

Persistent Receipts and Simultaneous Records

A Direct Scaffold for Even Palindromes

Joshua Gay · FPRD Lab, Insight Forge

Draft 5 · 21 September 2026

We construct an online palindrome recognizer from persistent records. Typed receipts identify the historical palindrome records needed by each update; a shared-tail identity permits their sequences to be reused. A simultaneous-record compiler packs each bounded batch, including cycles among fresh records, into one scaffold node. The direct recognition theorem uses the strict, opaque sequence interface stated in Section 11 and symbolic finite resource bounds.

I The construction

1 The question and the direct route

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. $P = \{ww^r \mid w \in \{0,1\}^\} \notin \text{PEG}$.*

An earlier paper [5] derives palindrome membership in PEG from a real-time multitape recognizer and a machine-to-scaffold compiler. Here we develop the persistent-record construction directly.

1.2 What this paper does

The recognizer stores the longest even-palindromic suffix and the context to its left. Its receipts make the next update accessible through a bounded number of record operations. The sequence interface and its source-level implementation evidence are treated separately in Section 11 and Appendix D.

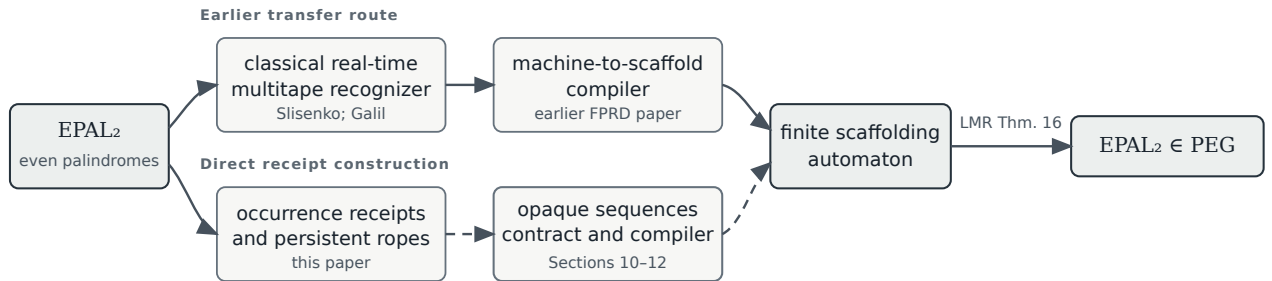


Figure 1. The transfer route and the receipt route. The latter uses the sequence contract of Section 11; both use the Loff–Moreira–Reis correspondence.

The construction has two main ingredients:

- **Receipt closure.** Each input letter updates the longest suffix and its context using a fixed number of persistent sequence operations (Theorem 8.9).
- **Simultaneous compilation.** A bounded graph of fresh records, including fresh cycles, can be encoded in one scaffold node (Theorem 10.2).

1.3 How to read this paper

Sections 2–4 define the machine, distinguish words from their stored records, and give a running example. Sections 5–8 prove the receipt recurrence. Sections 9–12 compile and assemble the recognizer. The theorem numbers are retained from the previous edition.

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 has terminal and nonterminal symbols, a start symbol, and rules that replace one nonterminal by a finite string of symbols [2, pp. 120–122]. It generates the terminal words reachable from its start symbol.

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 $R(A)$.

A grammar is *total* when recognition terminates on every input. We use complete-match recognition: a word is accepted only when the start expression consumes the entire input. Let PEG denote the languages of total PEGs [7, Definition 4].

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

The context-free alternatives in Example 2.1 cannot simply be read as ordered PEG choices: a successful alternative commits before a later failure can reconsider it. We instead use LMR's machine characterization below.

2.5 Scaffolding automata

A scaffolding automaton reads one letter and appends one immutable graph node per step. It chooses the new label and outgoing edges from a fixed-radius labelled unfolding of the previous top. Edges may point to boundedly accessible old nodes, the new node itself, or nowhere.

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 $x = \sigma_1 \dots \sigma_n$ 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 $L \in \mathbf{SA}$ 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.

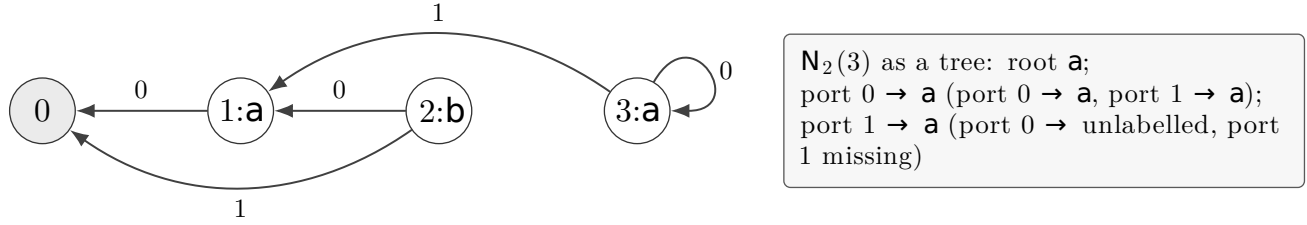


Figure 2. A scaffold of degree 2 after three letters. Node 1 was created with descriptors $(\epsilon, \emptyset)$ from top 0; node 2 with $(\epsilon, 0)$ from top 1 (port 1 follows port word 0 from node 1 and reaches node 0); node 3 with $(\text{SELF}, 0)$ from top 2. The depth-2 neighbourhood of node 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 the scaffold-to-PEG direction. The reversal causes no difficulty because $\text{EPAL}_2^R = \text{EPAL}_2$.

2.7 Persistent data structures

A persistent data structure retains old versions after an update. Here records are immutable: updates create new records and may share old ones, but cannot change their fields.

The key consequence is *structural sharing*. If a list is represented as a chain of pairs, then $\text{cons}(x, \ell)$ 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.

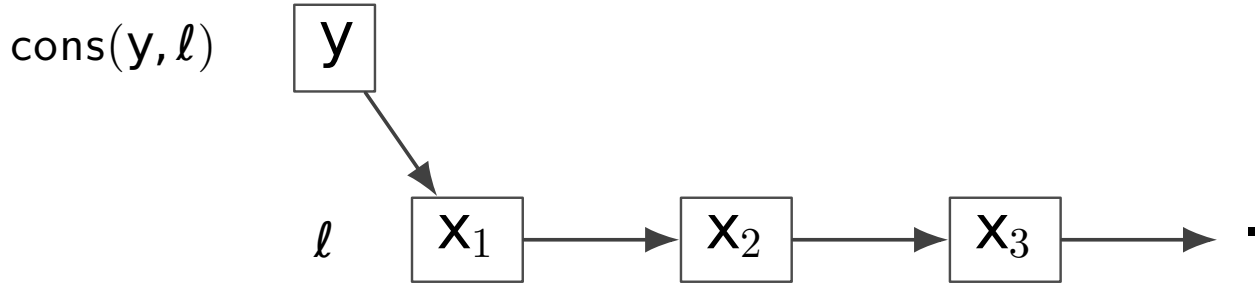


Figure 3. Structural sharing. Both lists exist; the new one reuses the old chain.

A *steque* supports insertion at either end and removal at the front. A catenable steque also joins two sequences. We require worst-case bounds for every operation, independent of sequence length; Section 11 states the exact interface.

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 Rubinichik 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 Words, records, and scaffold nodes

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.

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.

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.

A palindrome type is a word; an occurrence is a particular interval of the input. The record's semantic fields depend on its type, but its physical address belongs to its occurrence. Receipts preserve access to a suitable historical record without modifying a type-global table.

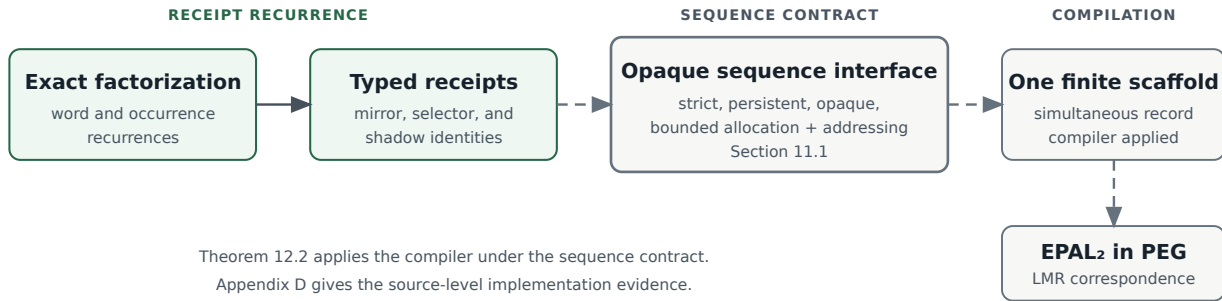


Figure 4. The proof dependency ladder. The upper row is word combinatorics; the lower row is representation and compilation. Executable checks in the public package test individual arrows but are not used as proofs of them.

4 A running palindrome trace

For each prefix S , let $P = P(S)$ be its longest even-palindromic suffix and write $S = CP$, $\ell = C^R$. The recognizer accepts exactly when $\ell = \epsilon$. Table 1 follows the input 00011001.

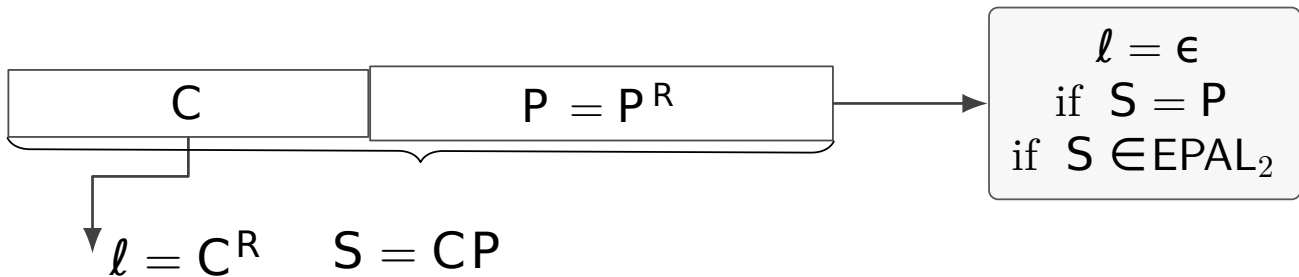


Figure 5. The occurrence-context factorization. The context is reversed so that the letter immediately preceding the current occurrence of P is at the front of ℓ .

An external update matches the first context letter; an internal update selects a suffix inside the current palindrome; a reset leaves the empty palindrome.

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.

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

After the second letter, $P = 00, \ell = \epsilon$. The third letter gives an internal update to another occurrence of 00 . Row 8 will illustrate the receipt and scaffold representations in Examples 8.10 and 12.1.

II Persistent palindrome state

5 The occurrence-context recurrence

The state tracks the longest even-palindromic suffix and its left context. The recurrence below avoids searching an unbounded suffix-link chain.

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.

When defined, $D_a(P)$ is the suffix that can be extended by the next letter a ; $\omega_a(P)$ is the displaced context.

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, $P = P^R = L^R b R^R = LbR^R$, 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

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

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

external match	internal direct	reset
$S = C' aP$ $Sa = C' aPa$ $\ell = a\ell_0 \rightarrow \ell_0$	$P = ZaD_a(P)$ $P' = aD_a(P)a$ $\ell \rightarrow Z^R \ell$	$D_a(P) \text{ undef ned}$ $P' = \epsilon$ $\ell \rightarrow aP\ell$

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.

Moreover,

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

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

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

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

In Table 1, rows 2, 5, 6, and 7 are external matches; rows 3 and 8 are internal-direct updates; rows 1 and 4 are resets. The final equivalence in the theorem explains why row 2 is accepting and the final row is not. 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 $(00110)^R = 01100$.

For the bridge letter, the suffix link supplies the transition. Every other letter reuses the suffix-link record's stored transition. The alphabet is fixed, so the table has fixed width.

6 Typed mirror receipts

Theorem 5.4 reduces online recognition to concatenating 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.

The word **001100** is a type; each interval spelling it is an occurrence. A receipt points to an old record of the required type.

Here "typed" specifies the palindrome represented by the target record.

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. $\square$

The mirror theorem establishes that the required record occurs in the old history. The receipt construction must additionally expose a pointer to it at bounded depth.

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.

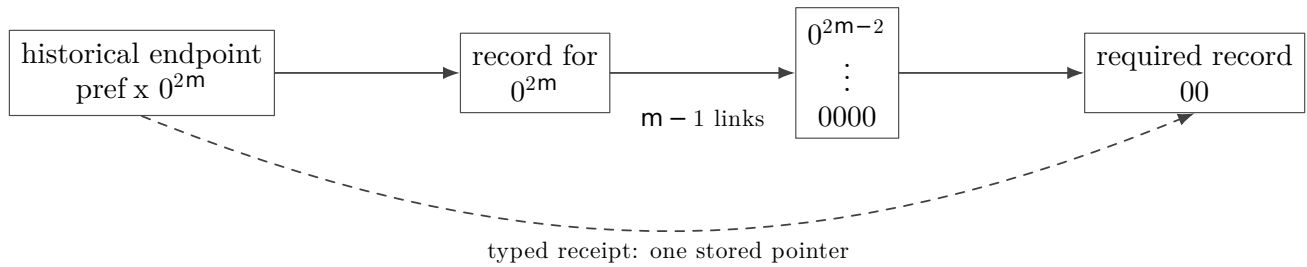


Figure 7. Endpoint information and typed information are different. The endpoint identifies a historical prefix, but its longest-palindrome record may be arbitrarily high above the required type.

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. $\square$

This rules out the endpoint-only representation; it is not a lower bound against all scaffolding automata.

7 Why simpler representations fail

The following counterexamples explain why the representation needs occurrence receipts, anchor information, and stored direct transitions.

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 $A_r = 0^{2r}$,*

$$P_r = A_r 11 A_r, \quad T_r = 1 A_r 1.$$

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 . $\square$

A table fixed at a type's first appearance cannot include this later transition. The construction therefore transports receipts with occurrences.

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

Thus each of the first d records rejects the requested bridge letter 0 and delegates to its suffix link. At $P_{r,0} = A_r$, the bridge letter before its longest proper even suffix is 0 (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.

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

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

We now construct anchored receipt ropes whose shared tails retain their required receipts.

8 Receipt ropes and shadow transport

8.1 What a receipt rope represents

A rope for context $\ell = c_1 \dots c_k$ stores each letter together with the receipt needed if that letter extends the current palindrome. Its header is the receipt for the current anchor.

8.2 Definitions

Definition 8.1 (anchored path, receipt trace, headered rope). For an even palindrome A and a word $w = c_1 \dots c_k$, 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

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

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 $\perp$.

Taking a tail promotes the first cell's receipt to the new header. Sharing a tail is valid only when its receipt trace, as well as its letters, agrees.

Example 8.2. The live context at row 7 of Table 1 is $\widehat{\Pi}(001100, 0) = (00; (0, 00))$: 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.

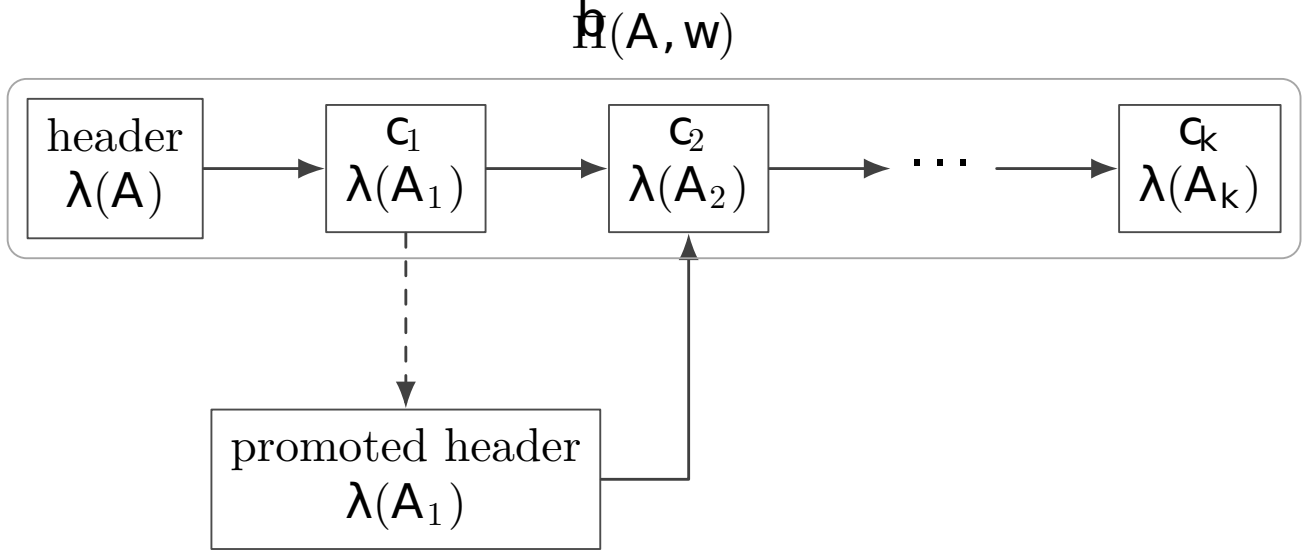


Figure 8. A headered receipt rope. Tail removes the first cell and promotes its stored receipt; the persistent remainder is shared. The positive proof must show that legal online reanchoring leaves the receipts in that shared remainder unchanged.

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 $\lambda(100001) = \epsilon = \lambda(1001)$.

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

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

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

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

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

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

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

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

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

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

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

The length inequality keeps the next inspected bridge inside the proper suffix and allows the induction to continue.

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*

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

When it is undefined, define the *reset rope*

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

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 next lemma identifies the final receipt used in Theorems 8.7 and 8.9.

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,

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

The off-bridge rope is the old corresponding rope, one new bridge cell, and the shared tail of the bridge rope. The proof must establish equality of their receipts as well as their words.

Proof. Raw words. Let $c \neq b$. By Lemma 5.6, $D_c(Y) = D_c(L)$ with the same definedness, and $\omega_c(Y) = \omega_c(L) b B$. 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, $Z_c(Y) = \widehat{\Pi}(\epsilon, cLbB)$ similarly begins with the cells of $Z_c(L) = \widehat{\Pi}(\epsilon, cL)$, continues with a b cell, and ends with $\Pi(H', B)$ where $H' = b(LccL)b$. On the other side, $\text{body } \Omega_b(Y) = \Pi(bLb, B)$ 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 $c = 0$: $\Omega_0(L) = \widehat{\Pi}(00, 0) = (\epsilon; (0, 00))$, $\Omega_1(Y) = \widehat{\Pi}(1001, 100) = (\epsilon; (1, 11), (0, 0110), (0, 001100))$, 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 $\widehat{\Pi}(00, 01100)$ 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, every nonreset update obtains a fresh occurrence record, its complete direct/reset rope table, and the updated live context from old receipts by a fixed number of head, tail, singleton, inject, and catenation operations. On reset, only the live rope is updated and the occurrence root becomes the fixed empty record.*

This is closure under sequence operations. Section 11 supplies the interface needed to implement those operations with bounded records.

Proof. 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 $(b, H)\mathcal{T}$.

1. *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 $a = b$.
2. *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 $H = \lambda(bLb)$.

3. *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 .

4. *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). The bridge transition receipt is the suffix-link receipt L ; every off-bridge direct transition receipt is copied from the corresponding entry of the record for L .

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 , either $\ell = \epsilon$ or its first letter differs from a , 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 $\widehat{\Pi}(11, 001)$. 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 $\widehat{\Pi}(1001, 1000)$ 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.

III Turning bounded records into a scaffold

9 The cyclic allocation obstacle

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.

Strict sequential allocation permits references only to old records or previously allocated records. A simultaneous batch instead specifies a finite adjacency table whose edges may target any fresh slot.

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.

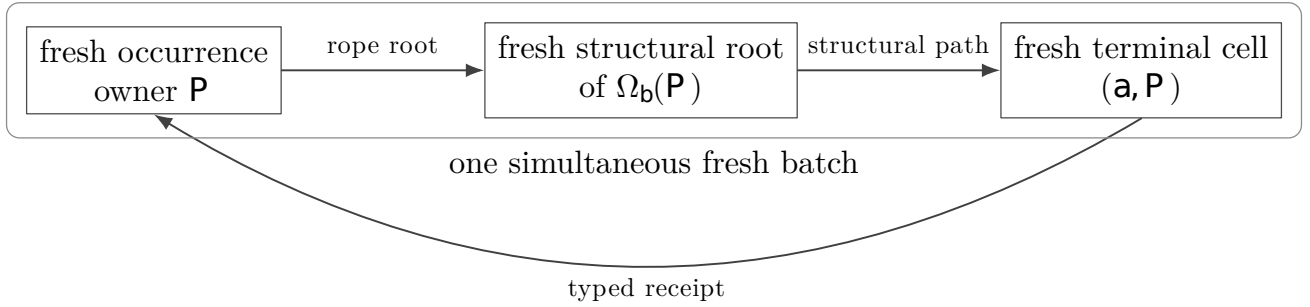


Figure 9. The owner/rope same-step cycle. The steque's structural records are acyclic when the payload P is treated as an opaque atom. Substitution of the fresh owner pointer creates the cycle only at the batch boundary.

The batch is a finite labelled graph. Its specification requires no recursive evaluation of the owner/rope cycle.

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:

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

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 or point into a fixed finite, immutable, pointer-closed initial record

graph. The compiled decoder stores that graph and its initial roots as finite table data; those virtual records are present from time zero and are never charged to a fresh batch.

10.2 Tuple packing

Pack the at most A logical records into finite slots of one physical node. The node label records the active slots, their tags, and the target-slot component of each pointer.

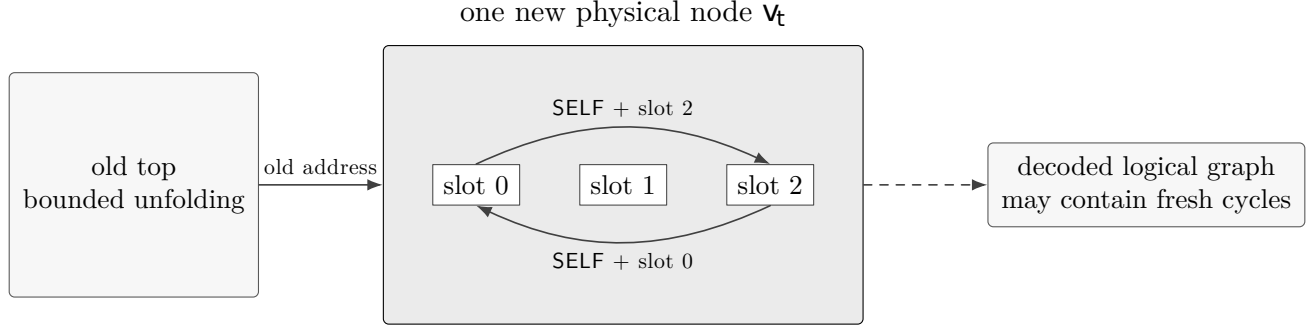


Figure 10. Tuple packing. Several logical fresh records occupy finite slots of one physical scaffold node. Every intra-batch edge has the same physical target **SELF**; its finite tag selects the logical slot.

Theorem 10.2 (Simultaneous batch compiler). *Every transducer of Definition 10.1 whose initial record graph is fixed, finite, immutable, and pointer-closed compiles exactly and letter-synchronously into a finite scaffolding automaton of degree*

$$d = \max\{1, s + Aq\}$$

and observation/update distance at most $r + 1$. The source transition uses only bounded labelled rooted unfoldings and does not branch on pointer identity; acceptance is in finite control.

Proof. Name the fixed initial records $v_0, \dots, v_{H-1}$. Store their labels, fields, and initial roots in a finite decoder table. A logical pointer is missing, virtual (an index in this table), or dynamic (a physical node and slot). Virtual pointers need no scaffold edge.

At time zero an initialization flag selects the fixed initial roots and acceptance value. The first transition reads the table, creates only its at-most- A dynamic records, and clears the flag. At later times a dynamic pointer uses the root or field port below. A virtual pointer is followed entirely inside the table: pointer-closure prevents a path entering it from returning to later dynamic records.

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.

A path that stays dynamic uses one root edge and one edge per field dereference. If it enters the virtual table, its remaining suffix is computed there; the physical descriptor is ignored for a virtual target.

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 virtual-initial-record 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. $\square$

A self-loop, forward reference, or mutual cycle in the fresh batch uses the same physical **SELF** target with different finite slot tags.

Remark 32 (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

The receipt update needs persistent sequences with the following list semantics and source-record properties.

11.1 The sequence interface

Let $\text{seq}(q)$ denote the sequence of token occurrences stored by a root q . The operations satisfy:

$$\text{seq}(\text{empty}) = []; \quad \text{seq}(\text{single}(x)) = [x]; \quad \text{seq}(\text{cat}(q, t)) = \text{seq}(q) \text{seq}(t).$$

The operation $\text{uncons}(q)$ returns **None** exactly on the empty sequence; otherwise it returns its exact first token and a root for the remaining sequence. All operations preserve every old version, including when their inputs share structure.

In addition, the implementation is strict and immutable, uses a finite record signature and finite inspected data, and has a uniform bound on primitive evaluation steps. It copies and returns base tokens without inspecting them or testing pointer identity. Each old pointer comes from an input root by field selection or is copied from known fresh structure. Temporary control values do not escape. These properties are the *sequence contract* used in Theorem 12.2.

11.2 The inherited theorem and its model

Kaplan and Tarjan's theorem motivates this interface. Applying it here also requires the source-record properties just stated.

Theorem 11.1 (Kaplan–Tarjan [6, Theorem 5.1, p. 592]; source scope). *On regular steques, push, pop, inject, and catenation have worst-case constant cost and return regular steques; catenation assumes two regular inputs.*

Regularity is the invariant of their steque representation. Their purely functional model uses pair construction and projection, without memoization [6, p. 578].

Recursive pairs and buffers are structural data that an operation may inspect. Base tokens are opaque. In this application a token is the complete receipt cell, whose receipt field may eventually point to the fresh owner.

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.

A symbolic element may denote a designated fresh payload record, and a fresh owner may store a fixed tuple of resulting roots. Adding those payload and owner slots to the structural skeleton forms one bounded simultaneous batch. Substitution is sound even when a payload field points back to the owner and creates a cycle.

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

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

Now add the bounded fresh payload slots and the owner slot, replace each designated symbol by its payload slot, and fill the payload's receipt field with the owner slot where required. The owner stores its fixed tuple of result roots. The resulting cycle crosses from the owner to a structural root, then to an opaque payload cell, and returns to the owner. Structural evaluation never follows the payload field, so neither the sequence denotation nor the resource bound changes. Theorem 10.2 compiles the whole simultaneous batch, including that feedback edge, with `SELF`. $\square$

Implementation. Appendix D identifies the length-free four-operation source variant and the semantic and resource arguments used for it. The normalization theorem below states the source-record assumptions independently of that implementation.

Theorem 11.3 (Whole-client bounded symbolic normalization). *Fix a finite immutable record signature. Let one deterministic strict client transition have a uniform bound on calls, constructor tests, loops, recursion, and retained fresh records. Suppose it observes only finite tags, finite nonpointer data, old records reached by bounded rooted addresses, and constructors of ordinary fresh structural records already known to its symbolic evaluation. It neither observes pointer identity or allocation state nor inspects an unresolved opaque atom, designated fresh payload, or designated fresh owner. Suppose also that every old pointer later read or retained has a bounded rooted provenance, temporary control packages do not escape, and unreachable failure leaves are completed by a fixed finite plan. Then the whole client normalizes to one bounded simultaneous record-transducer plan determined only by finite control, the input letter, and a bounded labelled unfolding of the old roots.*

Proof. Represent every pointer during the step as missing, virtual, an old root followed by a field word, a fresh structural slot, or an opaque atom. Symbolically execute the bounded client. Inspecting an old pointer reads the corresponding node of the bounded old unfolding. Inspecting a fresh structural pointer reads the constructor and finite fields already stored for that symbolic slot, so every apparent same-step structural test is partially evaluated while the finite transition table is defined. Inspecting an unresolved atom or a designated fresh payload or owner is excluded. Because the program does not test identity, two old addresses may alias without changing a branch.

Strict structural allocation adds fixed-arity symbolic records. Transition-local options, bounded vectors, products, GADT witnesses, and result packages are eliminated by substituting their pointer expressions at their uses; they need no persistent slot because they do not escape. After structural evaluation, add the designated payload and owner slots and perform the opaque-atom substitution. This constructs a finite adjacency table rather than running more client code, so it may introduce self-loops, forward references, or mutual cycles without creating a new observation.

The old-provenance hypothesis translates every old target into a permitted bounded address. The allocation and width hypotheses bound the fresh table. There are finitely many control states, input letters, and labelled old unfoldings of the fixed depth, so the symbolic results form a finite transition table. Unreachable assertion and partial-result leaves receive the fixed rejecting plan and do not affect the run from the valid initial state. Thus the sequential client and simultaneous plan have the same retained roots, record graph, control, and acceptance bit, up to fresh-slot renaming. $\square$

Finite-access lemma. Suppose a client executes at most K primitive steps. Pointers start at old roots or fixed virtual records; primitives copy pointers, select fields, and construct fixed-arity records. Then every old pointer

inspected or retained has a bounded address from an original root.

Proof. Give an old root level zero. Copying preserves its level; selecting a field of an old record increases it by one. Selecting a field of a known fresh record substitutes the pointer expression stored there, introducing no old edge. If a primitive performs several selections, expand it into individual selections. Induction over the at most K steps bounds every old expression by K . Virtual records are resolved in their fixed table. Retaining an old subtree needs only its root expression, not a traversal of all its descendants. The argument therefore bounds both reads and exposed old pointers, including those passed between calls. $\square$

For a concrete program, a finite acyclic call graph with bounded vector folds supplies such a K after each called primitive is expanded. A finite source file alone would not: recursive calls, unbounded loops, or unbounded primitives must be excluded.

Pointer-identity tests, inspection of unresolved payload atoms, and unbounded traversal are excluded by the hypotheses. Each would invalidate the symbolic evaluation argument.

In the palindrome client, the external acceptance test inspects the structural tail returned by the first *uncons*; later catenations also consume earlier results. These are known fresh structural values during symbolic evaluation. The newly designated receipt cells and owner are not inspected. The external and internal branches use at most eight sequence calls each; reset uses two (Appendix B). The sequence contract and the finite-access lemma thus give a uniform bound for the complete client.

12 A direct palindrome scaffold

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 whether the current palindrome is empty and the acceptance bit determined by whether the live rope has empty body.

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 belongs to the fixed virtual initial graph and is decoded from time zero. A *rope value* is a header receipt together with a steque root whose elements are cells $(c, \text{receipt})$.

For the explicit source-resource certificate, this logical layout has three physical record shapes. An occurrence owner has seven direct pointer fields: suffix; transition, header, and body for entry 0; and transition, header, and body for entry 1. Its bridge and two direct/reset tags are finite payloads; a reset tag guards a missing transition pointer. A live-rope record has header and body fields. A cell record has a finite letter and one receipt pointer, and its pointer is the opaque element stored by the sequence. Table entries and their rope pairs are therefore flattened into the owner rather than separately boxed. The two roots remain the owner and live-rope records.

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 first cell and the a -entry of the current record. The old live-rope header is not read. 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:

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.

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.

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. Theorem 11.2 stages each opaque sequence operation, and Theorem 11.3 symbolically composes the complete eight/eight/two client into one old-dependent simultaneous plan. 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). After **00011001**, the new owner represents **1001**. Its suffix receipt points to the virtual empty record. The owner stores the header and body pointers of its two table ropes directly; the live rope has a separate root record. A fresh terminal receipt cell in the bridge rope points back to this owner.

Assign the owner, live-rope record, new cells, and sequence records distinct slots of the same new scaffold node. Each reference between them becomes a **SELF** edge with its target-slot tag. References to existing receipts and shared sequence tails keep their old targets. This realizes the owner/rope cycle while leaving every older scaffold node unchanged.

12.3 Uniform constants

Fix a sequence implementation satisfying Section 11.1. The client schedules are finite, so the finite-access lemma and Theorem 11.3 give constants A_*, q, R_* for allocation, record width, and old-address access, independent of input length. The initial empty owner and ropes are virtual records.

With two current roots, the compiler gives

$$d = \max\{1, 2 + A_*q\}, \quad k = R_* + 1.$$

The argument uses existence of these constants. Appendix B records the implementation's allocation inventory; no numerical radius is used in the theorem.

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 construction). *For a persistent sequence implementation satisfying the contract of Section 11.1, the receipt construction recognizes the binary even-palindrome language with a finite scaffolding automaton. If the complete client has source-record bounds (A_*, q, R_*) , it has degree $\max\{1, 2 + A_*q\}$ and distance at most $R_* + 1$.*

Proof. The fixed virtual empty-owner graph satisfies the run invariant. On each input letter, Theorem 5.4 selects the exact word update and Theorem 8.9 supplies its occurrence record, table entries, and receipt rope. The sequence laws preserve those denotations. Opacity permits the fresh receipt cells to remain symbolic under Theorem 11.2. The finite client schedule and finite-access lemma meet Theorem 11.3, including the external

tail-emptiness test, and produce one bounded simultaneous plan. Theorem 10.2 encodes it in one scaffold node. The context is empty exactly for an even palindrome, which is the acceptance bit in finite control. Induction on input length proves the claim. $\square$

Corollary 12.3 (Direct PEG route). *Under the sequence contract in Theorem 12.2, the construction yields a PEG for EPAL_2 .*

Proof. Loff–Moreira–Reis Theorem 2.7 sends the scaffold language of Theorem 12.2 to a complete-match PEG for its reversal. The reversal of ww^R is again of the form uu^R , so $\text{EPAL}_2^R = \text{EPAL}_2$. $\square$

13 Relation to the transfer construction

The earlier transfer paper [5] obtains palindrome membership from a classical real-time recognizer. The present construction exposes the stored state: occurrence receipts, anchored ropes, and a finite simultaneous batch. Its representation counterexamples concern only the specific interfaces in Section 7.

The compiler applies to any bounded simultaneous record update. The palindrome application additionally uses receipt closure and the sequence contract. Both routes use the Loff–Moreira–Reis correspondence to obtain a PEG.

14 Supporting evidence

The receipt, compiler, and normalization arguments are written proofs. Bounded word and graph checks test the recurrence, shadow transport, receipt closure, and batch decoding. Source audits address the separate sequence contract and resource accounting. The [proof and evidence index](#) links those records.

This work was developed with AI assistance. The recorded AI reviews share model dependencies; no external specialist review is recorded.

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.

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

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

B Client operation inventory

The complete schedules include observations used for branching and acceptance:

Branch	Sequence calls	Allocation expression
External	3 uncons, 2 single, 3 cat	$3A_U + 2A_S + 3A_C + \Lambda_E$
Internal	2 uncons, 2 single, 4 cat	$2A_U + 2A_S + 4A_C + \Lambda_I$
Reset	1 uncons, 1 cat	$A_U + A_C + \Lambda_R$

Here A_U, A_S, A_C bound the respective operations; the Λ terms count client records in the chosen layout. The flattened layout has a seven-pointer owner, a two-pointer live-rope wrapper, and one-pointer cells. The archived [fixed-client allocation audit](#) reports $A_* = 8838$, $q = 7$. These implementation-specific ceilings are not needed for the qualitative theorem.

C Supporting files

The [evidence index](#) separates the word recurrence, receipts, compiler, sequence implementation, and assembled client. The [typed-call inventory](#) checks source bindings and bounded call recurrences. The [constructor-flow analysis](#) records progress toward an automated pointer-provenance proof.

D Sequence implementation

The [length-free four-operation variant](#) adapts the handwritten OCaml implementation of Viennot, Wendling, Guéneau, and Pottier [10], at upstream commit [444358e963db89b7d90a5065c4680ddd70248f82](#). It removes cached-length observations from the relevant interface and uses bounded endpoint probes. The facade exposes `empty`, `single`, `uncons`, and `cat`.

The supporting source arguments cover the deque list equations, the Buffer lower-bound invariant and assertion safety, allocation, and bounded calls. The relevant vector folds have at most six elements. Recursive sequence-wide folds elsewhere in the module are outside this facade's operation paths. The finite-access lemma applies to the bounded primitive evaluation, irrespective of the depth of recursively paired data.

These are informal source-level arguments for the contract in Section 11.1. The automated analysis does not yet establish complete old/fresh pointer provenance or a numerical access radius. The separate Rocq development proves properties of its own implementation; no checked correspondence to this handwritten source is used here. The implementation evidence and the generic conditional construction should therefore be read as distinct parts of the argument.

Acknowledgment of inherited work

Loff, Moreira, and Reis supply the scaffold/PEG correspondence; Kaplan and Tarjan supply the real-time catenable-sequence result; Viennot and coauthors supply the modern implementations. The receipt construction, simultaneous-record compiler, and normalization argument are the contributions developed here.

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. [FPRD reading edition](#).

- [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>.

Revision history

Draft 5, 21 September 2026. Consolidated the web and PDF editions; integrated virtual initialization into the compiler proof; stated the sequence contract in the direct construction; replaced the numerical distance claim with symbolic bounds; and shortened repeated exposition.

10 September 2026 corrections. The initial fixed graph is encoded in a finite decoder table. The six-call formulas $A = 6K + 4$, $r = 6K + 8$ were replaced by complete client schedules. The later proposed radii 55/874, client radius 1750, and scaffold distance 1751 remain withdrawn because the checker did not establish source-path coverage or pointer provenance.

[Archived Draft 4 PDF](#) · [Historical radius audit](#)