Division of Explicit Finite Dynamical Systems Is NP-Complete

Nuutti Kuosa

22 November 2026

Abstract

Given the successor tables of two finite dynamical systems A and B, can we find a system X whose synchronous product with A is isomorphic to B? We prove that deciding whether such an X exists is NP-complete even when A and B are permutations. This answers a question of Naquin and Gadouleau. Different cycle lengths in X can contribute to the same cycle length in B. We use their multiplicities to encode a 3-SAT formula so that its satisfying assignments correspond bijectively to quotient isomorphism classes. Counting these classes for permutation inputs is #P-complete under parsimonious reductions. Hardness holds with distinct, pairwise coprime, squarefree cycle lengths in A and squarefree cycle lengths in B. For a formula with n variables and m clauses, the two output tables can be constructed with $(n+m)^{5+o(1)}$ states in total.

hexagon:2611.00886v1

1 The division problem

A finite dynamical system is a nonempty finite set with a map to itself. Given systems A and B, can we label the states of B by pairs (a, x) so that the first coordinate follows A and the second follows a single map X? We write $AX = B$ if a bijection identifies the transition of B with $(a, x) \mapsto (A(a), X(x))$, and call X a *quotient*. Its isomorphism class is a *quotient class*. The input lists the successor of every state in A and B. The quotient may have any number of connected components.

Even a four-state permutation can have two different quotients. Write C_d for one d-cycle and $+$ for disjoint union. Take $A = C_2$ and $B = 2C_2$. If $X = 2C_1$, the product has one 2-cycle above each fixed point of X. If $X = C_2$, label both factors by 0 and 1. The product swaps $(0, 0)$ with $(1, 1)$ and $(0, 1)$ with $(1, 0)$. Both quotients therefore give B.

More generally, a pair of states on cycles of lengths a, b first returns to its starting position after $\operatorname{lcm}(a, b)$ steps. The ab pairs therefore split into $ab/\operatorname{lcm}(a, b) = \gcd(a, b)$ cycles of this length, giving

$$C_a C_b = \gcd(a, b)\, C_{\operatorname{lcm}(a,b)}. \tag{1}$$

A permutation is determined up to isomorphism by the number of its cycles of each length.

Naquin and Gadouleau asked whether division is NP-complete for general systems and raised the question even "on cycles" [11, §7, p. 17]. Doré et al. recorded the exact complexity of division as unknown [6, §1]. After solving division for connected quotients, they identified removing the connectivity assumption as a main research direction [6, §6].

Theorem 1. *Deciding whether $AX = B$ for some finite system X is NP-complete under polynomial-time many-one reductions for explicitly listed successor tables. NP-hardness holds when A and B are permutations, the cycles of A have distinct, squarefree, pairwise coprime lengths, and every cycle length of B is squarefree.*

Proof idea. Suppose $X = uC_7 + vC_{21} + wC_{35}$. Multiplication by C_{15} sends all three lengths to 105, with multiplicities $u, 3v, 5w$. Thus one entry of the product reads the weighted sum $u + 3v + 5w$. We choose cycle lengths so that several such sums read overlapping parts of an array. Section 2 encodes intervals on these sums by a single division equation. Section 3 uses sums of two bits to copy a truth value and sums of three bits to test clauses. Each satisfying assignment gives one quotient up to isomorphism. Reading its cycle multiplicities recovers the assignment.

Membership in NP. Dennunzio et al. established membership in NP for polynomial equations with a constant side [4, Thm. 2]. For division, if $|A|$ divides $|B|$, guess a successor table on $|B|/|A|$ states and a bijection from its product with A to B. Bijectivity and commutation with the two maps can be checked entry by entry. If $|A|$ does not divide $|B|$, reject. If B is a permutation, surjectivity of the product also forces X to be a permutation.

Related work

Why one unknown matters. Porreca reported NP-completeness for a single linear equation with a constant right-hand side and a variable number of unknowns, crediting joint work with Bridoux and Naquin-Touileb [13]. Ascone, Porreca and Rolland proved NP-hardness for linear permutation equations with three unknowns and asked how many unknowns suffice [1, p. 2]. Porreca and Rolland singled out $AX = B$ as unclassified [14, §1]. Gadouleau and Johnson also recorded the problem as open for permutations [8, §1]. Theorem 1 answers the variable-count question for these linear equations. One unknown suffices even when every input state is listed. The same instances establish hardness for permutations specified by cycle lengths and multiplicities in binary, since traversing the successor tables gives those lists in polynomial time.

Where polynomial algorithms apply. The algorithms for restricted cases depend on what we know about the divisor or the quotient in advance. Doré et al. find a connected quotient, when one exists, in polynomial time [6, Thm. 3.13]. Their final question asks for two unknown connected factors when only the target is given [6, §6].

For permutation inputs, let m_A be the least common multiple of the cycle lengths of A, and let $\tau(m_A)$ count its positive divisors. After the size check $|A| \mid |B|$, Bridoux et al. count quotient classes in time $O(|B|^3(|B|/|A|)^{\tau(m_A)})$ [2, Thm. 1, eq. (2)]. This gives one polynomial bound for every family on which $\tau(m_A)$ has a constant upper bound, including the case of a fixed A. They also decide division in polynomial time when every cycle length of A greater than one is prime, however many such lengths occur. In that case they can list all solutions in polynomial time when every cycle length of B divides m_A [2, §§4–6]. Dennunzio et al. decide $C_pX = nC_q$ in polynomial time even when p, q, n are encoded in binary [5, Thm. 4].

Porreca and Rolland solve division in polynomial time when the shortest divisor cycle length divides all the others [14, §1], as part of their treatment of pseudo-injective equations [15, Thm. 61]. A fixed point in the divisor guarantees uniqueness whenever a solution exists [11, Thm. 34]. Polynomial algorithms are also known for pure permutation roots [16, 12, 15]. Gadouleau et al. relate polynomial-time division for partial and total systems [9].

Rolland gives a fixed-parameter algorithm for division, parameterized jointly by the number of distinct connected component types of B and the shortest cycle length of A [17, Thm. 4.43, p. 159]. In our reduction the shortest divisor cycle has length two, while the number of distinct target cycle lengths grows with the formula. The divisor has $\Theta(\log N/\log\log N)$ cycles, where $N = |A| + |B|$.

After the reduction, Section 4 bounds its cost in states, Section 5 checks small instances, and Section 6 treats scalar polynomials and uniqueness.

2 Realizing intervals by permutation division

Give each of three literal bits a positive weight. The interval from the smallest weight to the sum of all three accepts exactly the triples with at least one true bit. How can one product equation enforce both endpoints?

Write $[C_d]Y = y_d$ for the coefficient of C_d in a finite sum $Y = \sum_d y_d C_d$. We extend the cycle product bilinearly to rational coefficients while constructing the target. A sum represents an

actual nonempty permutation when its coefficients are nonnegative integers and at least one is positive.

Think of a rectangular array, with one prime labelling each position along an axis. We use different primes on different axes so that a site can be identified from the product of its labels alone. Take $D \geq 2$ pairwise disjoint finite sets of primes H_j, each with at least two elements. Write $Q_j = \prod_{p \in H_j} p$ and choose a prime ω outside all the groups. A *site* is a full tuple and a *face* is a partly or fully specified tuple. Their respective sets are

$$V = \prod_{j=1}^{D} H_j, \qquad E = \prod_{j=1}^{D} (\{1\} \cup H_j).$$

The label 1 leaves a coordinate unspecified. A face with just one unspecified coordinate is a line. For $e \in E$, put $d_e = \prod_j e_j$ and let $s(e)$ count its unspecified coordinates. Write $v \succeq e$ when $v \in V$ agrees with all specified coordinates of e. Because the Q_j are coprime, the factors $e_j = \gcd(d_e, Q_j)$ can be read back from d_e.

We store an entry $0 \leq x_v \leq \kappa_v$ at each site v, with integral capacity $\kappa_v \geq 0$. A site is *active* if its capacity is positive. On a face e, the weight of an entry is the product of its labels in the unspecified coordinates. The weighted sum and its maximum are therefore

$$f_e(x) = \sum_{v \succeq e} \frac{d_v}{d_e} x_v, \qquad W_e = f_e(\kappa). \tag{2}$$

Give each e an integral interval $[L_e, R_e] \subseteq [0, W_e]$. At a site require $[L_v, R_v] = [0, \kappa_v]$. A feasible array satisfies $L_e \leq f_e(x) \leq R_e$ for every e.

A one-bit example. Take the prime groups $\{3, 5\}$ and $\{7, 11\}$, with $\omega = 2$. Give the site $(3, 7)$ capacity one and every other site capacity zero. There is just one entry to choose, $x = x_{(3,7)}$. At $(1, 7)$ the weighted sum is $3x$, since $d_{(3,7)}/d_{(1,7)} = 21/7 = 3$. Give every other face its full interval $[0, W_e]$. The interval at $(1, 7)$ then makes the following choices possible.

Interval for $3x$	$[0, 3]$	$[3, 3]$	$[1, 2]$
Feasible bits x	0, 1	1	none

The last column explains the rational feasibility assumption in the next lemma. Neither bit works, yet $x = \frac{1}{2}$ gives $3x = \frac{3}{2}$. As the lemma shows, we can still construct a target permutation B for this interval. The resulting equation $(C_2 + C_{15} + C_{77})X = B$ then has no quotient. Figure 1 shows the entry and the line that reads it.

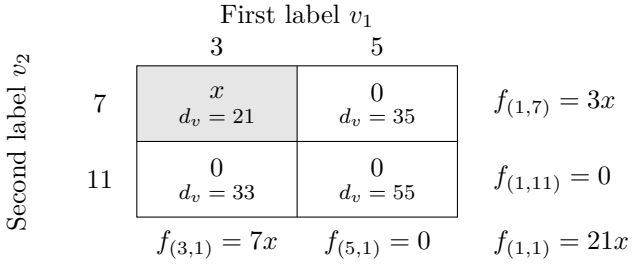

Figure 1: The four sites of the one-bit example. The shaded site has capacity $\kappa_{(3,7)} = 1$, and the other three have capacity zero. On the top row the first label supplies the weight, giving $3x_{(3,7)} + 5x_{(5,7)} = 3x$. On the whole array both labels supply weights, so the shaded site's contribution is $3 \cdot 7x = 21x$.

The construction. The interval at e holds exactly when its two signed distances, $f_e(x) - L_e$ and $R_e - f_e(x)$, are nonnegative. Their sum is the known width $U_e = R_e - L_e$. We put one distance at cycle length ωd_e, and ω times the other at d_e. Multiplying by C_ω sends both entries to length ωd_e. Since $\gcd(\omega, d_e) = 1$, the cycle rule gives $C_\omega C_{d_e} = C_{\omega d_e}$ and $C_\omega C_{\omega d_e} = \omega C_{\omega d_e}$. Each distance thus contributes ω times its value, for a total of ωU_e.

Which distance should go at the longer length? A face sum is a weighted sum of the face sums obtained by specifying one more coordinate. To make their changes cancel in the product, reverse the choice of distance each time a coordinate is left unspecified. For any rational array x, put

$$z_e(x) = \begin{cases} f_e(x) - L_e, & s(e) \text{ even}, \\ R_e - f_e(x), & s(e) \text{ odd}, \end{cases} \qquad Y(x) = \sum_{e \in E} (z_e(x) C_{\omega d_e} + \omega(U_e - z_e(x)) C_{d_e}).$$

All displayed lengths are distinct, so $Y(x)$ has nonnegative coefficients exactly when x is feasible. Set $A = C_\omega + \sum_j C_{Q_j}$ and compute $B = AY(0)$ in signed integer arithmetic. This computation uses only the capacities and intervals.

Lemma 2. *Suppose that some $\kappa_v > 0$ and that a feasible rational array exists.*

(a) *For every rational array x, $AY(x) = B$. The target B is a nonempty permutation.*

(b) *The map $x \mapsto Y(x)$ is a bijection from feasible integral arrays to quotient isomorphism classes for $AX = B$. Its inverse reads $x_v = [C_{\omega d_v}]X$ at each site v.*

(c) *The number of states in B satisfies*

$$|B| = |A|\, \omega \sum_{e \in E} d_e U_e \;\le\; |A|\, \omega\, 2^D \sum_{v \in V} d_v \kappa_v. \tag{3}$$

Proof. Independence of the array. In the one-bit example let R be the upper bound on $3x$. Keep the other three site entries at zero. The multiplicities of C_{42} and C_{14} in $Y(x)$ are x and $R - 3x$. Multiplication by C_{15} sends both to length 210. The two contributions are $3x$ and $R - 3x$, giving R in all. The third possible length on this line, 70, has multiplicity zero because the site $(5, 7)$ has capacity zero.

For the general calculation fix a coordinate j and all the other labels, allowing those labels to be 1. Let δ be their product. Write $e^{(1)}$ for the face leaving coordinate j unspecified and $e^{(p)}$ for the face specifying $p \in H_j$ there. The sites above $e^{(1)}$ split according to their jth label. A site with label p has weight p times its weight on $e^{(p)}$, giving

$$f_{e^{(1)}}(x) = \sum_{p \in H_j} p\, f_{e^{(p)}}(x).$$

Use subscripts $1, p$ for the corresponding values of z, U, L, R. Multiplication by C_{Q_j} gives $C_{Q_j} C_\delta = C_{\delta Q_j}$ and $C_{Q_j} C_{\delta p} = p C_{\delta Q_j}$, since $\gcd(\delta, Q_j) = 1$. Both identities also apply with $\omega\delta$ in place of δ. Thus

$$[C_{\omega\delta Q_j}]\, C_{Q_j} Y(x) = z_1 + \sum_{p \in H_j} p z_p,$$

$$[C_{\delta Q_j}]\, C_{Q_j} Y(x) = \omega\Big(U_1 + \sum_{p \in H_j} p U_p\Big) - \omega\Big(z_1 + \sum_{p \in H_j} p z_p\Big).$$

Each $e^{(p)}$ has one fewer unspecified coordinate than $e^{(1)}$, so their z's use opposite signs for the face sums. Writing k for the common value of $s(e^{(p)})$, substitution gives

$$z_1 + \sum_{p \in H_j} p z_p = \begin{cases} R_1 - \sum_{p \in H_j} p L_p, & k \text{ even}, \\ \sum_{p \in H_j} p R_p - L_1, & k \text{ odd}. \end{cases}$$

Both product coefficients are therefore independent of x. These are all the lengths contributed by this line, and each determines δ. The remaining divisor cycle gives

$$C_\omega Y(x) = \sum_e \omega U_e C_{\omega d_e}. \tag{4}$$

We have proved $AY(x) = AY(0) = B$. Evaluating at zero gives integer coefficients, even if zero violates an interval. Evaluating at a feasible rational array shows that the same coefficients are nonnegative. At a site with $U_v = \kappa_v > 0$, the product with C_ω makes a positive contribution, so B is nonempty. This proves (a).

Ruling out other cycle lengths. So far every feasible integral array gives a quotient. To reverse the construction, first consider any cycle in a putative quotient. We can rule it out if multiplying it by even one cycle of A would produce a cycle length absent from B. For instance, in the one-bit example a quotient cycle of length 15 would give a cycle of length $15 \cdot 77$ on multiplication by C_{77}. That cycle length uses both prime groups in full. To see why it is absent, examine the terms of $B = AY(0)$ before cancellation.

All cycle lengths divide the squarefree number $\omega \prod_j Q_j$. In each cycle length, at most one group contributes more than one prime. Indeed $C_\omega Y(0)$ uses at most one label from each group. Every cycle length of $C_{Q_j} Y(0)$ contains all primes of H_j and at most one from each other group. Cancellation only removes cycle lengths from this list.

If $AX = B$, then X is a permutation since B is surjective. Let C_b be any cycle in X. Every cycle contributed by a pair of cycles from A and X occurs in their product with positive multiplicity. In particular, the length $\operatorname{lcm}(\omega, b)$ occurs in B, so b divides $\omega \prod_j Q_j$. If b used two primes from H_j, choose $k \neq j$. Then $C_{Q_k} C_b$ would give a positive term of B using at least two primes from each of H_j, H_k, which we have just ruled out. This step uses a second group and at least two primes in it. Hence $b = d_e$ or $b = \omega d_e$ for some e.

Reading the entries back. For X with this support, the cycle lengths contributed to AX by its different divisor summands are disjoint. A cycle length in $C_\omega X$ contains no full group because every group has at least two primes, and one in $C_{Q_j} X$ contains exactly H_j in full. The same separation holds for $AY(x)$ for every x. Looking at which prime groups occur in full therefore tells us which summand of A contributed each cycle length of the product. In particular we can read the product $C_\omega X$ separately. By (4) its coefficient at ωd_e is ωU_e. Put $z'_e = [C_{\omega d_e}]X$. Equation (4) now gives $[C_{d_e}]X = \omega(U_e - z'_e)$, including at tuples with $U_e = 0$. At a site v we have $s(v) = 0$ and the interval is $[0, \kappa_v]$, so $z_v(x) = x_v$. Set $x_v = z'_v$. These entries are nonnegative integers, being cycle multiplicities in X.

It remains to identify the coefficients at the faces with unspecified coordinates. Use the notation $e^{(1)}, e^{(p)}, \delta$ from (a). Only $C_{Q_j} X$ contributes to length $\omega\delta Q_j$, and dividing by ωQ_j recovers δ and hence the fixed labels. The cycle rule on this line gives

$$z'_{e^{(1)}} = [C_{\omega\delta Q_j}]B - \sum_{p \in H_j} p\, z'_{e^{(p)}}.$$

Each face on the right has one more specified coordinate. Starting at the sites and proceeding in order of increasing $s(e)$, the recurrence determines every z'_e. Part (a) gives $AY(x) = B$ for every rational array, so it applies to the integer array just recovered. The coefficients of $Y(x)$ therefore satisfy the same recurrence, and they agree with z'_v at the sites by the definition of x_v. Induction gives $z'_e = z_e(x)$ everywhere. The coefficients at both lengths now agree, so $X = Y(x)$. The nonnegativity of X forces $0 \leq z_e(x) \leq U_e$ for every face, including faces of zero width. Hence x is feasible. Conversely, each feasible integral x gives a quotient by (a), and its site coefficients recover it uniquely. This proves (b).

Counting the states. Finally the two terms for e in $Y(x)$ have total size $\omega d_e U_e$. Summing and multiplying by $|A|$ gives the exact size in (3). For the upper bound replace each U_e by W_e. In $d_e W_e$ a site $v \succeq e$ contributes $d_v \kappa_v$. Each of the D coordinates of v may be kept or left unspecified, so this contribution occurs exactly 2^D times in the sum over e. □

De Loera and Onn proved that feasibility of a $3 \times c \times h$ nonnegative integer table with prescribed two-dimensional marginals is NP-complete [3, Thm. 1.1]. Their marginal sums have unit coefficients. Here the coefficients d_v/d_e come from cycle multiplication.

3 From a Boolean formula to two successor tables

How can an interval test a Boolean condition? Suppose two sites on a line carry bits with weights 3 and 5. The possible sums are 0, 3, 5 and 8, so the interval $[3, 5]$ makes the bits opposite. Along a path of two such tests the original bit reappears at the other end. On a line with three bits, the weighted sum is at least the smallest of its three positive weights exactly when at least one bit is 1. Now imagine testing a clause such as $(x \vee \neg y \vee z)$. We want one line with three active sites, one for each literal in its clause position. Give each clause position its own plane. On each plane, copying lines carry a literal's bit to every address where a clause asks for it. A second kind of line links the planes so that every occurrence of x agrees and every occurrence of $\neg x$ reads its complement. The clause line can then test the three bits with a single interval. The construction below gives each of these lines precisely the sites it needs.

Take a conjunction of clauses with at most three literals each [10]. If its variables are given by an explicit list, first add $(x \vee \neg x \vee x)$ for every listed but unused variable. This preserves both choices for x when we count assignments. Otherwise the variables are those occurring in the formula. Set aside the empty cases until the end of the construction. In every remaining short clause, repeat a literal until the clause has three positions. Relabel the $n \geq 1$ variables. A literal may occur in any position in many clauses. We give it a copy on each of three planes, one plane for each clause position. There are $t = 2n + 1$ identifiers, one for each signed literal and one padding identifier $\perp$. Padding will let us test just two planes while giving all three positions a definite address.

Assigning addresses. Choose the least $u \geq 3$ with $u^u \geq t$, and the least $g \geq 1$ with $u^g \geq t$. The two choices balance the cost of an address. At a site, d_v uses one prime per coordinate, whereas the divisor cycle for a coordinate uses all its primes. Taking $g \leq u$ and $u \log u = O(\log(t + 2))$ will keep both lengths polynomial in t (Section 3.3). Assign distinct words in $\{0, \ldots, u - 1\}^g$ to the identifiers. The address $\widehat{\alpha}$ of an identifier α is its word with each digit duplicated. A site has three slots of $2g$ coordinates each and one plane coordinate with three values. Fixing the three slots and varying the plane gives a line through the three clause positions. Each address coordinate has u labels. Assign a distinct prime to every label of every coordinate using consecutive primes starting at 3, taking one address coordinate at a time and the plane last. A label's weight is the prime assigned to it in its own coordinate. For example, the digit 0 has positive weight. Since all label primes are odd, we may take the guard prime $\omega = 2$.

Connecting the addresses of one literal. To copy a bit we need to travel between duplicated words by changing one coordinate at a time. Let P be the path of digit pairs obtained by listing (a, a) for $0 \leq a < u$ and inserting $(a, a+1)$ between successive diagonal pairs. For $u = 3$ it is

$$(0,0),\ (0,1),\ (1,1),\ (1,2),\ (2,2).$$

A slot ranges through P^g, the Cartesian product of g such paths. Why use a pair of coordinates for one digit? Fix either member of a pair and at most two points of P remain. This is what lets a copying line isolate two sites even when the alphabet is large.

On plane $j \in \{1, 2, 3\}$, for each signed literal ℓ fix slot j to $\widehat{\ell}$ and allow the other two slots to range through $P^g \times P^g$. Give these sites capacity one and all other sites capacity zero. Each plane thus has one disjoint copy of $P^g \times P^g$ per literal. The padding address can be visited in either varying slot because every duplicated word belongs to P^g. On plane j, a site with padding in its own slot j has capacity zero.

Copying a bit. In each copy we need only enough edges to reach every site. Take a spanning tree of its Cartesian product graph. For every tree edge impose a condition on the face leaving

only its varying coordinate unspecified. Exactly the two active endpoints lie on this line. The plane and its literal are fixed, as is the partner coordinate of the varying digit. If the two endpoint bits are b, b' and the two labels are p, q, the face sum is $pb + qb'$. The interval $[\min(p,q), \max(p,q)]$ imposes $b + b' = 1$.

Number the positions in P starting at zero. A graph edge reverses the parity of the sum of the positions in the varying slots. Every site where all digit pairs are diagonal has even parity. Consequently, all such sites in the copy for (j, ℓ) carry one bit $b_{j,\ell}$. Either value of this bit extends uniquely to the copy by assigning its complement at sites of odd parity. Figure 2 shows one choice of the tree and the sites where the construction reads the bit.

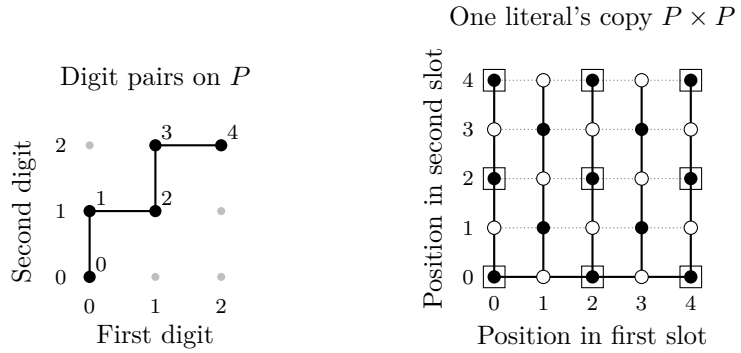

Figure 2: Copying one bit when $u = 3$ and $g = 1$. On the left the five positions on P are numbered in path order. On the right only the solid edges of the grid are used as copying lines. They form a spanning tree. If the bit at $(0, 0)$ is b, filled sites carry b and open sites carry $1 - b$. At the nine boxed sites both slots contain duplicated digits. All nine have even parity and carry b. The tests that compare literal signs or check clauses read the bit only at boxed sites.

3.1 Identifying a literal across the three planes

Write $(\alpha_1, \alpha_2, \alpha_3)$ for the line that fixes the three slots to their respective diagonal words $\widehat{\alpha_i}$ and leaves the plane unspecified. On it, plane j has an active endpoint precisely when α_j is a literal. Its bit there is b_{j,α_j}. For example, $(x, \bot, \neg x)$ exposes the bit for x on the first plane and the bit for $\neg x$ on the third. The second plane contributes zero. Let π_1, π_2, π_3 be the plane primes.

For each variable x and every ordered pair $j \neq k$, place x in slot j, $\neg x$ in slot k and $\bot$ in the remaining slot. We call these six lines the *sign tests* of x. On each line impose the interval $[\min(\pi_j, \pi_k), \max(\pi_j, \pi_k)]$. It forces exactly one of $b_{j,x}, b_{k,\neg x}$ to be one. The six conditions form the cycle

$$b_{1,x},\ b_{2,\neg x},\ b_{3,x},\ b_{1,\neg x},\ b_{2,x},\ b_{3,\neg x},\ b_{1,x}.$$

Thus the three positive copies agree and the three negative copies are their complements. A truth value for x satisfies all six conditions and determines all six bits.

3.2 Enforcing a clause

For each clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ impose on its line (ℓ_1, ℓ_2, ℓ_3) the interval

$$\min(\pi_1, \pi_2, \pi_3) \ \leq\ \pi_1 b_{1,\ell_1} + \pi_2 b_{2,\ell_2} + \pi_3 b_{3,\ell_3} \ \leq\ \pi_1 + \pi_2 + \pi_3. \tag{5}$$

All three planes are active, also when literals repeat. This interval excludes exactly the all-zero triple.

Could two instructions give the same line incompatible intervals? A copying line fixes its plane and the address of its owning literal. It therefore belongs to one copy, and its two active

endpoints determine a single tree edge. Both other kinds of line vary the plane. A sign test has one padding slot, which identifies the remaining pair of positions. A clause has three literal addresses. Repeating an ordered clause simply repeats its interval. This accounts for every marked line. Give each remaining face its full interval $[0, W_e]$.

An integral feasible array now yields a truth assignment from $b_{1,x}$ for each variable. The six-cycle gives the value of every literal on every plane, and (5) makes every clause true. Conversely, assign the truth values at diagonal words and extend by parity in each copy. This gives a feasible integral array for every satisfying assignment. At all inactive sites the entry is zero.

A feasible *rational* array exists even if the formula has no satisfying assignment. Give every active site value $1/2$ and every unused one zero. On a two-endpoint line the weighted sum is $(p + q)/2$, which is between its two labels. On a clause line it is $(\pi_1 + \pi_2 + \pi_3)/2 \geq \min_j \pi_j$. All other faces have their default intervals. Lemma 2 therefore constructs a nonempty integral target on satisfiable and unsatisfiable inputs alike, and supplies the converse for arbitrary candidate quotients.

The half assignment gives a further check on the construction. All entries of $Y(x)$ at this point are nonnegative half-integers. Doubling their multiplicities gives a genuine quotient of $B + B$ by A, even when the original formula is unsatisfiable.

Empty cases. After accounting for unused variables, an empty conjunction has one assignment and gives (C_2, C_2) with unique quotient C_1. A formula containing an empty clause has no satisfying assignment. For this case take $(A, B) = (C_2 + C_3, C_2 + 3C_6)$. A fixed point of a quotient would give a 3-cycle in B upon multiplication by C_3. Since B has no such cycle, the only possible contributions to its 2-cycles come from $C_2C_2 = 2C_2$. Yet B has just one 2-cycle. This fixed instance also satisfies $(C_2 + C_3)(C_2 + C_6) = 2C_2 + 6C_6$.

3.3 Writing the successor tables

At this point a formula has determined two lists of cycles. A short list can describe a very large permutation. To finish the reduction in the explicit input model we must bound the number of states in the cycles, including all their copies. The divisor has $6g$ address cycles, one plane cycle and C_2. Every coordinate cycle length Q_j is a product of distinct primes assigned only to that coordinate. Thus the lengths of A are squarefree and pairwise coprime. Every length in $B = AY(0)$ is the least common multiple of squarefree lengths and is also squarefree. By (4), each active site v contributes a distinct length $2d_v$ to B. Their number grows with the number of variables.

For $u > 3$, minimality gives $(u - 1)^{u-1} < t$, hence $u \log u = O(\log(t + 2))$, including $u = 3$ after changing the constant. Also $g \leq u$. The $6gu + 3$ address and plane primes have largest value $p_* = u^{O(1)}$ by the bound on the ith prime $p_i = O(i \log(2i))$ [18, p. 69, (3.13)]. They can be found by deterministic sieving. An address cycle length is at most p_*^u, and the plane cycle length is at most p_*^3. Each site length uses one label from each coordinate, hence is at most p_*^{6g+1}. Each of these bounds is $t^{O(1)}$.

There are $D = 6g + 1$ coordinates and $|E| = 4(u + 1)^{6g} = t^{O(1)}$ faces. Since all capacities are at most one, (3) gives

$$|B| \;\leq\; 2|A|\, 2^{6g+1}|V|p_*^{6g+1} = t^{O(1)}.$$

This counts all copies of all cycles of B. The algorithm enumerates the tuples, constructs the copying trees, marks the lines of the input formula, computes (2) and multiplies $AY(0)$ by the cycle rule (1). All loops take time polynomial in the input size, including the number of clauses. In particular, $|z_e(0)| \leq W_e \leq |V|p_*^{6g+1}$ bounds the signed intermediate coefficients before multiplication. Writing a cycle of length d uses d consecutive states with cyclic successors. Both successor tables together require $O(N \log(2 + N))$ bits for $N = |A| + |B|$. This proves Theorem 1.

The three one-bit intervals of Section 2 also admit a direct check. Building their successor tables and recovering the quotients gives two classes for $[0, 3]$, one for $[3, 3]$ and none for $[1, 2]$. Section 5 gives the table sizes and describes the independent decoder.

3.4 Counting quotient classes

Corollary 3. *For explicitly given permutations, counting the isomorphism classes of X satisfying $AX = B$ is #P-complete under parsimonious polynomial-time reductions. The hardness holds with all the cycle restrictions of Theorem 1. All instances produced by the reduction also satisfy $A \mid (B + B)$.*

Proof. Counting satisfying assignments to clauses of width at most three is #P-complete under count-preserving reductions [19, §3, Facts 1–2]. In the bounded-fan-in circuit encoding, fields for internal gate values have a unique assignment given the inputs. Each gate relation uses at most three variables, and a unit clause requires the output to be true. Our input convention explicitly accounts for every variable counted. The tautology preprocessing and clause padding preserve all assignments, including those of an initially empty conjunction. The fixed empty-conjunction instance has one quotient class, and the empty-clause instance has none. Otherwise each satisfying assignment has exactly one parity extension to the copying trees. Conversely the bits $b_{1,x}$ recover the assignment, and Lemma 2 gives exactly one quotient class for each feasible array. Doubling the half assignment, or using the displayed products in the fixed cases, gives $A \mid (B + B)$.

To count quotient classes, each class must have exactly one accepting certificate. Guessing a labelled successor table could count the same permutation many times. Its list of cycle multiplicities is unique. If $|B|/|A|$ is not integral, reject. Otherwise set $n_X = |B|/|A|$ and guess the multiplicity of each cycle length $1, \ldots, n_X$, using exactly $\lceil \log_2(n_X + 1) \rceil$ bits per entry. Check that the total size is n_X, and use the cycle rule to compare the product with the cycle histogram of B. This is polynomial in the explicit table sizes. Every quotient class yields exactly one accepting string. □

A procedure that halts in polynomial time on every input and returns a quotient on every yes-instance would decide 3-SAT in polynomial time. For the constructed permutation instances, an output can be checked by testing bijectivity and comparing the cycle multiplicities of AX and B.

4 The cost in states

How many states do we pay for each bit in the reduction? Consider a nontrivial formula. We use the guard prime 2, and each active site has capacity one and interval $[0, 1]$. Put $V^+ = \{v : \kappa_v = 1\}$. Keeping just the terms for these sites in the exact state count (3) gives the lower bound below. The upper bound from the same lemma includes all faces.

$$2|A| \sum_{v \in V^+} d_v \;\le\; |B| \;\le\; 2^D\, 2|A| \sum_{v \in V^+} d_v.$$

In each construction 2^D grows more slowly than any fixed positive power of the input scale. It therefore suffices to count the divisor states, the length d_v at an active site, and the active sites. For example, the first row of Table 1 gives $2 + 12 + 3 = 17$ for the exponent of $N = |A| + |B|$.

4.1 Size of the duplicated-digit construction

Let $t = 2n + 1$ count the identifiers. For the construction in Section 3, minimality of u and g gives $\log t \sim u \log u$ and $g \log u = \log t + O(\log u)$, so $g \sim u$. There are $r = 6g$ address

Construction	Scale M	$\lvert A\rvert$	d_v	$\lvert V^+\rvert$	N
Duplicated digits	$2n+1$	2	12	3	17
One checksum	$2n+1$	0	6	3	9
Test and literal addresses	$n+m$	0	4	1	5

Table 1: State counts for n variables and m padded clauses. In the last four columns, an entry a means the quantity is $M^{a+o(1)}$ as M grows. The estimate for d_v holds uniformly at all active sites. In particular, a zero allows subpolynomial growth.

coordinates. The largest prime p_* is at most $u^{2+o(1)}$ by the prime bound in Section 3.3. The groups receive consecutive primes in coordinate order. Thus every prime in group $j \geq 2$ is at least $(j-1)u$. The last address cycle alone has length at least $((r-1)u)^u$. Each address cycle has length at most p_*^u, and hence $|A| = t^{2+o(1)}$. At every site the same ordering gives

$$u^{r-1}(r-1)! \;\leq\; d_v \;\leq\; p_*^{r+1}, \qquad d_v = t^{12+o(1)}$$

by Stirling's formula. There are exactly $6n(2u-1)^{2g} = t^{3+o(1)}$ active sites. Since $D = r+1$ gives $2^D = t^{o(1)}$, the two bounds on $|B|$ yield $N = t^{17+o(1)}$, uniformly over the clauses and the choice of copying trees.

4.2 A shorter address with one checksum

Can we still isolate two endpoints on a copying line with fewer coordinates? Duplicating every digit used $2g$ coordinates for a word of length g. We will use just one extra digit to record a checksum, bringing the cost down to $g+1$.

We also shrink the alphabet. A group with u labels costs up to p_*^u states in a single divisor cycle. We would like that cost to be $t^{o(1)}$, while retaining enough words to name all t identifiers. Let $\lambda \geq 3$ be the least integer with $\lambda^\lambda \geq t$, set $u = \max(3, \lceil \lambda / \lceil \log_2 \lambda \rceil \rceil)$, and take the least $g \geq 1$ with $u^g \geq t$. Dividing the alphabet size by a logarithmic factor makes $u = o(\lambda)$, yet $\log u \sim \log \lambda$. Thus the required word length still satisfies $g \sim \lambda$.

For a word $a = (a_1, \ldots, a_g)$, put $\sigma(a) = \sum_i a_i \bmod u$. A varying slot can occupy any of the $2u^g$ sites

$$S(a) = (a, \sigma(a)), \qquad I(a) = (a, \sigma(a) + 1 \bmod u).$$

Think of $S(a)$ as a stable address and $I(a)$ as the intermediate site on a move to the next address. In base three, for instance, a move from $S(2,1)$ to $S(0,1)$ visits $(2,1,0), (2,1,1), (0,1,1)$. Give every identifier a distinct word, including the padding identifier. Its address is the corresponding S site. On plane j fix slot j to the stable address of its literal and let the other two slots vary independently. Sign and clause tests still take place at stable addresses.

Join $S(a)$ to $I(a)$. From $I(a)$ one can move to $S(a')$ whenever a' is obtained by increasing one digit of a by one modulo u. This second edge keeps the checksum fixed, even when the digit wraps around. Why do we see exactly two possible sites on each coordinate line? If a digit varies, the other digits and the checksum determine it uniquely for each of the two types. If the checksum varies, the word fixes its two possible values. Hence an edge line in one copy isolates its two endpoints.

These edges form a connected bipartite graph on the slot, since a path of length two increments any digit of a stable site and each edge joins an S site to an I site. In the product of the two varying slots, the parity of the number of I slots changes along every edge. In particular, all pairs of stable addresses have even parity. As in Section 3, impose complementary bits along a spanning tree. Each literal bit extends uniquely, the six sign tests still identify the copies of a variable, and every clause still reads its three literal bits. Assigning $1/2$ at every active site remains feasible. Lemma 2 therefore gives the same assignment–quotient bijection and cycle restrictions.

For the state count, put $r = 3(g+1)$, so $D = r+1$. We have $\log t \sim \lambda \log \lambda$. Assign consecutive primes in coordinate order as before. The largest prime has index $O(gu)$, so $\log p_* \le (2+o(1)) \log \lambda$. The r address groups each cost at most p_*^u states and the plane group at most p_*^3. Since $u = o(\lambda)$, this gives $\log |A| = o(\log t)$ as intended. At every site we again have

$$u^{r-1}(r-1)! \le d_v \le p_*^{r+1}.$$

By Stirling's formula the logarithms of the two bounds are both $(6+o(1)) \log t$. The $6n$ literal copies each have $(2u^g)^2$ active sites, giving $t^{3+o(1)}$ in total. Since $2^D = t^{o(1)}$, we obtain $N = t^{9+o(1)}$, uniformly over formulas and copying trees. Enumerating the $4(u+1)^r = t^{3+o(1)}$ faces and writing the tables as in Section 3.3 takes polynomial time in the formula length. The estimates count states. Writing their successors in binary adds at most a logarithmic factor.

4.3 Routing by test and literal

In the checksum construction a literal occupies an entire product of two slot graphs on a plane, although a test of that literal on the plane reads it at just one site. To reduce the cost further, we give separate addresses to tests and literals. A *test* is a clause or one of the six sign tests of a variable. At stage zero a test keeps its address fixed and follows a path from padding to the literal it reads. At stage one the literal keeps its address fixed, and a tree joins the tests that read it. A single edge between the two stages at each such meeting point makes all occurrences of that literal on the plane read the same bit. Only the paths and trees needed to deliver those bits have to be activated. Figure 3 follows two tests that request the same literal.

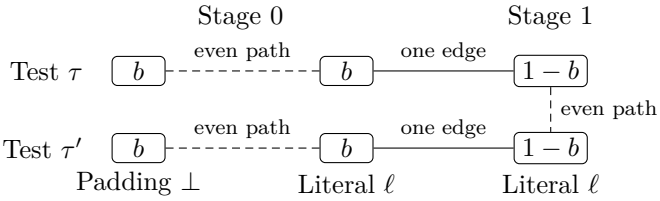

Figure 3: Two tests on one plane request the same literal ℓ. Each stage-zero path varies the literal address at a fixed test. A single stage edge then reverses the bit. The stage-one tree joins the two stable test addresses at fixed ℓ. Its path has even length because stable and intermediate sites alternate. Dashed lines stand for paths with their internal sites omitted. Both tests therefore read b at padding.

Use two address slots, a stage coordinate with two labels and a plane coordinate with three labels. Give distinct identifiers to the m clauses and the $6n$ sign tests of Section 3. A clause requests one literal on each of its three planes, even if it repeats a literal. A sign test requests a literal on two planes. There are $m+6n$ test identifiers and $2n+1$ literal or padding identifiers. Thus a common address space of capacity $T = \max(m+6n, 2n+1)$ suffices for either slot. Choose λ, u, g as in the checksum construction with T in place of t. Each slot uses its own prime labels on the same abstract checksum graph with $g+1$ coordinates. Write $S(\alpha)$ for the stable address of identifier α, including $S(\perp)$ for padding in the literal slot.

On each plane give the following sites capacity one, and give every other site capacity zero.

- At stage zero, fix the test slot to $S(\tau)$ for each participating test τ. In the literal slot take a path from $S(\perp)$ to $S(\ell)$, where ℓ is the literal requested on this plane. Mark its edges.
- At stage one, fix the literal slot to $S(\ell)$ for each requested literal ℓ. In the test slot choose one test requesting ℓ and join its stable address to those of all the others. Take a spanning tree of the union of the paths and mark its edges. If only one test requests ℓ, the tree is a single site.

For both stages we route between stable addresses by incrementing digits in coordinate order. There are at most $u-1$ increments per digit and two edges per increment, hence at most $2g(u-1)$ edges per route. Use a fixed vertex order to select each spanning tree.

Which active sites does each marked line read?

1. A stage-zero edge line fixes the plane, stage and test address, so only one path can meet it. Its two marked endpoints are active. The two-site property of the checksum graph excludes any further site.

2. A stage-one edge line fixes the plane, stage and literal address, so only one tree can meet it. The same two-site property isolates its endpoints.

3. For an occurrence (τ, ℓ), mark the stage line through $(S(\tau), S(\ell))$ on its plane. It meets the end of the test path and a site of the literal tree. These are its only two sites, since the stage takes just two values. A path or tree may visit another stable address along its route. We mark the stage line there only if the address pair is itself a requested occurrence.

4. The plane line through $(S(\tau), S(\perp), 0)$ reads exactly the planes participating in τ. Only those planes have a stage-zero path for that test, and each such path starts at padding.

On the first three kinds of lines impose the complementary-bit interval for their two endpoint primes. On a plane line impose its sign or clause test. Every other face receives its default interval. Lines from different kinds free different coordinates or fix different stages. At a given stage, edge lines for different tests (at stage zero) or different literals (at stage one) have different fixed addresses. If two traversals use the same line, its two endpoints give the same interval.

For consistency, colour each active site by the parity of its stage plus the number of I addresses in its two slots. Every address edge changes one address type, and a stage edge changes the stage. Each complementary-bit equation therefore joins opposite colours. All the stable test addresses in a stage-one literal tree carry the same bit. Crossing to stage zero complements that bit once. The remaining path to padding has even length because both ends are stable. Thus every occurrence of ℓ on plane j reads the same bit $b_{j,\ell}$, the complement of the tree's stable bit. Conversely, specifying these readouts fills the trees and then the paths uniquely. Every literal is requested on each plane by the sign tests. They impose the same six-cycle as in Section 3, and the clause tests impose (5). We recover both the assignment–quotient correspondence and the feasible half array.

There are $r = 2(g+1)$ address groups followed by the stage and plane groups of sizes two and three. Assign consecutive odd primes in that order and retain the guard prime 2. Here $T = \Theta(n+m)$ and $g \sim \lambda$. As before, $|A| = T^{o(1)}$. At every site the bounds $u^{r-1}(r-1)! \le d_v \le p_*^{r+2}$ now give $d_v = T^{4+o(1)}$. The $3m + 12n$ participating test–plane pairs give distinct stage-zero padding sites. Each pair needs at most one route in each stage, so both stages together use $O((n+m)gu)$ sites. Hence $|V^+| = T^{1+o(1)}$. Since $2^{r+2} = T^{o(1)}$, the state-count bounds yield $N = (n+m)^{5+o(1)}$. There are $12(u+1)^r = T^{2+o(1)}$ faces, and the construction takes polynomial time including the explicit output. In particular, $N = n^{5+o(1)}$ when $m = O(n)$. For unrestricted clause density the three-slot construction gives the separate bound $N = (2n+1)^{9+o(1)}$.

5 Computational checks

These computations illustrate the reduction on instances small enough to check independently. For the one-bit example in Section 2, we built the three target successor tables. In the order of the intervals $[0,3]$, $[3,3]$ and $[1,2]$, they contain respectively 15 792, 11 844 and 13 160 states. From their cycle types we recovered two, one and zero quotient classes. For each quotient we built its product table with A and checked its cycle type against the target.

For the SAT constructions we stored each cycle length with its multiplicity. Equation (1) lets us multiply these lists exactly, even for the following targets built from the single clause $(x \vee \neg x \vee x)$.

Construction	Distinct cycle lengths in B	$\lvert B\rvert$
Duplicated digits	27 693	329 236 589 030 252 537 126
Test and literal addresses	58 868	42 733 726 046 144 284 221 840

The clause is a tautology, so a decoder should find two quotient classes. Replacing it by $(x \vee x \vee x)$ leaves one assignment. Adding $(\neg x \vee \neg x \vee \neg x)$ to that clause leaves none. A separate decoder receives just the cycle lists of A and B. It factors the lengths in A to recover the prime groups and reads the coefficients in B to reconstruct the interval constraints. It then enumerates the feasible bits without access to the formula. For the duplicated-digit construction, which uses 150 active sites for one variable, it reconstructed the constraints, found respectively two, one and zero quotient classes, and verified each quotient by multiplication. For the two-address construction in Section 4.3, the decoder reconstructed all 49 152 faces on each of the three inputs and found the same counts.

For seven formulas with one to five variables we also compared the assignments obtained from the two-address interval constraints with direct Boolean enumeration. The two sets agreed for every formula. The formulas, scripts and per-instance counts are in the reproduction package (commit `f7145d2`).

6 Scalar polynomials and uniqueness

What can we learn about X by observing AX after k steps? A pair returns to its starting point exactly when both coordinates do. If $\phi_k(X)$ counts the states fixed by the kth iterate, then $\phi_k(AX) = \phi_k(A)\phi_k(X)$. On a d-cycle all d states return when $d \mid k$, and none return otherwise. This gives the formula $\phi_k(Z) = \sum_{d|k} d z_d$, which extends to every rational cycle sum $Z = \sum_d z_d C_d$. Multiplicativity extends to these sums by bilinearity. In the language of Burnside rings, this count is the mark of the subgroup generated by the kth power of a generator of the infinite cyclic group acting by the permutation [7, §2, (2-6)]. The marks determine Z one cycle length at a time. Once the shorter lengths are known, subtracting their contributions from $\phi_k(Z)$ leaves $k z_k$.

Call k *visible* when $\phi_k(A) > 0$, that is, when some cycle length of A divides k. Division by $\phi_k(A)$ then recovers the mark of X. At every other index the product mark is zero for every X. Thus $AZ = AZ'$ exactly when Z, Z' agree in all visible marks. For the opening example $A = C_2$, only the even indices are visible. Both candidate quotients $2C_1$ and C_2 have two returning states at every even step. Their numbers of fixed points differ at the first step, which the product cannot detect.

Corollary 4. *Fix a nonconstant scalar polynomial $q(z) = \sum_{i=0}^{r} c_i z^i$ with nonnegative integral coefficients and $c_r > 0$. Interpret a coefficient c_i as $c_i C_1$ and write $\mathcal{Q}$ for this polynomial on systems. For explicit permutations A, B, deciding $A\mathcal{Q}(X) = B$ is NP-complete and counting its quotient classes is #P-complete under parsimonious reductions. Hardness holds under the cycle restrictions of Theorem 1, with A part of the input.*

Proof. We will replace the target of each hard division instance by a target $B_{\mathcal{Q}}$ with exactly the same solutions X. The reason this can work is that marks add under disjoint union and multiply under direct product. Hence $\phi_k(\mathcal{Q}(X)) = q(\phi_k(X))$, so we can apply q to each visible mark separately.

For the constructed instance let $Y_0 = Y(0)$ and $Y_* = Y(\bar{x})$ at the feasible half assignment. Then Y_0 is signed integral, Y_* is nonnegative rational, and $AY_0 = AY_* = B$. In the constant yes branch use $Y_0 = Y_* = C_1$. In the constant no branch of Section 3 one may take $Y_0 =$

$-3C_1 + 2C_2 + C_3$ and $Y_* = (C_2 + C_6)/2$, which have the same stated properties by (1). Form $B_{\mathcal{Q}} = A\mathcal{Q}(Y_0)$. On visible indices its marks are $\phi_k(A)q(\phi_k(Y_*))$, and on other indices they vanish. Hence $B_{\mathcal{Q}} = A\mathcal{Q}(Y_*)$. Its cycle multiplicities are integral by the first construction and nonnegative by the second. Its size is $|A|q(|B|/|A|) > 0$. Its cycle lengths are squarefree, since direct products take least common multiples. For every permutation X, strict increase of q on the nonnegative reals gives

$$A\mathcal{Q}(X) = B_{\mathcal{Q}} \iff \phi_k(X) = \phi_k(Y_*) \text{ for every visible } k \iff AX = B.$$

To compute $B_{\mathcal{Q}}$ we enumerate the ordered tuples of terms of Y_0 in each signed product. There are polynomially many such tuples for fixed r, and each coefficient and cycle length has polynomial bit length. The displayed size of $B_{\mathcal{Q}}$ is polynomial, so we can write out its successor table.

For membership, $1 \le |X| \le |B|$ on a yes-instance. The summand AX^r in a permutation target forces X to be a permutation, since a positive Cartesian power of a nonsurjective map is nonsurjective. One fixed-width guess for each entry of the cycle histogram up to $|B|$ gives one accepting path per solution. For a guessed histogram of total size n_X, first check that $1 \le n_X \le |B|$ and $|A|q(n_X) = |B|$. Reject the guess if either check fails. For each i with $c_i > 0$, the summand $c_i X^i$ then has $c_i n_X^i \le q(n_X) = |B|/|A|$ states. Evaluate the fixed polynomial using the cycle product rule and compare the cycle type of $A\mathcal{Q}(X)$ with that of B. □

Suppose now that we know there is at most one quotient. How hard is it to decide whether it exists? Another question starts with an arbitrary instance and asks whether it has at most one quotient. The bijection with satisfying assignments answers both.

Corollary 5. *If a deterministic polynomial-time algorithm distinguishes zero quotient classes from exactly one on promised explicit permutation instances, then* NP = RP. *Deciding whether there is at most one quotient class is coNP-complete. Both statements hold under the divisor and target restrictions of Theorem 1.*

Proof. Apply Valiant–Vazirani isolation to a SAT instance [20]. An unsatisfiable formula always stays unsatisfiable. A satisfiable one acquires exactly one solution with inverse-polynomial probability. Use bounded-fan-in Boolean gates and a biconditional for each gate when converting to clauses of width at most three. Each input assignment then has exactly one gate extension if it is accepted. The parsimonious reduction preserves the promise in the two cases just described. Polynomially many independent repetitions give RP success probability. Answers on multiple-solution outputs cannot cause a false acceptance on an unsatisfiable input.

For the second claim adjoin a fresh variable through a tautological clause. The number of quotient classes is twice the number of satisfying assignments. It is at most one exactly on unsatisfiable formulas. Two distinct verified cycle histograms certify the complement. □

The promise here concerns a single pair A, B. Global injectivity of the polynomial across all right-hand sides is a separate tractable condition [15].

How many divisor cycles suffice? Our reduction uses a growing number of divisor cycles. What happens with just two? For $A = C_a + C_b$ with $a = p^\alpha$ and $b = q^\beta$ powers of distinct primes, $\operatorname{lcm}(a, b)$ has $(\alpha + 1)(\beta + 1)$ positive divisors. The counting bound of Bridoux et al. therefore applies with exponent four for two primes and nine for their squares, even as p, q vary [2, Thm. 1]. Now allow arbitrary lengths a, b with neither dividing the other. Can one polynomial-time algorithm decide whether $C_a + C_b$ divides an explicit permutation B, even when $\operatorname{lcm}(a, b)$ may have arbitrarily many divisors? Our reduction leaves this question open.

References

[1] R. Ascone, A. E. Porreca, and M. Rolland. *Polynomial equations over bijective dynamical systems and their semiring extensions*. Extended abstract, Workshop on Cellular Automata and Boolean Automata Networks, UCNC 2026. doi:10.5281/zenodo.20431158. Accompanying slides, nos. 17–21.

[2] F. Bridoux, C. Crespelle, T. H. D. Phan, and A. Richard. Dividing sum of cycles in the semiring of functional digraphs. *Natural Computing* 25, article 11, 2026. Publisher. arXiv:2504.11943v2.

[3] J. De Loera and S. Onn. The complexity of three-way statistical tables. *SIAM Journal on Computing* 33, 819–836, 2004. doi:10.1137/S0097539702403803.

[4] A. Dennunzio, V. Dorigatti, E. Formenti, L. Manzoni, and A. E. Porreca. Polynomial equations over finite, discrete-time dynamical systems. In *Cellular Automata (ACRI 2018)*, LNCS 11115, 298–306, 2018. Publisher.

[5] A. Dennunzio, E. Formenti, L. Margara, and S. Riva. On solving basic equations over the semiring of functional digraphs. *Discrete Mathematics & Theoretical Computer Science* 27(3), article 6, 2025. doi:10.46298/dmtcs.14590.

[6] F. Doré, K. Perrot, A. E. Porreca, S. Riva, and M. Rolland. Infinite trees for division and roots over finite discrete-time dynamical systems. *Natural Computing* 25, article 16, 2026. doi:10.1007/s11047-026-10068-7.

[7] A. W. M. Dress and C. Siebeneicher. The Burnside ring of the infinite cyclic group and its relations to the necklace algebra, λ-rings, and the universal ring of Witt vectors. *Advances in Mathematics* 78, 1–41, 1989. Publisher.

[8] M. Gadouleau and M. Johnson. Semirings of formal sums and injective partial transformations. arXiv:2603.26508, 2026.

[9] M. Gadouleau, S. Riva, M. Rolland, and M.-E. Voge. Solving polynomial equations over partial discrete dynamical systems. arXiv:2609.39724, 2026.

[10] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pp. 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

[11] É. Naquin and M. Gadouleau. Factorisation in the semiring of finite dynamical systems. *Theoretical Computer Science* 998, 114509, 2024. Publisher. Preprint, 2022, §7, p. 20.

[12] T. H. D. Phan, C. Crespelle, and A. Richard. Tensor product of functional graphs: algorithms for factorization and polynomial root-finding restricted to the cycle part. SSRN 5316210, 2025.

[13] A. E. Porreca. The semiring of dynamical systems. Author's account, 28 June 2020, and CANA seminar slides, 6 October 2020, PDF pp. 91–94.

[14] A. E. Porreca and M. Rolland. Solving pseudo-injective polynomial equations over finite dynamical systems. In *AUTOMATA 2025*, LNCS 15831, 171–187, 2025. Publisher. arXiv:2504.06986.

[15] A. E. Porreca and M. Rolland. Injective and pseudo-injective polynomial equations: from permutations to dynamical systems. arXiv:2604.04065v2, 2026. See also the CiE 2025 paper on injectivity.

[16] S. Riva. *Factorisation de systèmes dynamiques discrets*. Doctoral thesis, 2022. University repository.

[17] M. Rolland. *Des unrolls aux systèmes dynamiques totaux et partiels : algorithmes de résolution et énumérations des (in)équations polynomiales*. Doctoral thesis, Aix-Marseille Université, 2026. NNT 2026AIXM0179. Author's copy.

[18] J. B. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics* 6, 64–94, 1962. doi:10.1215/ijm/1255631807.

[19] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing* 8, 410–421, 1979. doi:10.1137/0208032.

[20] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science* 47, 85–93, 1986. Publisher.