Proof Concept: Active-Route Collatz Proof Package

The Collatz conjecture looks impossible because an escaping orbit would not need to be large, random, or dramatic. It would only need one unaccounted-for mechanism: one branch that keeps avoiding every known trap forever.

This package makes a deliberately narrow claim about that escape route:

`there is no sixth place for it to go.`

Why This Package Is Different

Most attempted Collatz proofs fail at the same point: they explain why the orbit usually goes down, or why many families close, but leave room for one unseen infinite mechanism to escape.

This package is built to attack exactly that weak point. The proof does not ask a reviewer to trust a global trend, a probabilistic heuristic, or one large monotonicity claim. Instead, it turns the remaining obstruction into a finite audited stock problem:

`every live branch must enter known stock, enter a finite splice ledger, terminate, or descend.`

The central question is therefore not "does the orbit seem to shrink?" The central question is sharper:

`is there any branch left outside the five audited bridge targets?`

The archive is organized so that a reviewer can try to break precisely that claim. If one bridge target fails, the proof fails. If all five bridge targets hold, then no autonomous infinite mechanism remains; the minimal-counterexample closeout has nowhere left to hide, and the Collatz theorem is solved on the assembled route.

Purpose And Status

This document is a readable guide to the active-route proof package. It is not a new proof carrier and it does not replace the formal files in the archive.

The formal route files remain:

- `ACTIVE/foundation/active_route_proof_stack.md`;
- `ACTIVE/review/dependency_matrix.md`;
- the review modules and imports under `ACTIVE/review/`;
- the theorem-facing lemma files under `ACTIVE/proved_lemmas/`;
- the finite audit scripts and JSON artifacts where explicitly cited.

The goal here is simple: explain how the proof is built, why the five bridge targets are the right checkpoints, and how the local algebra, finite stock, and standard-side descent fit together.

Reader's Vocabulary

The package uses a few repeated technical words. Read them in this practical sense first:

- `stock` means a finite or already accepted set of known behaviors.
- `splice` means a controlled transition from one branch family into known stock or into another finite ledger.
- `atlas` means a named menu of old continuation patterns that have already been audited.
- `residual branch` means a branch that has survived the earlier filters and still needs to be classified.
- `continuation object` means the canonical future behavior attached to a fixed entrance state.

In informal terms, the proof keeps asking the same question:

```
is this branch genuinely new, or has it already entered known behavior?
```

1. The Main Strategy

The Collatz theorem is proved by the standard minimal-counterexample plan.

It is enough to prove the following descent statement:

```
Every positive odd integer n > 1 reaches either 1
or a strictly smaller positive odd integer after finitely many odd Collatz steps.
```

If this holds, a smallest positive odd counterexample cannot exist. It would either reach `1`, which is not a counterexample, or reach a smaller positive odd counterexample, contradicting minimality. Even positive integers reduce to odd integers by division by powers of two, so the ordinary positive-integer Collatz theorem follows.

The proof package therefore has one central job:

```
force every theorem-relevant trajectory into finite audited stock
or into a strict descent episode.
```

The proof is not organized as one short formula. It is organized as a finite stock system. Every live branch must either terminate, enter known finite stock, enter an audited splice package, or descend.

The review package reduces the global claim to five explicit bridge targets:

1. global entry / Lemma 1a;
2. finite launch menu;

3. finite splice menu;
4. bounded-tail splice;
5. `(R, lambda)` determinism / finiteness / stock splice.

The package is complete if and only if these five bridge targets are correct.

2. Two Coordinate Systems

There are two coordinate systems in the package.

The ordinary Collatz odd map is

```
T_+(n) = (3n + 1) / 2^v2(3n+1).
```

The active core-side proof often works in the `q`-coordinate with

```
T_-(q) = (3q - 1) / 2^v2(3q-1).
```

The bridge between them is negation:

```
q = -n.
```

The map `J(x) = -x` conjugates the two odd maps:

```
T_-(-n) = -T_+(n).
```

This means valuation words and valuation cylinders transfer exactly between the two pictures.

So statements that only depend on exact valuation words can be moved across the bridge.

The proof package is careful about what does not transfer automatically. The post-break quantity

```
v2(T_-(q) + 1)
```

does not become

```
v2(T_+(n) + 1)
```

under negation. Under `q = -n`, it becomes a quantity involving `1 - T_+(n)`.

Therefore the final strict descent statement is not read from negation alone. The final bridge uses:

```
q-coordinate stock closure
+ valuation/cylinder transfer
+ standard-side re-entry and strict-descent stock
=> ordinary positive-odd descent.
```

This prevents the common error:

```
q-coordinate closure => standard Collatz descent
```

without a separate standard-side bridge.

3. Global Entry: Lemma 1a

The first major task is to prevent an arbitrary live branch from producing uncontrolled new behavior. This is the role of Lemma 1a.

The active Lemma 1a route is:

```
Official A/B/C/D
-> Reachability
-> Identification
-> Input A
-> Input B
-> Lemma 1a
```

Official A: finite live projection

A theorem-level active live branch is projected to a refined finite state

`K = (C, t mod 2, p),`

Here `C` is an exact centered coordinate, `t mod 2` records a parity datum, and `p` is the survivor pair.

The exact centered state is used because the old coarser projection was not enough to determine the next live behavior.

The important point is finiteness:

`0 <= C < 3^17.`

Thus a live branch that stayed live forever would give an infinite walk in a finite reachable graph.

Official B and B.1: finite graph check for terminal continuations

An infinite walk in the finite graph must eventually enter a recurrent class. Each terminal recurrent class is then checked by a finite directed graph of its terminal continuation states.

There is an edge `x -> y` in this continuation graph exactly when applying the specified terminal transition rule of type `D` to `x` produces `y`. Outcomes of other terminal transition types are checked separately and leave the type-`D` graph into one of the recorded outcomes:

`E, 11, 21, bounded, exit.`

The checked claim is that no directed cycle in this continuation graph is reachable from the states induced by the original terminal recurrent class. Since every infinite path in a finite directed graph must eventually repeat a vertex and hence contain a directed cycle, this rules out an infinite sequence using only type-`D` terminal transitions.

Any terminal `E` leaf returns to a finite set `R`. Every element of `R` reaches a finite funnel `F`, and `F` collapses to `1`.

The funnel is:

`F = {1, 3, 15, 39, 6651, 70939, 378341}.`

The displayed chain is:

```
378341 -> 70939 -> 6651 -> 39 -> 15 -> 3 -> 1
70939  -> 6651  -> 39 -> 15 -> 3 -> 1
6651   -> 39    -> 15 -> 3 -> 1
39     -> 15    -> 3 -> 1
15     -> 3     -> 1
3      -> 1
1      -> 1
```

The finite returned-coordinate stock is recorded in `ACTIVE/scripts/terminal_r_certificate.json`, rebuilt by `ACTIVE/scripts/build_terminal_r_certificate.py`.

Official C: no terminal finite-cycle obstruction

If a live branch entered a terminal recurrent class, then the finite directed-graph check above would force it to leave any attempted infinite type-`D` terminal recursion.

It then either returns through `R -> F -> 1` or enters downstream stock.

Therefore a terminal recurrent class cannot be a genuine infinite obstruction.

Reachability

After the finite-cycle obstruction is removed, the remaining theorem-level live branches are routed to either the finite funnel or downstream atlas stock.

The downstream atlas objects are:

```
Q = 1 + 6w
Q = 3 + 6w
(),(3),11
(3,3)
bounded
exit
```

These are not arbitrary names and they are not hidden assumptions.

They are the existing cheap-side continuation stock used on the active line.

The large branch analysis in the lemma stock, especially `break_state_transition_map.md`, shows how local residual branches reduce into these objects rather than creating new continuation mechanisms.

Identification

Reachability says where a live branch lands.

Identification says how to read the landing.

If a branch reaches `F` or one of the atlas objects, then it is already captured by the canonical continuation language

`Sigma(q_*).`

This is the bridge from:

`the branch landed in known stock`

to:

`the branch has no new future behavior left to introduce.`

Input A and Input B

Input A is the global entry statement:

```
Every theorem-level live residual branch that does not close earlier
reaches one canonical continuation object Sigma(q_*)
on one exact residual entrance fiber q_in = q_*.
```

Input B is the fixed-fiber arithmetic statement.

Once the entrance fiber `q_in = q_*` is fixed, the forward odd dynamics is deterministic.

Raw relaunch history does not create a second canonical object on the same fixed fiber.

Thus:

`Input A + Input B => Lemma 1a.`

Lemma 1a is the global cheap-side entry theorem used downstream by the core-side closure theorem.

4. Why The Atlas Stock Is The Right Stock

The atlas stock is the list of standard outputs that can occur when the active branch is no longer producing a new live mechanism.

`bounded` means the branch has entered a finite small kernel.

`exit` means the branch has left the active dangerous/live corridor.

`Q = 1 + 6w` and `Q = 3 + 6w` are base-template families. When a residual tail normalizes to one of these forms, it has returned to existing stock rather than creating a new family.

`(),(3),11` and `(3,3)` are recursive atlas objects.

They represent previously audited continuation patterns. Later continuation-side and Appendix E branches can land on these old ladders without creating an autonomous new mechanism.

The proof work is therefore not merely to name these objects. The proof work is to show that all theorem-relevant live branches either enter the finite funnel or reduce to one of these stock objects. This is what the reachability and downstream stock carriers provide.

5. Core-Side Global Closure

Lemma 1a gives global entry on the active cheap-side line. The core-side theorem then closes the admissible-state burden after the interface.

The core-side theorem imports:

1. Lemma 1a;
2. terminal stock and `R -> F -> 1`;
3. Appendix C feeder splice;
4. Appendix D suffix pullback;
5. Appendix E `3939` finite package;
6. Official Corollary E.2.

The conclusion is:

```
Every theorem-relevant core-side branch either closes before
the admissible-state interface, or reaches that interface and is then
absorbed by finite audited stock/splice data.
```

This is the core no-escape statement.

6. Official Corollary E.2

Official Corollary E.2 closes the core-side admissible-state burden. It has three clauses.

Terminal-side clause

The terminal-side contribution is reduced to terminal splice inputs:

```
(L_term,1,1)
(L_term,2,1)
```

Appendix C handles the `(L_term,1,1)` feeder-splice route. Appendix D handles the `(L_term,2,1)` suffix-pullback route. Therefore terminal-side contributions enter the common finite audited terminal menu.

Continuation-side clause

The continuation-side contribution preserves global stock-splice closure. The large-depth branch lands in pre-existing stock, the audited small-depth menu lands in pre-existing stock except for one outsider, and the outsider is reduced to the corrected 3939 finite package on the old (3,3) ladder.

Thus the continuation side contributes only finitely many explicit splice states and no autonomous active mechanism.

Feeder-side clause

The feeder-side burden is closed by the Appendix C finite splice landing. The central corridor is:

```
125 (mod 512) -> 486w + 121 -> 13122t + 1333 -> finite stock.
```

Sibling channels either close by finite exact splice or enter accepted recursive templates such as Q = 1 + 6u and Q = 3 + 6u.

Putting the three clauses together:

```
terminal-side stock
+ continuation-side finite package
+ feeder-side finite splice
=> every admissible (R, lambda) contribution lands in finite stock/splice.
```

This is bridge target 5.

7. The Main Splice Corridor

The most visible active-route splice is:

```
13-bank -> 125 (mod 512) -> 486w + 121 -> 13122t + 1333 -> finite closure.
```

The 13-bank is a recurrence-forced region. It is not enough to know that a branch enters the broad class 13 (mod 16). The proof uses an exact quarter-selection inside that bank.

The active branch hits a unique pullback quarter

```
q   a_L (mod 1024),
```

whose next odd state lies in

```
125 (mod 512).
```

From there, the exact feeder route gives:

```
125 (mod 512) -> 486w + 121.
```

Inside that feeder family, the channel

```
w   3 (mod 32)
```

lands on

```
13122t + 1333.
```

The Appendix C item 4 carrier and its ledger verifier show that this feeder package lands in finite stock or already accepted stock. The important point is that the 13-bank is not used as a vague negative-gap heuristic. It is used through an exact selected subquarter that enters a finite splice route.

8. Appendix D And Appendix E

Appendix D handles the suffix-pullback side, especially the `(L_term,2,1)` terminal splice input. Its carrier is the theorem file `uniform_mod64_entry_to_mod16_class_9_on_branch_21.md`. The key effect is that the branch lands in the locked negative-gap class `9 (mod 16)` under the exact branch law.

Appendix E handles the corrected `3939` finite package. The third-step theorem for

```
q_in = 32611 + 4096t
```

splits into bounded cases, exact exit families, exact danger families, and one residual family:

```
t = 16c + 11,
q_in = 77667 + 65536c.
```

This residual family is already inside the corrected `3939` stock:

```
77667 + 65536c = 3939 + 4096(18 + 16c).
```

The local ledger records a finite active-state menu:

```
[0,4]
[0,1,5]
[1,5]
[3,7]
[0,2,6]
[0]
```

with the active chain:

```
[0,4] -> [0,1,5] -> [1,5] -> [3,7] -> [0,2,6] -> [0].
```

The script `ACTIVE/scripts/audit_3939_bifurcation.py` is an audit support artifact for the displayed Appendix E arithmetic. It does not replace the theorem carrier or the local transition ledger.

9. Standard-Side Strict Descent

The core-side closure is not by itself the final Collatz theorem. The final bridge must return to the ordinary positive odd Collatz map.

The standard-side descent package supplies:

1. finite re-entry from outside phases to the low-valued section;
2. exact low-valued transition grammar;
3. valuation/cylinder transfer through negation conjugacy;
4. exact path families feeding low-valued obstruction stock;
5. a finite `L = 1` one-step partition with contraction on non-stock alternatives;
6. the recurrence-forced `13`-bank quarter-selection into Appendix C;
7. the assembled core-side stock/splice closeout.

Thus, after a standard positive odd orbit re-enters the low-valued section, the active route has only the following outcomes:

```
direct terminal stock or audited sink;
finite corridor into the 13-bank quarter-selection and Appendix C;
finite splice into imported stock;
strict descent of the active odd/break-state parameter.
```

Each outcome is read as a finite continuation to `1` or to a positive odd integer strictly smaller than the starting odd integer for the current descent episode.

The standard-side bridge used by the package is the descent statement:

```
Every positive odd n > 1 reaches 1
or reaches a smaller positive odd integer.
```

The minimal-counterexample argument then gives the Collatz theorem.

10. Where Scripts Enter

The scripts and JSON files are finite audit artifacts. They do not replace the theorem-facing proof carriers.

Examples:

- `build_terminal_r_certificate.py` rebuilds the terminal `R -> F -> 1` certificate.
- `terminal_r_certificate.json` records the finite returned-coordinate stock.
- `scan_128u85_closure.py` checks the Appendix C feeder ledger.
- `audit_3939_bifurcation.py` checks displayed Appendix E arithmetic.

The intended rule is:

```
The theorem file carries the mathematical statement.
The script checks bounded finite ledgers or exact finite artifacts.
```

If a rebuilt artifact ever disagrees with the review text, the relevant import must be treated as failed until the text and artifact are reconciled.

11. What Would Refute The Package

A reviewer does not need to read the archive as an undifferentiated mass. It is enough to break one bridge target.

Possible refutations include:

1. a live branch that does not enter `F`, atlas stock, terminal stock, Appendix C/D/E stock, or standard-side descent;
2. a failure of the finite `K = (C, t mod 2, p)` representation to determine the needed future stock behavior;
3. a reachable terminal continuation graph with a directed cycle using only type-`D` terminal transitions;
4. an admissible `(R, lambda)` contribution not absorbed by the E.2 stock/splice menu;
5. a standard-side orbit where the claimed active-route outcome does not produce `1` or a smaller positive odd integer.

Conversely, if the five bridge targets withstand review, then the package establishes:

```
every positive odd integer reaches 1 or a smaller positive odd integer,
```

and the minimal-counterexample closeout gives the ordinary Collatz theorem.

12. Reading This With The Full Package

This document is best read before the detailed files. The recommended path is:

1. `Reviewer_Overview.pdf`;
2. this `Proof_Concept.md`;
3. `ACTIVE/review/README_FOR_REVIEW.md`;
4. `ACTIVE/review/dependency_matrix.md`;
5. `ACTIVE/review/final_review_manuscript.md`;
6. the five bridge target carriers named in the dependency matrix.

The short version is:

```
Lemma 1a gives global entry.
Terminal stock, atlas stock, and Appendices C/D/E absorb the core side.
Standard-side re-entry and strict descent return the result to ordinary Collatz.
Minimal counterexample closes the theorem.
```