix-link receipt is promoted to the new header. This promotion is what an ordinary list of letters lacks.

8.2 Definitions

Definition 8.1 (anchored path, receipt trace, headered rope). For an even palindrome A and a word $w = c_1 \cdots c_k$, define the *anchored path* $\Pi(A, w)$ by

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

and write $\text{wrap}_w(A) = A_k$ for its final palindrome. Cell i of the path 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

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

whose first component is the *header* and whose sequence of cells is the *body*. Palindrome words in a receipt position denote typed pointers to the particular occurrence records supplied by the construction, not mere type tags. In particular, the displayed header $\lambda(\epsilon) = \epsilon$ of a rope anchored at ϵ is physically represented by $\perp$.

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 7.2 is the one-letter instance of this fact.

Example 8.2. The live context at row 7 of Table 1 is $\hat{\Pi}(001100, 0) = (00; (0, 00))$: the header is $\lambda(001100) = 00$, and the one cell records that after an external match by 0 the palindrome 00011000 would have suffix link 00.

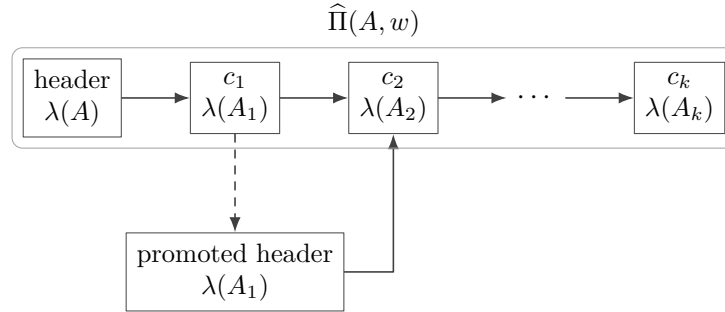


Figure 8: A headered receipt rope. Tail removes the first cell and promotes its stored receipt; the persistent remainder is shared. The positive proof must show that legal online reanchoring leaves the receipts in that shared remainder unchanged.

8.3 The bridge-shadow identity

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. Its two anchors H and P are *different words*; the claim is that wrapping them through the same context produces the same sequence of suffix links.

Lemma 8.3 (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*

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

(The statement allows $P = \epsilon$: then $H = Rb$ is any nonempty even palindrome with $\lambda(H) = \epsilon$, and the hypothesis on $w^R P = w^R$ says w^R has no nonempty even-palindromic suffix.)

Example 8.4. $H = 0000$, $P = \lambda(H) = 00$, $b = 0$, $w = 1$: the hypotheses hold ($1 \neq 0$; $w^R P = 100$ has longest even-palindromic suffix 00), and indeed $\lambda(1\,0000\,1) = \epsilon = \lambda(1\,00\,1)$.

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. Strategy. Compare the two wrapped words cell by cell. Both end in the same suffix; as long as the receipt is strictly shorter than that common suffix, the next bridge letter is read inside it and the two sides agree. The only way the receipt could grow to the full common length is excluded by the maximality hypothesis.

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.

8.4 Direct and reset ropes, and the mirror-prefix lemma

For nonempty Y with suffix link L , bridge b , and bridge block B , 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*

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

The direct rope is the context rope that the internal case of Theorem 5.4 will need: it is anchored at the palindrome $cD_c(Y)c$ that the letter c creates, and its cells carry the receipts for the displaced chunk $\omega_c(Y)$ that becomes the front of the new context. The reset rope is the analogue when nothing can be wrapped: anchored at ϵ , its cells carry the receipts for the whole failed candidate cY .

The last cell of such a rope has a receipt that can be named explicitly. This fact is used three times below (in Theorems 8.7 and 8.9), so we state it once.

Lemma 8.5 (Mirror-prefix lemma). *Let Y be a nonempty even palindrome and c a letter.*

1. *If $D_c(Y)$ is defined, the final palindrome of $\Pi(cD_c(Y)c, \omega_c(Y))$ is $F = Yc\omega_c(Y)$, and $\lambda(F) = Y$.*
2. *If $D_c(Y)$ is undefined, the final palindrome of $\Pi(\epsilon, cY)$ is $F = YccY$, and $\lambda(F) = Y$.*

In both cases the last receipt of the rope is the anchor's source Y itself.

Proof. (1) Write $Y = ZcD$ with $D = D_c(Y)$ and $\omega = \omega_c(Y) = Z^R$. Wrapping cDc through ω gives $F = \omega^R cDc\omega = ZcDcZ^R = YcZ^R$. Since F is a palindrome with prefix Y , the word $Y = Y^R$ is also a suffix of F , and it is a proper even-palindromic suffix. Suppose a longer proper even-palindromic suffix $V = yY$ existed, with y a nonempty suffix of Zc , $|y|$ even. Palindromicity of V gives $yY = Yy^R$, so Y has period $|y|$: its suffix Y'' of length $|Y| - |y|$ equals its prefix of that length, and since Y is a palindrome that prefix is the reversal of Y'' ; hence Y'' is an even palindrome. The same equality identifies y with the prefix of Y of length $|y|$, so the terminal occurrence of Y'' in Y is preceded by the last letter of y , which is c because y is a suffix of Zc . By maximality of $D = D_c(Y)$, $|Y''| \leq |D|$, i.e. $|y| \geq |Y| - |D| = |Z| + 1$; so $y = Zc$ and $V = F$ is not proper, a contradiction.

(2) Wrapping ϵ through cY gives $F = (cY)^R \epsilon (cY) = YccY$, a palindrome with prefix Y , so Y is a proper even-palindromic suffix. If a longer proper one $V = yY$ existed, with y a nonempty suffix of Ycc of even length, the same periodicity argument shows that the suffix Y'' of Y of length $|Y| - |y|$ is an even palindrome. When $|y| \leq |Y|$, the equality $yY = Yy^R$ identifies y with the corresponding prefix of Y ; because y ends in c , the terminal occurrence of Y'' is preceded by c , which would make $D_c(Y)$ defined. And $|y| > |Y|$ forces $|y| = |Y| + 2$, i.e. $V = F$, not proper. $\square$

Corollary 8.6. *With $Y = LbB$ as above and $c \neq b$: if $D_c(Y)$ is defined, write $L = ZcD_c(L)$; then $\lambda(ZcL) = L$. If $D_c(Y)$ is undefined, then $\lambda(LccL) = L$.*

Proof. By Lemma 5.6, $D_c(Y)$ is defined iff $D_c(L)$ is (for $L = \epsilon$ both are undefined). Apply Lemma 8.5 to L in place of Y : the final palindrome of $\Pi(cD_c(L)c, \omega_c(L))$ is ZcL , and that of $\Pi(\epsilon, cL)$ is $LccL$. (For $L = \epsilon$ the second case reads $\lambda(cc) = \epsilon$.) $\square$

8.5 The receipt-rope recurrence

Theorem 8.7 (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. Note that the third piece is shared between two ropes whose anchors differ: in $\Omega_c(Y)$ the cells after the bridge cell are anchored at $b(ZcL)b$, in $\Omega_b(Y)$ at bLb .

Proof. Strategy. The raw words agree by Lemma 5.6; the receipts of the shared tail agree by the bridge-shadow identity applied to all cells but the last, and the last receipts agree by the mirror-prefix lemma.

Raw words. Let $c \neq b$. By Lemma 5.6, $D_c(Y) = D_c(L)$ with the same definedness, and $\omega_c(Y) = \omega_c(L)bB$. Hence, when defined, $\Omega_c(Y)$ is anchored at the same palindrome $cD_c(L)c$ as $\Omega_c(L)$ and wraps through $\omega_c(L)$, then b , then B ; by the catenation law its first $|\omega_c(L)|$ cells are literally those of $\Omega_c(L)$, its next cell has letter b , and its remaining cells are $\Pi(H', B)$ where $H' = b \text{ wrap}_{\omega_c(L)}(cD_c(L)c)b = b(ZcL)b$ with $L = ZcD_c(L)$. In the reset case, $Z_c(Y) = \hat{\Pi}(\epsilon, cLbB)$ similarly begins with the cells of $Z_c(L) = \hat{\Pi}(\epsilon, cL)$, continues with a b cell, and ends with $\Pi(H', B)$ where $H' = b(LccL)b$. On the other side, body $\Omega_b(Y) = \Pi(bLb, B)$ since $D_b(Y) = L$ and $\omega_b(Y) = B$.

Receipts. Put $H = ZcL$ (direct case) or $H = LccL$ (reset case), so that $H' = bHb$, and write $B = B'x$ with x its last letter (B is nonempty because $|Y| - |L| - 1 \geq 1$). We claim $\Lambda(H, bB) = \Lambda(L, bB)$; the first entry of this identity is the statement that the b cell of $\Omega_c(Y)$ (resp. $Z_c(Y)$) carries $\lambda(bLb) = \rho_b$, and the remaining entries identify $\Pi(H', B)$ with $\Pi(bLb, B)$ receipt by receipt.

Apply Lemma 8.3 with H , $P := L$, and $w := bB'$. Its hypotheses hold: $H = R'cL$ has bridge $c \neq b$ and $\lambda(H) = L$ by Corollary 8.6; and $w^R L = B'^R bL$ is Y with its first letter removed, whose even-palindromic suffixes are exactly the proper even-palindromic suffixes of Y , the longest being L . The lemma gives agreement of the receipts on all cells except the last. For the last cell, the final palindromes are $Yc\omega_c(Y)$ (resp. $YccY$) on the left and YbB on the right; both have suffix link Y by Lemma 8.5. Hence $\Lambda(H, bB) = \Lambda(L, bB)$, which is the displayed identity.

Definedness. $\Omega_c(Y)$ is defined iff $\Omega_c(L)$ is, by Lemma 5.6; and $Z_c(Y)$ is defined (i.e. $D_c(Y)$ undefined) iff $Z_c(L)$ is. $\square$

Example 8.8 (the recurrence on $Y = 001100$). Here $L = 00$, $b = 1$, $B = 100$. For $c = 0$: $\Omega_0(L) = \hat{\Pi}(00, 0) = (\epsilon; (0, 00))$, $\Omega_1(Y) = \hat{\Pi}(1001, 100) = (\epsilon; (1, 11), (0, 0110), (0, 001100))$, so $\rho_1 = \epsilon$ and the recurrence predicts $\Omega_0(Y) = (\epsilon; (0, 00), (1, \epsilon), (1, 11), (0, 0110), (0, 001100))$. Direct computation of $\hat{\Pi}(00, 01100)$ gives the same five cells. The shared tail is anchored at 100001 on the left and at 1001 on the right; the two anchors differ, and their receipts agree.

8.6 Full receipt closure

Theorem 8.9 (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. Which old *fields* hold the receipts named below is fixed by the record layout of Section 12.

Proof. Strategy. Build the fresh record's bridge rope from one old rope by one pop and one inject; build its other entries by Theorem 8.7; build the new live context by Theorem 5.4.

Suppose the update creates $P = aQa$, where Q is the old longest suffix (external case) or $Q = D_a(P_{\text{old}})$ (internal case), and let $L = \lambda(P)$ as given by Theorem 6.1. Select the old rope $\Omega_a(Q)$ when $D_a(Q)$ is defined and the old reset rope $Z_a(Q)$ otherwise; call it the *source rope*. Its body is nonempty (its word is $\omega_a(Q)$, resp. aQ , and both are nonempty); write the body $(b, H)\mathcal{T}$.

- (i) *The bridge of P is b .* If $D_a(Q)$ is defined, write $Q = Z'aD_a(Q)$; then $P = aZ'aD_a(Q)a = (aZ')L$, so the bridge of P is the last letter of Z' , which is the first letter of $\omega_a(Q) = Z'^R$, i.e. b . If $D_a(Q)$ is undefined, $L = \epsilon$, the bridge of P is its last letter a , and the first letter of the reset rope's word aQ is $a = b$.
- (ii) *The header of $\Omega_b(P)$ is H .* The header of $\Omega_b(P) = \widehat{\Pi}(bLb, \omega_b(P))$ is $\lambda(bLb)$. The first cell of the source rope is obtained by wrapping its anchor $(aD_a(Q)a = L, \text{ resp. } \epsilon = L)$ by b , so its receipt is $H = \lambda(bLb)$.
- (iii) *The body of $\Omega_b(P)$ is $\mathcal{T}$ followed by (a, P) .* By (i), $P = (aZ''b)L$ with $Z' = Z''b$, so $\omega_b(P) = Z''^R a$, while the source rope's word is $\omega_a(Q) = Z'^R = bZ''^R$ (in the reset case $\omega_a(P) = Qa$ and the source word is aQ). So $\Omega_b(P)$ wraps bLb through the same letters as the source rope after its first cell, followed by one final letter a . By the catenation law the cells coincide with $\mathcal{T}$, and the final cell has letter a and receipt $\lambda(\text{wrap}_{\omega_b(P)}(bLb)) = P$ by Lemma 8.5. In symbols,

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

Operationally: pop the source body, inject the terminal cell (a, P) , and reuse the old header H .

- (iv) *The suffix-link pointer of P .* In the external case $Q = P_{\text{old}}$ and $L = \lambda(aP_{\text{old}}a)$ is the receipt stored in the first cell of the live rope $\mathcal{K}(P_{\text{old}}, \ell)$, which the pop of (3) below promotes to the new header. In the internal case $L = \lambda(aD_a(P_{\text{old}})a)$ is the header of the old entry $\Omega_a(P_{\text{old}})$, whose anchor is $aD_a(P_{\text{old}})a = P$. In both cases one old receipt names a record of type L .

Every other entry of the fresh record follows from (2), Theorem 8.7, and the finite alphabet: for $c \neq b$, the rope $\Omega_c(P)$ or $Z_c(P)$ is $\Omega_c(L)$ (resp. $Z_c(L)$) catenated with the singleton (b, H) and with body $\Omega_b(P)$ —two catenations on ropes read from the record of L , which is reached by the pointer of (iv).

Let $\mathcal{K}(P, \ell) = \widehat{\Pi}(P, \ell)$ be the live context rope. The three cases of Theorem 5.4 give

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

where P denotes the old palindrome. The first case is the catenation law read backwards. In the second case the new live rope is $\widehat{\Pi}(aD_a(P)a, \omega_a(P)\ell) = \Omega_a(P) \cdot \Pi(\text{wrap}_{\omega_a(P)}(aD_a(P)a), \ell)$, and the tail is anchored at $H := ZaP$ where $P = ZaD_a(P)$; in the third case the tail is anchored at $H := PaaP$. Lemma 8.3 applies to H , P , and $w := \ell$: $\lambda(H) = P$ by Lemma 8.5 (applied to P and a), the bridge of H is a , the first letter of ℓ differs from a in both cases, and $w^R P = \ell^R P = CP = S$, of which P is the longest even-palindromic suffix by definition. So the old body is literally reusable even though its concrete anchor changes. Equations (2)–(3) use a fixed number of rope primitives and old fields. $\square$

Example 8.10 (row 8 at the rope layer). Before row 8 the state is $P = 001100$, $\ell = 0$, with live rope $\mathcal{K} = (00; (0, 00))$. The letter is $a = 1$, the internal case with $Q = D_1(001100) = 00$, and the fresh palindrome is $P' = 1001$. The source rope is $Z_1(00) = (\perp; (1, \epsilon), (0, \epsilon), (0, 00))$ since $D_1(00)$ is undefined; so $b = 1$, $H = \epsilon$, $\mathcal{T} = ((0, \epsilon), (0, 00))$, and (2) gives

$$\Omega_1(1001) = (\epsilon; (0, \epsilon), (0, 00), (1, 1001)),$$

which agrees with the direct definition $\hat{\Pi}(11, 001)$. The suffix-link pointer of the fresh record is the header of $\Omega_1(001100) = (\epsilon; (1, 11), (0, 0110), (0, 001100))$, namely ϵ , the empty record. The new live rope by (3) is

$$\mathcal{K}' = \Omega_1(001100) \cdot \text{body } \mathcal{K} = (\epsilon; (1, 11), (0, 0110), (0, 001100), (0, 00)),$$

which is $\hat{\Pi}(1001, 1000)$ computed directly. The last cell $(0, 00)$ was created at row 7 around the anchor 001100 and is now read around the anchor 1001 ; its receipt is unchanged. The terminal cell $(1, 1001)$ of the fresh bridge rope points at the fresh record itself: this is the cycle of Part III. What generalizes: every nonreset update has exactly this shape. What is incidental: the particular lengths.

Part III

Turning bounded records into a scaffold

9 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?*

9.1 Records, fields, and batches

From here on a *record* is a finite tuple with a finite *tag*, a finite *payload* (such as a letter), and a fixed number q of *pointer fields*, each of which is missing or points to a record. A persistent program holds a fixed number s of *roots* (pointers kept in finite control) and, at each step, reads old records by following fields from the roots and creates a finite *batch* of fresh records. The fresh batch is a finite labelled directed graph: its vertices are the fresh records (*slots* $0, \dots, A - 1$ for some bound A), and each field of each slot is missing, points to an old record, or points to a slot.

In ordinary functional programming a fresh record can only point to records that already exist when its constructor runs, so the fresh batch of one step is acyclic and can be allocated in *topological order*: first the records with no fresh targets, then those pointing only to them, and so on. Three small batches show what a cycle would mean. A *self-loop* has one slot x with field $f(x) = x$. A *forward reference* has two slots x, y , created in that order, with $f(x) = y$. A *mutual cycle* has $f(x) = y$ and $g(y) = x$. All three are perfectly good finite labelled adjacency tables; none can be produced by allocating one constructor at a time, because the first constructor would have to store a pointer to a record that does not exist yet.

9.2 The cycle the palindrome construction forces

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. Example 8.10 exhibits the cycle concretely: the record for 1001 and the cell (1,1001) are both created at row 8 and point at each other.

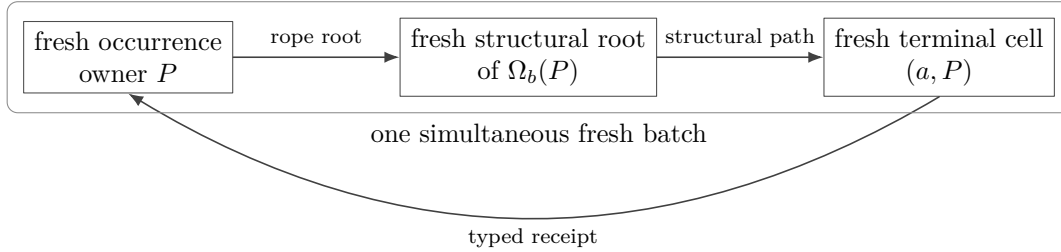


Figure 9: The owner/rope same-step cycle. The steque’s structural records are acyclic when the payload P is treated as an opaque atom. Substitution of the fresh owner pointer creates the cycle only at the batch boundary.

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 9; 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. The palindrome batch is larger than the three toy batches but no different in kind.

10 Bounded simultaneous record transducers

We now define the source machine precisely and compile it into the scaffold model of Definition 2.6. This section is independent of palindromes: a reader may regard its input as an arbitrary bounded persistent-record update.

10.1 The source model

Definition 10.1 (Bounded simultaneous record transducer). 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, and control does not branch on whether two old addresses name the same record.

An *old address* is a root together with a sequence of at most r field names; it is dereferenced by starting at the root and following the named fields. “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.

10.2 Tuple packing

The scaffold appends one node per letter; the transducer creates up to A records per letter. *Tuple packing* resolves the tension: the one physical node v_t appended at time t represents the entire batch born at time t , and a finite *slot number* $i < A$ says which logical record is meant. A logical pointer is decoded as a pair (v_t, i) . The physical node’s label records, for each of its ports, the slot number of the record that port’s target stands for; old logical field traversal therefore alternates between a physical port and the finite slot tag read at the node that owns the port. A fresh pointer uses **SELF** to return to v_t and carries the desired fresh slot in the new label. The physical graph remains a legal scaffold; “record” is an interpretation placed on its ports and label.

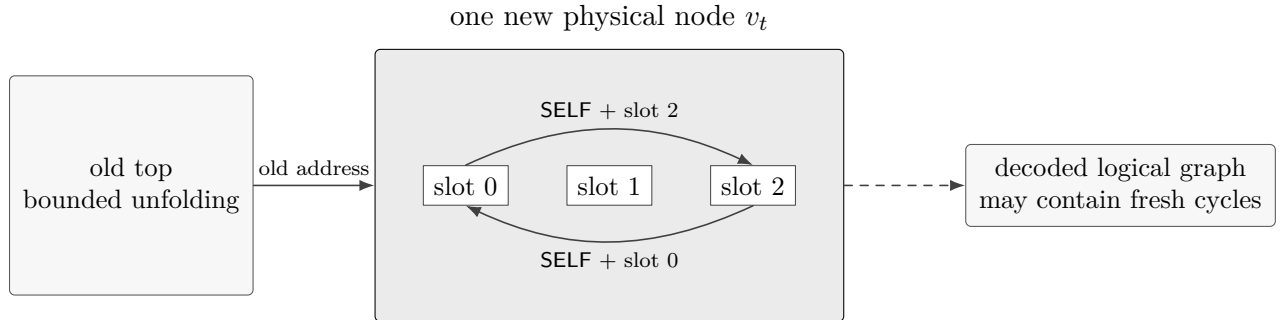


Figure 10: Tuple packing. Several logical fresh records occupy finite slots of one physical scaffold node. Every intra-batch edge has the same physical target **SELF**; its finite tag selects the logical slot.

Theorem 10.2 (Simultaneous batch compiler). *Every transducer of Definition 10.1 compiles exactly and letter-synchronously into a finite scaffolding automaton of degree*

$$d = s + Aq$$

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

Proof. Strategy. Fix the encoding (ports for roots and fields, slot tags in labels); show that every old address compiles to a port word of length at most $r + 1$ that the transition can compute from its neighbourhood; show that the decoded fresh batch equals the source batch as a finite labelled graph; then induct on the input.

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 for each of its d ports the finite target-slot tag of the record that port stands for (when the target is present).

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, because each slot tag is stored in the label of the node whose port is being followed, and that node lies within distance $m \leq r$ of the old top. 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. LMR's transition sees the labels of all nodes within distance k (Definition 2.5), so $k = r + 1$ suffices both to compute the word and to read the tags along it.

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 of Section 9, 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.

Remark 10.3 (Where the cycle lives). The physical scaffold still has only backward edges and self-loops, exactly as LMR's definition requires. A logical record pointer is the larger coordinate (v, i) , not the physical node v alone. Following a physical SELF edge keeps v fixed while its finite edge tag changes the logical slot from i to i' . An arbitrary finite cycle in the decoded fresh batch is

therefore represented by one physical self-loop together with finite slot dynamics. The encoding has moved cyclicity from physical time into the finite decoder state.

11 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).

11.1 Why an ordinary list is not enough

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* (a suspended computation stored in place of its result); copying is unbounded in the current step, while an unevaluated thunk merely moves that unbounded work to an unspecified later step. *Memoization*—caching a computed result inside the structure—is a side effect and is excluded from the strict model below. 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

11.2 The inherited theorem and its model

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 11.1 (Kaplan–Tarjan [6, Theorem 5.1, p. 592]). *There is a purely functional regular catenable steque whose push, pop, inject, and catenation operations take worst-case constant time and preserve the representation.*

Here “regular” is the invariant of Kaplan and Tarjan’s representation (§5.1 of their paper) that each operation is proved to restore; the reader may treat it as “in the normal form the operations expect”. For their paper, Kaplan and Tarjan define “purely functional” to mean built only from `car`, `cons`, and `cdr`—pair construction and the two projections—with memoization explicitly excluded [6, p. 578, footnote 1]. This strict model is stronger for our purpose than a bare unit-cost RAM claim.

One scoping remark is essential for the next theorem. The steque representation is recursive: a steque of elements of type A contains buffers and a child steque whose elements are *pairs* over A , and so on [6, §5.1, pp. 588–589]. The operations inspect those pairs and buffers freely; they are the structure’s own constructors. What the operations never inspect, compare, hash, or traverse is an element of the base type A . In our application the base elements are typed receipts.

Theorem 11.2 (Opaque-atom staging). *Let a strict purely functional sequence operation, parameterized by a base element type A , use finite constructor tests and a worst-case bounded number of `car`, `cons`, and `cdr` operations. It may inspect the representation’s own pairs, buffers, and recursive child structures, but it may not inspect, compare, hash, traverse, or identity-test values of the base type A . 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. Strategy. Run the operation with a variable in place of each new element; the run cannot look at the variable, so its control flow and allocations are fixed; then substitute.

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 10.2 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 9 depicts precisely this boundary.

Theorem 11.1 satisfies the hypotheses for the rope operations in Theorem 8.9. 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. Appendix D 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; nothing in the proof depends on that appendix.

12 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?*

12.1 The record layout

We assemble the construction. The persistent program keeps $s = 2$ roots: the *current occurrence record* and the *live context rope*. Finite control remembers only whether the current palindrome is empty.

An *occurrence record* for a nonempty palindrome P has the following fields.

1. A *suffix-link field*: a typed pointer to a record of type $\lambda(P)$, or the empty record $\perp$ if $\lambda(P) = \epsilon$.
2. A *bridge payload*: the letter b_P .
3. For each letter $c \in \{0, 1\}$, a *table entry*, which is either *direct*—a tag, a typed pointer to a record of type $D_c(P)$ (the *transition receipt*), and the rope root of $\Omega_c(P)$ —or *reset*—a tag and the rope root of $Z_c(P)$.

The empty record $\perp$ has no suffix link and no bridge, and its two entries are the reset ropes $Z_c(\epsilon) = \widehat{\Pi}(\epsilon, c) = (\perp; (c, \epsilon))$; it is created in the first batch. A *rope value* is a header receipt together with a steque root whose elements are cells $(c, \text{receipt})$.

The transition receipts materialize the chunk recursion of Lemma 5.6 persistently: for the bridge letter the transition receipt *is* the suffix link ($D_b(P) = \lambda(P)$), and for the other letter it is *copied* from the corresponding entry of the suffix-link record ($D_c(P) = D_c(\lambda(P))$), one old pointer read at creation time. This is what Proposition 7.3 demanded: the selector that a later step needs is already at depth one, not at the end of a chain.

12.2 One step

On input a , inspect the live rope’s header and first cell and the a -entry of the current record. Theorem 5.4 selects the case: external if the first cell’s letter is a ; internal if the a -entry is direct; reset otherwise. Then:

External ($P' = aPa$).

Source rope: the a -entry rope of the current record ($\Omega_a(P)$ or $Z_a(P)$). Suffix link of P' : the receipt in the first cell of the live rope. Live rope: pop.

Internal ($P' = aQa$, $Q = D_a(P)$).

Source rope: the a -entry rope of the record of Q , reached through the a -entry’s transition receipt of the current record. Suffix link of P' : the header of the current record’s a -entry rope $\Omega_a(P)$. Live rope: catenate $\Omega_a(P)$ with the body of the old live rope.

Reset ($P' = \epsilon$).

No fresh occurrence record; the root becomes $\perp$. Live rope: catenate $Z_a(P)$ with the body of the old live rope.

In the two nonreset cases the fresh record is filled by Theorem 8.9: bridge b and bridge rope from the source rope by one pop and one inject (equation (2)); transition receipt for b equal to the suffix link; for the other letter c , transition receipt copied from the c -entry of the suffix-link record and rope built by Theorem 8.7 with two catenations. Theorems 11.1 and 11.2 turn those operations into one bounded simultaneous record batch, and Theorem 10.2 appends that batch as the one scaffold node required for the arriving letter. Acceptance is the finite empty-body test on the live rope (Theorem 5.4).

Example 12.1 (row 8 at the scaffold layer). Continuing Example 8.10, the batch appended at time 8 contains the following logical records (slot numbers are one possible assignment).

slot	record	pointer fields
0	occurrence owner 1001	suffix link $\rightarrow \perp$ (old); bridge 1; entry 1 $\rightarrow$ slot 1; entry 0 $\rightarrow$ slot 2
1	rope value $\Omega_1(1001)$	header ϵ (old receipt $\perp$); body root $\rightarrow$ slot 4
2	rope value $Z_0(1001)$	header $\perp$; body root $\rightarrow$ slot 5
3	live rope $\mathcal{K}'$	header $\perp$; body root $\rightarrow$ slot 6
4	steque node: $\mathcal{T}$ injected with $(1, \cdot)$	tail $\rightarrow$ old cells $(0, \epsilon), (0, 00)$; terminal cell's receipt $\rightarrow$ slot 0
5	steque node: $(0, \epsilon) \cdot (1, \epsilon) \cdot \text{body}(\Omega_1(1001))$	$\rightarrow$ old cell, $\rightarrow$ fresh singleton, $\rightarrow$ slot 4
6	steque node: $\text{body}(\Omega_1(001100)) \cdot \text{body}(\mathcal{K})$	$\rightarrow$ old cells only
≥ 7	further structural steque records	old or earlier-slot targets

The two roots of the new node point to $(v_8, 0)$ and $(v_8, 3)$. Every intra-batch pointer is a SELF port whose tag names the slot; in particular the field of slot 4 that holds the terminal receipt is SELF tagged 0, and the entry field of slot 0 is SELF tagged 1: that is the cycle of Figure 9, encoded by two ports of one node. The entry $Z_0(1001)$ is $(\perp; (0, \epsilon), (1, \epsilon), (0, \epsilon), (0, 00), (1, 1001))$, obtained by Theorem 8.7 from $Z_0(\epsilon) = (\perp; (0, \epsilon))$, the singleton $(1, \rho_1)$ with $\rho_1 = \epsilon$, and body $\Omega_1(1001)$; the transition receipt of the 0-entry is absent (reset), that of the 1-entry is the suffix link $\perp$. What generalizes: the shape of the batch. What is incidental: the slot numbering and the particular steque node structure, which Theorem 11.1 fixes but which the proof never inspects.

12.3 Uniform constants

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. Concretely, let K bound the primitive steps and fresh allocations of any one steque operation of Theorem 11.1 (such a bound exists because each operation is a finite strict decision tree and the elements are opaque). One step performs at most six steque operations (Appendix B), so it allocates at most $A = 6K + 4$ slots ($6K$ structural, one owner, two table ropes, one live rope) and reads old records along paths of length at most $r = 6K + r_0$, where r_0 is the fixed maximum number of non-steque dereferences in the displayed one-step algorithm: root, entry, transition receipt, suffix link, and source- record selectors. Every record constructor has fixed arity; let q be the maximum after padding. Theorem 10.2 then

yields a scaffolding automaton of degree $d = 2 + Aq$ and distance $r + 1$. These are machine constants: K is not an input-dependent estimate but the size of a fixed decision tree, and nothing grows with the prefix length, the current palindrome, the rope length, or the number of historical occurrences. The construction retains an unbounded history, but every next transition observes only this fixed neighbourhood and appends exactly one physical node.

12.4 The run invariant and the theorem

After reading a prefix $S = CP$:

1. the decoded occurrence root names a record for the current occurrence of the longest even-palindromic suffix P (the empty record if $P = \epsilon$);
2. the decoded live-rope root denotes $\widehat{\Pi}(P, C^R)$;
3. every table entry in every reachable occurrence record has the word and receipt semantics of Section 8 (its rope is Ω_c or Z_c of its record's type, and its transition receipt, when present, names a record of type D_c of that type); 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 12.2 (Direct palindrome scaffolding). *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: the invariant holds initially; if it holds before a letter, Theorem 5.4 determines the new word state, Theorem 8.9 shows that the fresh record, its entries, and the new live rope computed from old fields have the semantics the invariant requires, Theorem 11.2 shows that this computation is one bounded simultaneous batch, and Theorem 10.2 shows that the appended node decodes to it. The automaton accepts exactly when the decoded left context is empty, which by Theorem 5.4 is exactly EPAL_2 . $\square$

Corollary 12.3 (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. Theorem 2.7 applied to Theorem 12.2 gives $\text{EPAL}_2^R \in \text{PEG}$. Since the reversal of ww^R is again of the form uu^R , the language EPAL_2 is invariant under reversal, so $\text{EPAL}_2^R = \text{EPAL}_2$. $\square$

What this does and does not say. The corollary refutes LMR's Conjecture 7. It does not say anything about palindromes of odd length, about other alphabets (the construction works verbatim over any finite alphabet, with one table entry per letter, but we have not stated it so), or about the class SA beyond this one language.

Part IV

Result, meaning, and evidence

13 What the direct route contributes

The corollary settles no membership question that was still open: the earlier transfer paper [5] 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.

13.1 Comparison of the two proof routes

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

Coordinate	Earlier transfer route [5]	Direct receipt route (this paper)
Inherited theorem	Slisenko’s real-time palindrome recognition [9] (Galil’s construction [4] is a second source)	Kaplan–Tarjan real-time catenable steques [6]
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 machine-to-scaffold 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.

13.2 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 and their transition receipts. 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.

14 Assurance, provenance, and the viewpoint

14.1 What kind of evidence supports what

The theorems of this paper have ordinary written mathematical proofs. The public package also contains executable checks; they answer different questions, and none of them changes the mathematical status of a theorem.

Layer	What it establishes
Written proof	the stated theorem for all objects satisfying its hypotheses
Finite audit	exact behavior over one explicitly bounded search domain
Source audit	the external theorem or model says what the paper attributes to it
Specialist review	an independent reader has attacked correctness and positioning
Novelty review	the contribution has been compared broadly enough to support priority language

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, on exhaustive domains of words up to a fixed length and on random longer instances (counts are recorded in the package’s `audit/RUN-OUTPUT.txt`). 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; the compiler checker generates bounded record heaps and cyclic batches without palindrome semantics; the source-interface checker inspects a pinned catenable implementation 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 real-time transfer route [5]. 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.

14.2 Provenance and AI assistance

The arguments were developed in FPRD Lab, an AI-assisted research project. The mathematics is the author’s responsibility; AI systems drafted and checked text and code under the author’s direction, and an adversarial pass by such a system found, before publication, that an earlier version of the source model excluded the very cycle its own receipt equation creates (Section 9); the present simultaneous model is the repair. The primary sources [7, 6] were read in full and the exact statements used are quoted with page numbers; copies and extracted text are held locally and are not part of the public repository. No proof assistant is used for this paper; an existing Lean development of the project independently reconstructs LMR’s sufficient direction, but the paper cites the published theorem and does not rely on that reconstruction.

14.3 The viewpoint, stated more fully

The viewpoint of Section 3 can be stated as a discipline with four layers, each implication between which is a theorem obligation:

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—and their quotient maps.
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.

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 transfer route of [5] studies a different presentation-generated dynamic with the same language observation; the two routes therefore complement one another even though both end at $\text{EPAL}_2 \in \text{PEG}$. 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. This failure-to-invariant progression (Section 7) is where the viewpoint 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.

A Notation

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
$\lambda(P)$	suffix-link word: longest proper even-palindromic suffix of nonempty P ; $\lambda(\epsilon) = \epsilon$ by convention
b_P, B_P	bridge letter and bridge block: $P = \lambda(P) b_P B_P$
$D_a(P)$	longest proper even suffix of P immediately preceded by a
$\omega_a(P)$	reversed word displaced into context when $D_a(P)$ is wrapped
$\Pi(A, w), \text{wrap}_w(A)$	anchored palindrome path obtained by wrapping A through w , and its final palindrome
$\Lambda(A, w), \widehat{\Pi}(A, w)$	its receipt trace; its headered rope
$\Omega_a(P)$	direct receipt rope when $D_a(P)$ exists
$Z_a(P)$	reset receipt rope when $D_a(P)$ does not exist
$\mathcal{K}(P, \ell)$	the live context rope $\widehat{\Pi}(P, \ell)$
$\perp$	the distinguished record representing a typed receipt to the empty palindrome
$\text{SELF}, \emptyset$	scaffold descriptors: the node being appended; no edge
(v, i)	decoded logical record: physical node v and finite slot i
s, A, q, r	roots, slots per step, fields per record, old dereference radius

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

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 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 the 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 table-rope values, and the live-rope value. Thus $A = 6K + 4$ is a conservative simultaneous allocation bound.

Old observation is bounded similarly. The non-steque dereferences of one step are: root to current record; its a -entry; the entry’s transition receipt (internal case); the source record’s a -entry and rope; the suffix-link pointer obtained by Theorem 8.9(iv); and the c -entry of the suffix-link record—at most eight. Inlining at most six operation trees gives the conservative radius $r = 6K + 8$. With $s = 2$ roots and A slots, Theorem 10.2 yields physical degree $d = 2 + Aq$ and distance $r + 1$. Sharing may

make the actual numbers smaller, and an optimized layout may combine some values, but no such optimization is used by the proof. 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, and records each theorem under an identifier in the project’s claim ledger. FPRD-D36 is Definition 10.1. FPRD-T125 is the receipt package comprising Theorems 5.4, 6.1, 8.7, and 8.9, with Lemmas 8.3 and 8.5. FPRD-T126 is Theorem 10.2. FPRD-T127 is Theorem 11.2. FPRD-T128 is Theorem 12.2 and Corollary 12.3. These identifiers are bookkeeping; they carry no mathematical weight.

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. 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.

D Modern source calibration

Viennot, Wendling, Guéneau, and Pottier give a later OCaml implementation and a separate Rocq implementation of Kaplan–Tarjan catenable dequeues [10]. 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’s Theorem 11.1 as the mathematical premise. The modern source is a hostile implementation calibration. At the pinned commit recorded in the package’s source audit, its operational core is strict and nonrecursive, but the audit found two exact mismatches with an opaque finite-record interface: four classifiers read an exact unbounded buffer length, and 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 12.2.

This separation matters because “functionally verified” and “satisfies the bounded-record cost interface” are different coordinates. 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.

Acknowledgment of inherited work

Kaplan–Tarjan catenable steques and the Loff–Moreira–Reis scaffold/PEG correspondence are inherited mathematical results. The Viennot et al. implementations are external work used for calibration. This paper 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 package files.

References

- [1] Noam Chomsky and Marcel-Paul Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, Studies in Logic and the Foundations of Mathematics, pages 118–161. North-Holland, Amsterdam, 1963.
- [2] James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989. [https://doi.org/10.1016/0022-0000\(89\)90034-2](https://doi.org/10.1016/0022-0000(89)90034-2).
- [3] Bryan Ford. Parsing expression grammars: A recognition-based syntactic foundation. In *Proceedings of POPL 2004*, pages 111–122. ACM, 2004. <https://doi.org/10.1145/964001.964011>.
- [4] Zvi Galil. Palindrome recognition in real time by a multitape Turing machine. *Journal of Computer and System Sciences*, 16(2):140–157, 1978. [https://doi.org/10.1016/0022-0000\(78\)90042-9](https://doi.org/10.1016/0022-0000(78)90042-9).
- [5] Joshua Gay. Real-time multitape languages transfer to parsing expression grammars. FPRD Lab review paper, released 5 August 2026. <https://gitlab.com/insight-forge/boxlab/fprd-lab/-/tree/main/publications/persistent-stacks-scaffolding>.
- [6] 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>.
- [7] Bruno Loff, Nelma Moreira, and Rogério Reis. The computational power of parsing expression grammars. *Journal of Computer and System Sciences*, 111:1–21, 2020. <https://doi.org/10.1016/j.jcss.2020.01.001>. Expanded preprint arXiv:1902.08272v2 (2020); definition and theorem numbers in this paper follow the preprint.
- [8] Mikhail Rubinchik and Arseny M. Shur. EERTREE: An efficient data structure for processing palindromes in strings. *European Journal of Combinatorics*, 68:249–265, 2018. <https://doi.org/10.1016/j.ejc.2017.07.021>. Preprint arXiv:1506.04862.
- [9] 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>.
- [10] Jules Viennot, Arthur Wendling, Armaël Guéneau, and François Pottier. Verified purely functional catenable real-time dequeues. arXiv:2505.07681v3, 3 July 2026. <https://arxiv.org/abs/2505.07681v3>.

Afterword

Persistent receipts and simultaneous records: a direct scaffold for even palindromes

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 direct palindrome paper makes the presentation question concrete: what must an immutable recognizer retain so that its next update needs only bounded access? Knowing that even palindromes have a recognizer does not answer that question.

The Lab separated three levels: an exact word recurrence for the longest even-palindromic suffix; persistent records implementing that recurrence; and physical nodes allowed by a scaffolding automaton. It did not identify a useful semantic state with an already implementable machine state. Each change of representation needed its own proof.

The unsuccessful representations were productive. An old endpoint did not necessarily expose the needed suffix. A table fixed when a palindrome type was born could not contain transitions discovered later. An unanchored rope lost occurrence information, and bounded bridge chasing could require arbitrarily many steps. These failures led to typed, occurrence-specific receipts and anchored receipt ropes. The invariant explaining when a shared tail keeps the same receipt information is part of the construction, not an empirical pattern inferred from examples.

A further audit found that the proposed allocation order excluded a cycle required by the receipts themselves. The repair was a simultaneous batch of logical records, encoded in finite slots of one physical node. The scaffold's existing SELF operation represents the fresh cycle. This yielded both a direct palindrome construction and a reusable compiler for bounded persistent updates with fresh feedback.

Inherited ingredients and the boundary of the contribution

Loff, Moreira, and Reis supply the scaffold model and the scaffold-to-PEG theorem. Kaplan and Tarjan supply the purely functional real-time catenable structure. The eertree literature provides a useful comparator whose mutable type-level updates the scaffold cannot simply copy. Those sources remain credited in the paper.

The earlier persistent-stack transfer already established the same palindrome membership conclusion. This paper contributes a different internal explanation and compiler, not a second discovery of that conclusion. The two routes share the Loff-Moreira-Reis bridge. Finite word and pointer checks test the interfaces; they do not prove the universal theorem, and this paper has no complete proof-assistant formalization.

Where to inspect the contribution: the paper, Sections 3, 7–10, and 13–14. These sections contain the representation layers, escape families, allocation repair, and evidence boundaries.