MATHEMATICAL SUPPLEMENT UNIVERSAL COMPUTATION WITH 144 RESIDUE CLASSES

MICHAEL SCHROEDER

This supplement accompanies *Universal Computation With 144 Residue Classes: A One-State Linear Operator Algorithm.*

Release `oloa144-2026-09-28.1` supplies the complete finite inventories and the additional proofs and execution records cited in the main paper. The 144 numerical rules are printed in the main paper's Appendix A and supplied as `machine_v8/machine_144.csv` in the canonical data archive.

Section, theorem, and equation numbers beginning with S refer to this supplement; unprefixed numbers refer to the main paper. The same numbers are used in the combined reading edition. The scope of the claims and formalization is stated once, at the end of the main paper's introduction.

S1. Historical definitions and related models

S1.1. **Kaščák's basic definitions.** Kaščák reported a universal one-state linear operator algorithm of modulus 396 in 1992 [6]. His Definitions 2.2–2.3 on p. 328 specify triples in $\mathbb{N} \times \mathbb{Z} \times \mathbb{N}$, exact transitions $y = (ax + b)/c$ between natural numbers for $c \neq 0$, and final numbers selected by $c = 0$. Definition 2.1 calls a unary partial recursive function u universal when there is "a binary recursive function c such that" $F_e(x) = u(c(e, x))$ for all e, x. In particular, that definition imposes no primitive-recursive or polynomial growth restriction on c.

Lemma S1.1 (compatibility with the basic definitions)**.** *The displayed table is an OLOA under Kaščák's Definitions 2.2–2.3, with an exact natural successor on every continuing progression. The decoded partial function $u = d_0 \circ F$ satisfies his Definition 2.1, using the encoding c of Theorem 1.1.*

Proof. The coefficient types and halt convention agree term by term. Lemma 6.1 proves the stronger whole-progression validity condition. Iterating the effective table until a halt computes the partial function F; composing with the total computable d_0 gives a partial recursive u. The identity and domain equality in Theorem 1.1(iii) are precisely $F_e(x) = u(c(e, x))$ with total computable c. □

Date: 28 September 2026.

2020 *Mathematics Subject Classification.* Primary 03D10; Secondary 68Q04, 68V20.

DOI: 10.5281/zenodo.23018009.

Author ORCID: 0009-0004-3249-0195.

S1.2. **Related models and decision problems.** The terminology overlaps with generalized Collatz functions and residue-class-wise affine mappings. In the latter setting, studied by Kohl [8], a map on $\mathbb{Z}$ is rational affine on each class of a finite congruence partition. Here the state space is $\mathbb{N}$, and designated residues stop the computation. The requirement of nonnegative successors is imposed on each entire continuing progression.

FRACTRAN provides a useful comparison between two size measures. Write its ordered positive fractions in reduced form as p_i/q_i, and put $m = \operatorname{lcm}(q_1, \ldots, q_k)$. The first applicable fraction depends only on $n \bmod m$, because applicability means $q_i \mid n$. Assigning the corresponding multiplication to each residue, with halt rows where none applies, gives a residue table. This standard conversion [3, 5] shows why the number of fractions and the number of residue classes are different measures: a short fraction list can expand into a large table.

The input to an undecidability problem also matters. Kurtz and Simon prove Π^0_2-completeness for the generalized Collatz question asking whether every positive initial value eventually reaches 1, with the map supplied as input [9]. Endrullis, Grabmayer, and Hendriks obtain Π^0_2-completeness for termination of a FRACTRAN program on every positive integer [5]. Ben-Amram proves that the generalized Collatz result persists for some fixed modulus [1, Theorem 9]; the coefficients still vary with the instance. In Theorem 1.1, the entire table is fixed and the input is one starting integer. Its halting set is Σ^0_1-complete, equivalently computably enumerable complete.

Other small descriptions permit different branch tests or simulate a particular arithmetic map. Lehtonen obtains an undecidable reachability problem using the formulas $n/2$ and $3n + t$, $-9 \le t \le 9$, selected by recursive sets [10]. Those sets are not required to be congruence classes for one fixed modulus. De Mol represents the classical Collatz iteration by a two-tag system with three symbols [4]; Knight gives a multiplication-only rule K with 30 conditions and explicitly introduces a modulus-60 variant K' in Section 4 [7, p. 6]. These are simulations of the classical Collatz map, rather than proofs of functional universality for all partial recursive functions. Their description sizes therefore do not supply a smaller universal OLOA.

Termination analysis supplies a further, distinct context. Yolcu, Aaronson, and Heule express the Collatz conjecture as termination of a string-rewriting system and prove termination for weakened variants using automated methods [12]. Carelli studies one-variable integer loops whose transition relation is one convex polyhedron, obtaining a polynomial-time termination result conditional on a generalized Collatz conjecture [2]. That restriction differs from a union of residue-selected affine transitions. Neither result supplies a termination procedure for the universal table studied here.

S2. Complete source interpreter and compressed paths

The tuples below use the I, D, T convention defined in the main text. A missing fourth argument in I, D is intentional. Label zero is the sole source halt.

Label	Instruction	Label	Instruction
1	$T(5, 2, 10)$	22	$D(6, 21)$
2	$D(5, 3)$	23	$T(1, 24, 26)$
3	$T(5, 4, 12)$	24	$D(1, 25)$
4	$D(5, 5)$	25	$I(4, 23)$
5	$T(5, 6, 16)$	26	$I(5, 27)$
6	$D(5, 9)$	27	$T(6, 28, 29)$
9	$I(7, 1)$	28	$D(6, 26)$
10	$T(0, 11, 19)$	29	$I(6, 30)$
11	$D(0, 20)$	30	$D(5, 31)$
12	$T(2, 13, 19)$	31	$D(4, 32)$
13	$D(2, 20)$	32	$I(1, 33)$
16	$I(0, 17)$	33	$T(4, 34, 35)$
17	$I(2, 20)$	34	$T(5, 29, 27)$
19	$I(7, 20)$	35	$T(5, 36, 23)$
20	$T(7, 21, 0)$	36	$D(7, 37)$
21	$T(6, 22, 23)$	37	$T(7, 23, 1)$

In the next table each list contains the executed primitive labels, stopping just before the next guarded node. The two lists correspond to divisibility and nondivisibility in the guarded table. Identical lists arise when the primitive operation is a truncated decrement whose two arithmetic cases are represented by different guarded coefficients.

q	Positive path	Zero path	q	Positive path	Zero path
1	1, 2	1	27	27, 28, 26	27
3	3, 4	3	29	29, 30	29, 30
5	5, 6, 9	5, 16, 17	31	31, 32	31, 32
10	10, 11	10, 19	33	33	33
12	12, 13	12, 19	34	34, 29, 30	34
20	20	20	35	35	35
21	21, 22	21	36	36	36
23	23, 24, 25	23, 26	37	37	37

S3. Complete source-mask certificate

Each integer is an eight-bit positivity mask with bit k representing $S_k > 0$. The register S_3 bit is zero throughout. These are overapproximating sets: every abstract successor must be present, but a listed mask need not be concretely reachable. The initial masks are $2, 6$ at label 1.

Label	Allowed masks
1	2, 6, 98, 99, 102, 103, 194, 195, 198, 199, 226, 227, 230, 231
2	98, 99, 102, 103, 226, 227, 230, 231
3	66, 67, 70, 71, 98, 99, 102, 103, 194, 195, 198, 199, 226, 227, 230, 231
4	98, 99, 102, 103, 226, 227, 230, 231
5	66, 67, 70, 71, 98, 99, 102, 103, 194, 195, 198, 199, 226, 227, 230, 231
6	98, 99, 102, 103, 226, 227, 230, 231
9	66, 67, 70, 71, 98, 99, 102, 103, 194, 195, 198, 199, 226, 227, 230, 231
10	2, 6, 194, 195, 198, 199
11	195, 199
12	66, 67, 70, 71, 194, 195, 198, 199
13	70, 71, 198, 199
16	66, 67, 70, 71, 194, 195, 198, 199
17	67, 71, 195, 199
19	2, 6, 66, 67, 194, 195, 198
20	66, 67, 70, 71, 130, 134, 194, 195, 198, 199
21	130, 131, 134, 135, 194, 195, 198, 199
22	194, 195, 198, 199
23	130, 131, 134, 135, 144, 145, 146, 147, 148, 149, 150, 151, 194, 195, 198, 199, 208, 209, 210, 211, 212, 213, 214, 215, 226, 227, 230, 231, 240, 241, 242, 243, 244, 245, 246, 247
24	130, 131, 134, 135, 146, 147, 150, 151, 194, 195, 198, 199, 210, 211, 214, 215, 226, 227, 230, 231, 242, 243, 246, 247
25	128, 129, 130, 131, 132, 133, 134, 135, 144, 145, 146, 147, 148, 149, 150, 151, 192, 193, 194, 195, 196, 197, 198, 199, 208, 209, 210, 211, 212, 213, 214, 215, 224, 225, 226, 227, 228, 229, 230, 231, 240, 241, 242, 243, 244, 245, 246, 247
26	144, 145, 146, 147, 148, 149, 150, 151, 176, 177, 178, 179, 180, 181, 182, 183, 208, 209, 210, 211, 212, 213, 214, 215, 240, 241, 242, 243, 244, 245, 246, 247
27	176, 177, 178, 179, 180, 181, 182, 183, 210, 211, 214, 215, 240, 241, 242, 243, 244, 245, 246, 247
28	210, 211, 214, 215, 240, 241, 242, 243, 244, 245, 246, 247
29	176, 177, 178, 179, 180, 181, 182, 183, 242, 243, 246, 247
30	240, 241, 242, 243, 244, 245, 246, 247
31	208, 209, 210, 211, 212, 213, 214, 215, 240, 241, 242, 243, 244, 245, 246, 247
32	192, 193, 194, 195, 196, 197, 198, 199, 208, 209, 210, 211, 212, 213, 214, 215, 224, 225, 226, 227, 228, 229, 230, 231, 240, 241, 242, 243, 244, 245, 246, 247
33	194, 195, 198, 199, 210, 211, 214, 215, 226, 227, 230, 231, 242, 243, 246, 247
34	210, 211, 214, 215, 242, 243, 246, 247
35	194, 195, 198, 199, 226, 227, 230, 231
36	226, 227, 230, 231
37	98, 99, 102, 103, 226, 227, 230, 231

S4. Complete scalar locations

For each control, (6.1) uses the following scale and offset. Work base is 3 for $MP, R9$ and 9 elsewhere. The last column is the allocated set of residues modulo 144, including both successful-halting cases in $R3$. In particular, a listed control's allocation may contain only a subset of the residues of its scale class, because excluded branches need no rule allocation.

Control	K_s	o_s	Allocated residues
C0	48	0	0, 48, 96
C1	48	1	1, 49, 97
C2	48	2	2, 50, 98
C3	48	4	4, 52, 100
C4	48	3	3, 51, 99
D10:0	9	40	40, 58, 76, 112, 121
D10:4	9	39	39, 57, 75, 111, 120
D10:8	18	37	37, 55, 73, 109
D12:0	9	38	38, 56, 74, 110, 119
D12:12	72	14	14, 86
D12:4	9	46	46, 64, 82, 118, 127
D12:8	9	41	41, 59, 77, 113, 122
D1:0	72	21	21, 93
D21:0	9	36	36, 45, 54, 72, 108, 117
D21:4	18	66	66, 84, 102, 138
D23:0	18	-12	6, 24, 60, 132
D23:12	18	-9	9, 27, 63, 135
D23:15	36	-3	33, 69, 141
D23:3	18	-10	8, 26, 62, 134
D23:6	18	-7	11, 29, 65, 137
D23:9	18	-11	7, 25, 61, 133
D27:0	72	17	17, 89
D27:4	72	79	79
D31:0	72	-30	42, 114
D31:4	72	-8	136
D34:0	72	31	31, 103
D36:0	72	15	15, 87
D3:0	72	13	13, 85
D5:0	72	16	16, 88
M2	72	18	18, 90
M3	72	22	22, 94
MP	72	-2	70, 142
R0	48	-13	35, 83, 131
R1	48	20	20, 68, 116
R10	48	-15	81, 129
R11	48	-1	47, 95, 143
R12	48	34	34, 130
R13	48	23	23, 71
R2	16	28	12, 28, 44, 92, 140

Control	K_s	o_s	Allocated residues
R3	16	107	43, 91, 107, 123, 139
R5	48	5	5, 53, 101
R6	48	-16	32, 80, 128
R7	48	-18	30, 78, 126
R8	48	19	19, 67, 115
R9	48	10	10, 106

S5. Complete arithmetic contracts

For completeness, the arithmetic preconditions needed throughout every routine are given next. All rows include $X, Y \geq 0$ and $W > 0$. In division rows $W = X + pY + \rho$ at phase ρ. In transfer rows W has the displayed form. Implications concern the *integer* W, not the parity of X alone.

Controls	W	Further conditions
$D1, D3, D5$	$X + 2Y$	$34 \mid W$
$D10$	$X + 11Y + \rho$	W odd; $17 \mid W$; $11 \mid W \Rightarrow 3 \mid W$
$D12$	$X + 13Y + \rho$	W odd; $17 \mid W$
$D21$	$X + 7Y + \rho$	$3 \mid W$; odd $W \Rightarrow 17 \mid W$; even $W \Rightarrow 5 \mid W$
$D23$	$X + 17Y + \rho$	$3 \mid W$; $17 \mid W$ or $5 \mid W$
$D27$	$X + 7Y + \rho$	W odd; $105 \mid W$
$D31$	$X + 5Y + \rho$	$105 \mid W$
$D34$	$X + 2Y$	$30 \mid W$
$D36$	$X + 3Y$	$102 \mid W$
$R0, R1$	$X + Y$	$17 \mid W$
$R2$	$X + Y$	$17 \mid W$; $W \bmod 6 \in \{0, 2, 3, 4\}$
$R3$	$X + Y$	odd $W \Rightarrow 17 \mid W$; even $W \Rightarrow 30 \mid W$
$R5$	$X + 3Y$	$51 \mid W$
$R6$	$X + 7Y$	$105 \mid W$
$R7$	$X + 7Y$	same divisibility/parity conditions as $D21$
$R8$	$X + 5Y$	$15 \mid W$
$R9, R12$	$X + 13Y$	W odd; $17 \mid W$
$R10$	$X + 11Y$	W odd; $17 \mid W$
$R11$	$X + 17Y$	$15 \mid W$
$R13$	$X + 2Y$	$30 \mid W$
MP	$11X + Y$	W odd; $17 \mid W$
$M3$	$3X + Y$	W odd; $51 \mid W$
$M2$	$2X + Y$	$30 \mid W$
C_i	$X + 17Y$	$357 \mid W$; $X \equiv 17i \pmod{85}$

These contracts are the literal release specifications. The Lean development first proves equivalent factorized contracts and then establishes their equivalence to this inventory. They apply at internal control points and are stronger than the unrestricted raw-input validity of the final scalar table.

S6. Quantitative properties of the construction

The shared routine reduces execution length as well as the residue budget. A separate conditional coefficient optimum for the five-control ring is proved in Section S9.

The supplied predecessor v6 provides a separate, realized comparison. The publication companion indexes its full source, table, proof, and per-control residue budget in `PREDECESSOR_COMPARISON.md`. Its budget is 47 controls and 143 occupied residues. Its shared division uses three phases with demands $4 + 4 + 2 = 10$. Quotient-parity selection replaces them by two phases with demands $6 + 4 = 10$, and makes the three-residue return control $R4$ unnecessary. Hence this change saves two controls and three occupied residues, without increasing the shared division's residue demand. The earlier source-path compression, fused tests, and phase ring are already present in that predecessor.

S6.1. **Exact scalar-step saving in the shared routine.** Consider a baseline using three division phases $3+3+1$, restoring an exact quotient through an increment-one transfer, and, when the quotient is even, subsequently multiplying it by two and restoring it again. For $N = 7Q + R$, the baseline countdown has $3Q + \lfloor R/3 \rfloor$ positive steps; the present countdown has $2Q + [R \geq 4]$. Each has one exit step.

On a false branch restoration is unchanged. On a true odd branch the quotient transfer costs $Q + 1$ in either implementation. On a true even branch the baseline restore/multiply/restore sequence costs

$$(Q + 1) + (Q + 1) + (2Q + 1) = 4Q + 3,$$

while direct transfer through $R13$ costs $Q + 1$. The exact saving per shared division call is consequently

$$\Delta(N) = Q + [R \in \{3, 6\}] + [R = 0 \text{ and } Q \text{ even}](3Q + 2) \geq 0. \tag{S6.1}$$

For complete corresponding native halting computations, with all other routines unchanged, summing (S6.1) proves a nonincrease of scalar transition counts. The archived v6 counter program is exactly this baseline; its only differences from the present counter program are the shared division phases and the removed $R4$ control. The companion's `check_predecessor.py` verifies this statement and regenerates both residue budgets from the frozen files.

S6.2. **Modulus and storage.** The present integer-scale construction uses both tests modulo 16 and modulo 9, so any modulus supporting it with $K = m/d \in \mathbb{N}$ must be divisible by 144. Removing unused residue positions alone does not change that requirement.

The particular affine/halt action signature in Appendix A also has least modulus 144. One checks that no proper divisor of 144 is a period. To see why this excludes an alternative smaller modulus m', intersect its progressions with the original ones. Whenever residues agree modulo $\gcd(144, m')$,

the intersection is infinite. Equality of actions there forces identical halt status and, on continuing rows, equality of the affine functions. The signature would therefore have period $\gcd(144, m')$, a proper divisor.

The largest numerator coefficient is 177147, the largest denominator is 16, and the largest absolute intercept is 524261. The archived predecessor has maximum absolute normalized coefficient $1572783 = 3 \cdot 524261$. Repacking therefore reduces this coefficient measure by a factor of three while retaining modulus 144; the companion checker verifies both maxima directly from the normalized tables. A signed 32-bit triple representation of the table alone occupies $144 \cdot 12 = 1728$ bytes. Packing a into 18 unsigned bits, b into 20 signed bits, and c into 5 unsigned bits gives 43 bits per row, or 774 bytes for the table.

S7. Explicit nonhalting behavior

Proposition S7.1 (infinitely many disjoint two-cycles)**.** *For every $k \in \mathbb{N}$, the continuing operation has the least-period-two orbit*

$$432k + 6 \longmapsto 1152k + 67 \longmapsto 432k + 6. \tag{S7.1}$$

Different parameters give disjoint orbits. None of these inputs halts successfully.

Proof. The two states have residues 6 and 67. Their rows are $(8, 153, 3)$ and $(3, -153, 8)$, giving

$$\frac{8(432k + 6) + 153}{3} = 1152k + 67, \qquad \frac{3(1152k + 67) - 153}{8} = 432k + 6.$$

Both rows continue, and the second state exceeds the first by $720k + 61 > 0$. The least period is therefore two. The different residues exclude a cross-match between the two state families, and each family is injective in k; hence the orbits are disjoint.

□

The smallest example is $6 \mapsto 67 \mapsto 6$. It gives a hand-checkable nonhalting computation alongside Example 2.1. The formula describes raw-input cycles.

Section S10 classifies all continuing fixed points and defines the total companion H obtained by sending halt and spare rows to the absorbing point 124. Its absorption basin is c.e.-complete, and the two-cycles above persist. Neither version has a global absorption-or-escape dichotomy.

S8. Bit complexity and simulation overhead

S8.1. **Cost of finite scalar execution.** We use ordinary binary representations and count elementary bit operations, as in a multitape implementation of integer arithmetic. Write

$$\ell(n) = \begin{cases} 1, & n = 0, \\ 1 + \lfloor \log_2 n \rfloor, & n > 0. \end{cases}$$

Thus $\ell(n)$ measures the working integer's storage, independently of the fixed rule table. A scalar transition, a source instruction, and a bit operation are different units of time.

Proposition S8.1 (finite-execution cost)**.** *A continuing scalar step satisfies*

$$\ell(n') \le \ell(n) + 17. \tag{S8.1}$$

The additive constant 17 *is optimal for this table. Given an already represented starting integer of length L_0, a straightforward execution of $T \ge 1$ continuing scalar transitions can be implemented in $O(TL_0 + T^2)$ bit operations and $O(L_0 + T)$ bits of space. The space bound retains the current state and arithmetic scratch space, rather than the complete execution history.*

Proof. Put $K = 2^{17}$. For each continuing row, the following two integer inequalities are checked directly from Appendix A:

$$a_r \le Kc_r, \qquad a_r r + b_r < Kc_r(r+1). \tag{S8.2}$$

Writing $n = r + 144q$ and adding $144q\, a_r \le 144q\, Kc_r$ to the second inequality gives $a_r n + b_r < Kc_r(n+1)$. By Lemma 6.1, therefore,

$$0 \le n' < K(n+1) \le 2^{\ell(n)+17},$$

which proves (S8.1), including $n' = 0$. Sharpness follows at $n = 1006$, whose residue is 142: its successor is

$$\frac{177147 \cdot 1006 + 354290}{2} = 89282086.$$

The input has 10 bits and the successor has 27 bits. Induction now gives $\ell(n_t) \le L_0 + 17t$.

Reduction modulo 144, multiplication by a fixed coefficient, signed addition of a fixed coefficient, and exact division by a fixed positive denominator each admit a linear-time binary implementation. The table is fixed, so lookup costs are bounded independently of the input. A transition therefore costs $O(\ell(n))$ bit operations and $O(\ell(n))$ scratch space. Summing the per-step bounds gives

$$\sum_{t=0}^{T-1} (L_0 + 17t) = TL_0 + \frac{17}{2} T(T-1), \tag{S8.3}$$

and the maximum current-state length is at most $L_0 + 17T$. The same bounds accommodate a final halt check.

□

The bound measures the execution of T scalar transitions from an already represented input. Input preparation is analyzed next.

S8.2. **Size of the supplied encodings.**

Proposition S8.2 (exact initial bit length)**.** *For native interpreter input* $C > 0, z \geq 0$*,*

$$\ell\big(9\,2^{17^C 13^z} + 40\big) = 17^C 13^z + 4. \tag{S8.4}$$

Consequently the three-counter input encoding (2.1) *satisfies*

$$\ell(c(e, x)) = 17^{C_e} 13^{3^x} + 4. \tag{S8.5}$$

Proof. Set $N = 17^C 13^z \geq 17$. Then

$$2^{N+3} \leq 9\,2^N + 40 < 2^{N+4},$$

since $40 < 7\,2^N$. Hence the bit length is exactly $N + 4$. Substituting $z = 3^x$ proves (S8.5).

□

For each fixed compiled program, (S8.5) is doubly exponential in the numerical value of x, and in particular is not polynomial in the binary input length $\ell(x)$. At $x = 2$, every positive program code requires at least

$$17\,13^9 + 4 = 180276489345$$

bits for the initial scalar, approximately 22.53 GB in ordinary binary storage, where 1 GB is 10^9 bytes. Minimizing over positive C gives this lower bound at $C = 1$. Writing out this binary encoding already takes at least as many bit operations as there are output bits. Thus the supplied translation, including input preparation, has superpolynomial time and storage costs even before scalar execution starts.

The Mathlib wrapper $I(C, x)$ in (7.1) has binary length $17^C 13^{2^{u(x)}} + 4$. For $x > 0$ with $k = \ell(x)$, its defining recurrence gives $2 \cdot 5^k \leq u(x) < 3 \cdot 5^k$ by induction on the binary digits. That wrapper therefore also has superpolynomial binary length.

The simulation routines themselves also incur substantial overhead. For example, an increment of one of the three original counters multiplies $A = 2^{R_0} 3^{R_1} 5^{R_2}$ by a fixed prime through a loop consuming all A units. This requires $\Omega(A)$ intermediate unit-counter instructions. Since A is exponential in the register values, the unit-counter implementation adds another source of large execution costs.

S8.3. **Uniform bounds and the meaning of efficient universality.** The standard finite-state argument from undecidability excludes any computable uniform time or state-bit bound over all successfully halting inputs of a given binary length. Proposition S8.3 in Section S8.4 records the argument.

The resource bound is uniform over all halting inputs. Bounded nonhalting executions coexist with this obstruction, as the identity progressions and

periodic examples show. Polynomial simulation overhead concerns a different relation: simulator resources as a function of source resources, including the input and output representations. Small universal Turing machines achieving such overhead are known [11].

S8.4. **A uniform halting-resource obstruction.** The following proof applies to any deterministic effective one-integer machine with an undecidable halting set.

Proposition S8.3 (no computable uniform halting-resource bound)**.** *There is no total computable function $B : \mathbb{N} \to \mathbb{N}$ that, for every successfully halting input n, bounds its number of scalar transitions by $B(\ell(n))$. There is likewise no such function bounding the maximum state bit length along every successfully halting run.*

Proof. Theorem 1.1 reduces ordinary halting to successful halting of the fixed scalar machine, so its halting set is undecidable. A time bound B of the stated kind would decide that set by simulating each input for the specified number of transitions, checking for halt at the initial and every subsequent state.

For a purported space bound, enlarge B if necessary so $B(L) \geq \max(1, L)$. Every state of a halting run from an L-bit input would lie in $\{0, \ldots, 2^{B(L)} - 1\}$. Simulate until halt, departure from this finite set, or repetition of a state. Departure rules out subsequent successful halt by the assumed bound. A repeated continuing state rules out halt by determinism. One of these cases occurs after finitely many steps, giving a decision procedure and a contradiction.

□

S9. A conditional coefficient optimum

The five ring controls have common scale 48, base 9, positive effect $(X, Y) \mapsto (X + 17, Y - 1)$, and offsets in cycle order

$$(o_0, o_1, o_2, o_3, o_4) = (0, 1, 2, 4, 3).$$

Their positive rows have

$$a = 2^{17} = 131072, \quad c = 9, \quad b_i = 9o_{i+1} - 131072o_i, \tag{S9.1}$$

with indices modulo 5.

Proposition S9.1 (conditional coefficient optimum)**.** *Retain this five-control cycle with common scale* 48*, work base* 9*, increment* 17*, disjoint full transfer domains, and validity on every raw input. No integer offset selection makes every normalized coefficient smaller than* 524261*. The displayed offsets attain the bound.*

Proof. Suppose all absolute coefficients are at most $B = 524260$, and let $M = \max_i |o_i|$. At an index attaining M, (S9.1) implies $(131072 - 9)M \leq B$, so $M \leq 4$. Full transfer domains occupy entire congruence classes modulo

48; their disjointness makes the five offsets distinct modulo 48, hence distinct integers in $[-4, 4]$.

For any nonnegative offset o_i in this range, the positive row includes the raw input $n = o_i$, and sends it to o_{i+1}. Global nonnegativity propagates around the cycle. Five distinct offsets cannot all be in $\{-4, -3, -2, -1\}$, so every offset is nonnegative. They must be $0, 1, 2, 3, 4$. The successor of offset 4 is at most 3, giving absolute intercept at least $4 \cdot 131072 - 3 \cdot 9 = 524261$, a contradiction. Equation (S9.1) for the saved offsets, together with Appendix A, gives attainment.

□

S10. Further raw dynamics and the total companion

S10.1. **Fixed points of the continuing operation.** The continuing fixed points consist exactly of

$$\{r + 144q : q \in \mathbb{N},\ r \in \{104, 105, 124, 125\}\}$$

and the following 15 individual points:

$$5, 10, 13, 15, 16, 18, 19, 20, 21, 22, 23, 28, 31, 34, 107. \tag{S10.1}$$

This is a finite algebraic classification. On each continuing row solve $(c_r - a_r)n = b_r$, and check its residue and nonnegativity. An identity row contributes its entire progression; any nonidentity row contributes at most one solution. Halt rows are not reinterpreted as fixed-point transitions.

S10.2. **Absorption and undecidability.** Define a total map $H : \mathbb{N} \to \mathbb{N}$ by replacing both halt rows and all four spare identity rows by the constant 124, keeping every other row. Then 124 is fixed and is outside all encoded control domains.

Corollary S10.1. *The set $\{n : \exists t \geq 0,\ H^t(n) = 124\}$ is computably enumerable complete under many-one reductions. In particular, deciding eventual absorption at 124 is undecidable.*

Proof. Forward iteration semidecides absorption. A compiled computation stays inside the occupied simulation domains until its successful halt. Those domains exclude 124 and all spare residues. Thus

$$F_e(x) \downarrow \quad \Longleftrightarrow \quad \exists t\ H^t(c(e, x)) = 124.$$

The total computable map $(e, x) \mapsto c(e, x)$ reduces the ordinary halting set to the absorption basin.

□

There is also an entirely nonnegative table form. Write $n = 144q + r$. On continuing, nonreset rows put

$$A_r = 144a_r/c_r, \qquad B_r = (a_r r + b_r)/c_r;$$

on reset rows put $A_r = 0, B_r = 124$. Then $H(n) = A_r q + B_r$ with $A_r, B_r \in \mathbb{N}$. Here $\max A_r = 12754584$ and $\max B_r = 12754582$. The larger intercepts illustrate that coefficient size depends on the chosen representation.

Every two-cycle in (S7.1) remains in H, because neither residue 6 nor residue 67 is changed. Thus neither the companion nor the stopped algorithm has a global dichotomy between absorption and escape to infinity. Section S12 also gives a longer cycle reached from a native interpreter input.

S11. Formal bridge and verification environment

S11.1. **The Mathlib bridge.** Mathlib supplies a proved simulation of its partial-recursive program codes by a four-stack machine with finite local storage. Each stack is packed in base five; four counters hold the stacks and a fifth supplies scratch space. Push, pop, and peek are compiled to finite arithmetic routines. The five counters are then packed by primes $2, 3, 5, 7, 11$ and compiled through the same M2 interpreter and scalar rules used above. The fifth counter changes the universality bridge, not the final rule table. Its initial prime packing is $2^{u(x)}$, which accounts for the difference between (7.1) and (2.1).

Both directions of simulation are essential. Termination of each arithmetic routine proves that a finite source transition cannot become an infinite internal computation. Conversely, a well-founded rank on omitted pure tests prevents an infinite source execution from being compressed into finitely many scalar transitions. Halting reflection then excludes accidental successful halts. These arguments apply to arbitrary natural counter values.

S11.2. **Reproduction.** The development pins Lean to version 4.34.0 and Mathlib to revision

```
5ed2965256430c3649e86755f9576b54eca72435
```

From the extracted Lean companion, the command

```
python3 scripts/verify.py
```

checks regenerated certificates against the frozen files, builds the root module, checks theorem dependencies, rejects deliberately false test statements, and replays the proof environment through a fresh official Lean kernel. The recorded run passes all stages. Only the standard logical axioms `propext`, `Classical.choice`, and `Quot.sound` are permitted. No `sorry`, custom machine axiom, or native-evaluation axiom is accepted. The external generators produce candidate data; the mathematical claims about that data are checked in Lean.

Both verifiers write fresh reports separately from the archived evidence, so the integrity checks listed in each archive's README remain valid after verification.

For Proposition S7.1, `Oloa.two_cycle_least_period` proves least period two and `Oloa.two_cycles_disjoint` proves disjointness. The theorem

`Oloa.companion_two_cycles` proves persistence in H. The companion's `STATUS.md` indexes the formalized results.

The numerical table has SHA-256

`52f2f012ebbfc80353de4df91b3fe537f413c47e45d96ee77cf6519650908668`

An independent checker uses only the CSV and Python's standard library. The command

```
python3 check_numerical.py
```

checks all continuing residue progressions, coefficient maxima, the fixed-point calculation, and the exact executions below. It also verifies the two-cycle identities symbolically, both quotient-parity examples, and all 142 row certificates (S8.2) together with the sharpness witness. The small halting example is regenerated directly from the same table. The publication package supplies one canonical data archive and a separate Lean archive, with manifests of file hashes. Extract the data archive to run the numerical checker. Its authoritative table is `machine_v8/machine_144.csv`; its compiler is the top-level `compiler_v8.py`, which binds the input and output offsets to 40 and 107.

The independent source and arithmetic checks include all 32 guarded branch identities, closure of the 431 source label/mask pairs, 138 occupied continuing affine identities, and the 142 raw continuing progressions. The counter-contract checks cover 45 positive edges and 76 fixed-counter exits. Their periodic reduction examines 1687 admissible infinite progressions. Solver and mutation checks provide further cross-checks.

S12. Exact executions

A diagnostic family.
For $L = 11^k$, $k \geq 0$, the prefetched seed

$$n_k = 72\,2^{4L} + 21$$

halts at $16\,2^{143L} + 107$ after exactly $18L + 6$ continuing transitions. These seeds are outside Lemma 5.2: their $D1$ configuration has $(X, Y) = (4L, 0)$, but $34 \nmid 4 \cdot 11^k$. In source-register notation their label-1 positivity mask is 32 for $k = 0$ and 33 for $k > 0$; neither belongs to Section S3. The formula therefore uses a direct calculation with the scalar rows, independently of the source invariant and the counter-contract lemma.

That calculation follows $D1/R0$ from $4L$ to $2L$ in $4L + 2$ transitions, $D3/R1$ from $2L$ to L in $2L + 2$, and $MP/R9$ from L to $143L$ in $12L + 2$. At these particular boundaries, parity selects the indicated next routine. Since L is odd and coprime to 3, the final $R3$ residue is a halt, with no further transition. The sum proves the formula for every k; exact executions for $k = 0, 1, 2, 3$ give $24, 204, 2184, 23964$ steps. The smallest seed is 1173.

Native executions.

The native input $C = 1, z = 1$, namely $9\,2^{221} + 40$, halts after 23,961 transitions at $16\,2^{119} + 107$. Its decoded output is zero, and its peak scalar length is 9,814 bits. The corresponding run of the comparison implementation took 25,784 steps; (S6.1) accounts for all 1,823 saved steps.

The native input $C = 2, z = 0$ is $9\,2^{289} + 40$. After 1,264 transitions it reaches

$$p = 9\,2^{867} + 36. \tag{S12.1}$$

An independent executor reading only the numerical CSV verifies

$$T^{253194}(p) = p, \qquad T^j(p) \neq p \quad (1 \leq j < 253194), \tag{S12.2}$$

where T is the continuing step operation. Every intermediate row continues with exact arithmetic; the checker compares integers, not hashes. Equation (S12.2) certifies **least period 253,194**. Peak length on this cycle is 134,675 bits. The orbit survives in H, since it never visits a halt or spare row.

To reproduce the least-period test, start at (S12.1), apply the CSV-selected rules, reject any halt, and stop at the first exact return. The independent checker also verifies the prefix.

The four CRT-compiled source examples require 12,929,534 primitive interpreter steps in total; their outer scalar inputs were not materialized. The smaller examples above were executed directly on the scalar table.

S13. An exact executor and finite arithmetic check

This reference implementation assumes `rows` is the ordered list of 144 triples from Appendix A. It uses unbounded integers. `run` implements the partial stopped computation, returning the current integer at a successful halt.

```
from math import gcd

def validate(rows):
    assert len(rows) == 144
    halts = []
    for r, (a, b, c) in enumerate(rows):
        if c == 0:
            assert (a, b) == (0, 0)
            halts.append(r)
            continue
        assert a >= 0 and c > 0
        assert gcd(gcd(abs(a), abs(b)), c) == 1
        assert (144 * a) % c == 0
        assert (a * r + b) % c == 0
        assert a * r + b >= 0
    assert halts == [43, 139]

def step(n, rows):
    assert isinstance(n, int) and n >= 0
    a, b, c = rows[n % 144]
    if c == 0:
        return None  # successful halt at the current n
    q, rem = divmod(a * n + b, c)
    assert rem == 0 and q >= 0
    return q

def run(n, rows):
    while True:
        nxt = step(n, rows)
        if nxt is None:
            return n
        n = nxt
```

The assertions explain the finite certificate and may be replaced by explicit error checks in production software. The actual supplied audit checkers retain their checks under optimized Python. Exact format extraction for (2.2), CRT encoding for (3.3), and the counter certificates are checked by the compiler and simulation layers.

References

[1] A. M. Ben-Amram, *Mortality of iterated piecewise affine functions over the integers: Decidability and complexity*, in 30th Symposium on Theoretical Aspects of Computer Science, Leibniz Int. Proc. Inform. **20**, Schloss Dagstuhl, 2013, 514–525. doi:10.4230/LIPIcs.STACS.2013.514.

[2] M. Carelli, *Loop termination and generalized Collatz sequences*, in 53rd International Colloquium on Automata, Languages, and Programming, Leibniz Int. Proc. Inform. **374**, Schloss Dagstuhl, 2026, 175:1–175:21. doi:10.4230/LIPIcs.ICALP.2026.175.

[3] J. H. Conway, *Unpredictable iterations*, in Proceedings of the 1972 Number Theory Conference, University of Colorado, Boulder, 1972, 49–52. Reprinted in J. C. Lagarias (ed.), The Ultimate Challenge: The $3x+1$ Problem, American Mathematical Society, Providence, RI, 2010, 219–223 (with editorial commentary).

[4] L. De Mol, *Tag systems and Collatz-like functions*, Theoret. Comput. Sci. **390** (2008), no. 1, 92–101. doi:10.1016/j.tcs.2007.10.020.

[5] J. Endrullis, C. Grabmayer, and D. Hendriks, *Complexity of Fractran and productivity*, in Automated Deduction—CADE 22, Lecture Notes in Comput. Sci. **5663**, Springer, 2009, 371–387. doi:10.1007/978-3-642-02959-2_28.

[6] F. Kaščák, *Small universal one-state linear operator algorithm*, in Mathematical Foundations of Computer Science 1992, Lecture Notes in Comput. Sci. **629**, Springer, 1992, 327–335. doi:10.1007/3-540-55808-X_31.

[7] K. Knight, *A small Collatz rule without the plus one*, Complex Systems **35** (2026), no. 1, 1–9. doi:10.25088/ComplexSystems.35.1.1.

[8] S. Kohl, *Wildness of iteration of certain residue-class-wise affine mappings*, Adv. in Appl. Math. **39** (2007), no. 3, 322–328. doi:10.1016/j.aam.2006.08.003.

[9] S. A. Kurtz and J. Simon, *The undecidability of the generalized Collatz problem*, in Theory and Applications of Models of Computation, Lecture Notes in Comput. Sci. **4484**, Springer, 2007, 542–553. doi:10.1007/978-3-540-72504-6_49.

[10] E. Lehtonen, *Two undecidable variants of Collatz's problems*, Theoret. Comput. Sci. **407** (2008), nos. 1–3, 596–600. doi:10.1016/j.tcs.2008.08.029.

[11] D. Woods and T. Neary, *On the time complexity of 2-tag systems and small universal Turing machines*, in 47th Annual IEEE Symposium on Foundations of Computer Science, IEEE, 2006, 439–446. doi:10.1109/FOCS.2006.58. Expanded version: arXiv:cs/0612089.

[12] E. Yolcu, S. Aaronson, and M. J. H. Heule, *An automated approach to the Collatz conjecture*, J. Automat. Reason. **67** (2023), article 15, 44 pp. doi:10.1007/s10817-022-09658-8.

Independent Researcher