

FORMAL SUPPLEMENT TO UNIVERSAL RANK TESTS FOR NONCOMMUTATIVE RATIONAL FORMULAS

MICHAEL SCHROEDER
INDEPENDENT RESEARCHER

ABSTRACT. This supplement records the exact formal contract, its semantic predicates, the correspondence of every numbered result with the Lean development, and explicit height and serialization certificates. It also supplies a parameter glossary and reproduction instructions. The source excerpts are checked against the distributed Lean files. The main theorem and its encoding are unchanged; the supplement distinguishes the written mathematics from implementation details and classical results outside the formalization.

S1. CORRESPONDENCE AND SCOPE

Numbers without an S prefix refer to the main paper. The authoritative proof is the distributed Lean project, pinned to Lean 4.34.1 and the dependency revisions in `lake-manifest.json`. Its unconditional main theorem is `TC10.Runtime.MainGenerator.main`. The named witnesses in `TC10.Runtime.FixedGenerator.specification` are shared by all coefficient-field universes. No arithmetic, honesty, hitting or runtime assumption is supplied by the user of that theorem.

Table 1 covers all 18 numbered mathematical results. “Paper proof” describes the argument presented in the paper, not an independent referee certification. “Lean implementation” marks the low-level arithmetic or compiler proof whose details are in the artifact. Classical citations and the general-tree strengthening are explicitly separated from the formal main theorem. The more detailed `PROOF_MAP.txt` records the source file for each declaration.

Date: 18 October 2026.
ORCID: [0009-0004-3249-0195](https://orcid.org/0009-0004-3249-0195).
DOI (reserved): [10.5281/zenodo.23453468](https://doi.org/10.5281/zenodo.23453468).

Table 1: Complete correspondence. Declaration names have prefix TC10..

Result	Checked declarations	Paper treatment and limits
Definition 2.1	<code>ExprMatrix.Admissible</code> <code>UpperRank</code> <code>Attains</code> <code>intrinsicRank</code> <code>structural_rank</code>	Definitions. Lean uses an integer upper bound and attained-rank witness; its structural theorem supplies the supremum interpretation.
Theorem 2.1	<code>MainStatement</code> <code>Runtime.MainGenerator.main</code> <code>Runtime.FixedGenerator.specification</code> <code>PaperContract.main</code>	Paper assembly; Lean implementation. One named machine and bounds are independent of coefficient universes. The expanded contract includes actual runtime and output.
Corollary 2.2	<code>scalar_corollary</code>	Paper proof. The main theorem discharges the simultaneous-rank premise of the checked scalar corollary.
Corollary 2.3	<code>SimultaneousRanks</code> <code>Runtime.FixedGenerator.specification</code>	Paper proof with classical citations. The operational statement is checked. Identification with abstract free-field rank is not newly formalized.
Proposition 2.4	<code>UniversalLowerBounds.list_length</code> <code>UniversalLowerBounds.naming_bound</code> <code>UniversalLowerBounds.signatures_injective</code>	Paper and Lean proofs. The eigenvalue argument uses arbitrary coefficient extensions and the shared size budget; the four-leaf matrix forces distinct signatures without inverses. Neither bound assumes unit or conjugate outputs.
Lemma 3.1	<code>QuantumMatrices.fourier_reconstruction</code> <code>QuantumLift.expression_rank_lift</code> <code>PermittedLift.expression_rank_lift</code> <code>DivisionRank.block_dimension_dvd_rank</code>	Paper proof by quantum-plane localization. Lean constructs a Hahn-series division algebra; both provide strict lifting, rank monotonicity and divisibility.
Lemma 3.2	<code>domain_and_unit_witnesses</code> <code>permitted_domain_witness</code> <code>permitted_unit_witness</code>	Paper proof and checked induction. Permitted domain witnesses precede nonzero-to-invertible witnesses at each height.
Lemma 3.3	<code>GenericEvaluation.strict_lift</code> <code>GenericEvaluation.rank_lift</code> <code>simultaneous_generic_rank</code>	Paper proof by domain polynomials. Lean uses generic matrices and local rings; common witnesses are proved, not assumed.

Result	Checked declarations	Paper treatment and limits
Theorem 3.4	<code>structural_rank</code> <code>simultaneous_rank_with_minors</code>	Paper proof and checked structural rank. Integral optimal ranks and simultaneous invertible block minors are obtained.
Proposition 3.5	<code>Formula.inverseNodes_iff_defined</code> <code>Formula.certificates_weight_budget</code> <code>family_certificates_bounded_order</code>	Paper proof and checked gate/domain induction. Exact certificate orders retain the original strict inverse obligations.
Lemma 4.1	<code>Runtime.PrimeNames.length_bound</code> <code>Runtime.PrimeNames.functions_separate</code> <code>Runtime.PrimeNames.construct_poly</code>	Paper proof follows the executed prime-field selector. The abstract <code>WideNames</code> construction has a different name field; the executed declarations give the saved-log bound.
Lemma 4.2	<code>RelativeCyclicCoding.honest</code> <code>TriangularDictionary.relative</code> <code>TriangularDictionary.term_value_path</code> <code>ShortHonesty.honest_dictionary</code> <code>ConjugateCoding.code_size_exact</code> <code>ShortWords.leaf_size_exact</code> <code>ShortWords.size_le_exact_cost</code>	Paper proof follows finite cyclic and triangular dictionaries. A unit rank witness transfers composite values to literal trees; arbitrary-input domains are not identified. <code>ExactCodeSize</code> also checks the literal cost $5 w +5$ and sharper local substitution bound.
Lemma 5.1	<code>StarOverlap.row_bound</code> <code>StarOverlap.overlap</code>	Paper proof for general trees. Lean checks the star case used by the generator; the general-tree strengthening is not separately formalized.
Lemma 5.2	<code>FoldedDeterminant.full_window_bound</code> <code>FoldedDeterminant.spaced_full_window_bound</code>	Paper proof and checked extended-window multiplicities. Node distinctness and boundary contributions are explicit in every characteristic.
Proposition 5.3	<code>SquareQuotient.geometric_design</code> <code>SquareQuotient.rank_and_threshold</code> <code>BoundedQuotientDesign.rank_of_full_kernel</code>	Paper proof and checked star quotient. The projection has the full kernel and remains valid after every scalar extension.
Lemma 6.1	<code>CompanionCurve.determinant</code> <code>CompanionCurve.reduced_determinant</code> <code>DeterminantCurve.nonzero</code> <code>DeterminantCurve.degree_bound</code>	Paper proof and checked companion construction. The even-order sign, determinant identity, nonvanishing and degree bound are checked.

Result	Checked declarations	Paper treatment and limits
Proposition 6.2	<code>SampledDescent.sample_then_descend</code> <code>CompanionSampling.pencil_sample</code>	Paper proof and checked sampling/descent. A common sample works in all geometric components and for every coefficient extension.
Lemma 7.1	<code>BoundedQuotientDesign.rational_square_bits</code> <code>LiftedCodeHeight.code_entry_bits</code> <code>RationalWordHeight.normalized_height</code> <code>RationalWordHeight.code_height_bound</code> <code>Runtime.RationalGenerator.data_poly</code>	Stagewise estimates in the paper; explicit integer certificates and executed arithmetic in Lean. The displayed asymptotic raw-height estimate is not an identity with every internal Lean bound.

S1.1. Dependency structure. The mathematical assembly has the following dependencies, with classical rank identification added only for Corollary 2.3:

$$\{3.1, 3.2, 3.3\} \longrightarrow 3.4 \longrightarrow 3.5,$$

$$\{3.3, 3.4\} \longrightarrow 4.2,$$

$$\{5.1, 5.2\} \longrightarrow 5.3,$$

$$\{3.1, 5.3, 6.1\} \longrightarrow 6.2,$$

$$\{3.5, 4.1, 4.2, 6.2, \S 7\} \longrightarrow 2.1 \longrightarrow \{2.2, 2.3, \text{S5 corollary}\}.$$

Proposition 2.4 gives independent necessary conditions for the semantic contract, without using the construction. The compiler and exact encoding additionally supply the machine clause of Theorem 2.1. They do not replace the domain and rank arguments.

S1.2. The characteristic-two example. Definition 2.1 allows all positive dimensions. The odd dimensions called permitted in Section 3 are needed for the auxiliary division lift. Thus the two-dimensional example is a valid operational-rank witness. For clarity, it has no one-dimensional domain witness over any extension. An explicit route to a permitted witness works first with the inverse-free operand $g = xy + yx$: append the scalar block 1 to both matrices, giving $g(U \oplus 1, V \oplus 1) = I_2 \oplus 0 \neq 0$ in dimension three. After adjoining a primitive cube root, Lemma 3.1 preserves this nonzero value inside a division algebra. It is therefore invertible there, and only then can one evaluate g^{-1} . This is an existence argument, not a claim that the illustrative pair is a printed generator output.

S2. PARAMETER GLOSSARY

The first-use column refers to the main paper. Local symbols in source excerpts retain their Lean spelling; in particular a local word length k is not the global machine-runtime exponent.

Symbol	Meaning	First use
n, s, h	Named variables, total entry-tree size and maximum inversion height.	2
p, K_p	Zero or prime characteristic; its prime field.	2.1
$\Lambda(t)$	$\lceil \log_2(t+2) \rceil$, including $t = 0$.	2.1
D, L	Actual matrix dimension and number of retained output tuples.	2.1
C, k	Absolute positive output-size and machine-time constants.	2.1
$\tau(M)$	Maximum normalized rank on the original strict domain, over extensions.	2
Ω, S	Diagonal clock and cyclic shift in the division lift.	3
λ, μ	Independent central parameters in that lift.	3
b, π	$b = 2^{\Lambda(s)}$; naming prime $16s^2b \leq \pi \leq 128s^3$.	4
m_*	Power-of-two palette $2^{\Lambda((4sb)^2)} \leq 514s^4$.	4
t, H	Naming values $\lfloor \Lambda(n)/\Lambda(s) \rfloor + 1$ and $8s^2t$.	4
$\mathcal{N}$	Separating family of maps from the n names to the palette.	4
g_i	Positive binary word of length $\Lambda(m_*) + 1$ ending in u .	4
q, w	Certificate-order bound; homogeneous core variable count, ultimately $w = 3$.	5
η, ν, δ	qw , $(w-2)q+1$, and $w\nu$.	5
r	Permitted cyclic order with $r > q\delta$.	5
d_0, ℓ	$2r - \nu$ and $r + \eta - 1$: coefficient rows and window spacing.	5
γ, α_j	High-order design element and $\gamma^{(j-1)\ell+\eta-1}$.	5
c	Number of quotient rows before padding to r rows.	5
ζ, k_0, κ	Primitive r th root, $q(r-1)+1$, and $(r-1)(k_0+1)$.	6
e, A, K	Dimension of the étale algebra A/K_p ; design/sample field K .	6–7
W	Sum of certificate orders; in assembly $W = q(h+2) + 1$.	6,8
$W_{\max}$	Capacity $4q^2$ used for field choices, independent of h .	7
m_2	$2\Lambda(q) + 5$; at $p = 2$, $r = 2^{m_2} - 1$ and $e = 4m_2$.	7
N_0, M, B	In odd characteristic: $r/2$, $6(r+3q-1)$, and $\binom{M}{2}$.	7
T_*	Capacity threshold $4q^2\kappa + B + 1$.	7
E_0, E, P	$2\lceil \log_2(2(T_*+1)) \rceil$, $\max(N_0, 2E_0)$, and $ET_* + 2$.	7
J_*	$eW\kappa + 1$: number of nonzero sample candidates.	7
H_{raw}	Raw rational sample height estimate in (7.4).	7

Symbol	Meaning	First use
b, b_{dec}	Local common-denominator height certificates for normalized and decoded matrices.	7
k_{word}	Maximum conjugator length in the explicit serialization bound.	7
$q = 300s\Lambda(s)$	Uniform order budget after renaming, coding and guards.	8

The bound $r \leq 12q^2$ for the odd/rational order follows from $r \leq 2 \cdot 3q(q+1)$. The implemented order and the conservative global constants in the main theorem are unchanged by this observation.

S3. HEIGHT CERTIFICATES AND DENSE SERIALIZATION

Write $H_b(M)$ if $M = N/z$ for an integer matrix N and a nonzero integer z , with every $|N_{ij}|$ and $|z|$ at most 2^b . The shared denominator is essential: an entrywise fraction bound alone does not give the same product estimate. For $D \times D$ matrices, the checked rules give

$$H_a(M), H_b(N) \implies H_{a+b+\Lambda(D)}(MN).$$

If integer raw matrices Z_0, Z_1, Z_2 have entries at most 2^H in absolute value and survive the invertibility guards, put

$$b = D(H + \Lambda(D)) + 1, \quad b_{\text{dec}} = 2(k_{\text{word}} + 1)(b + \Lambda(D)).$$

Cramer's rule supplies H_b for $Z_0^{-1}Z_i$ and for its inverse $Z_i^{-1}Z_0$. Products along a conjugator and its reverse therefore supply $H_{b_{\text{dec}}}$ for every decoded coordinate. Reduced numerators and denominators have magnitude below $2^{b_{\text{dec}}+1}$.

For L tuples with n coordinates, the literal rational output length is at most

$$(S3.1) \quad 2\Lambda(D) + 2\Lambda(L) + 2 + LnD^2(4(b_{\text{dec}} + 1) + 3).$$

The first three terms cover the dimension/list-length header. Each entry has one sign bit and two self-delimiting natural-number encodings. The factor n includes all copied variable names; no sparse or implicit output representation is substituted. These are output-length bounds. The independent compiler argument proves the full machine-time bound; no numerical value of its existential exponent is asserted here.

The paper gives the stages $O(r^2)$, $O(r^3)$, $H_{\text{raw}} = O(r^3 + \kappa \log(J_* + 1) + \log(r + 1))$, followed by the normalization and word bounds above. The checked integer bounds can be looser. In particular `LiftedCodeHeight.pair_height` feeds the implemented raw bound into both directions of normalization; `LiftedCodeHeight.code_entry_bits` supplies actual entry bounds. This does not assert that the paper's particular asymptotic expression is definitionally the internal Lean certificate.

S3.1. Exact checked height declarations. The following excerpts are in namespace `TC10.RationalWordHeight`, with ambient matrix dimension `d : Nat`. The local `k` in the last excerpt is a word-length bound. Theorem proof bodies are in the cited source; the displayed hypotheses and conclusions are verbatim.

Source: `TC10/RationalWordHeight.lean`.

```
def HasHeight (b : ℕ) (M : Mat ℚ d) : Prop :=
  ∃ (A : Mat ℤ d) (z : ℤ), z ≠ 0 ∧
    (∀ i j, M i j = (A i j : ℚ) / z) ∧
    (∀ i j, |A i j| ≤ (2b : ℤ)) ∧ |z| ≤ (2b : ℤ)
```

Source: `TC10/RationalWordHeight.lean`.

```
theorem HasHeight.mul {a b : ℕ} {M N : Mat ℚ d}
  (hM : HasHeight a M) (hN : HasHeight b N) :
  HasHeight (a+b+logBound d) (M*N)
```

Source: `TC10/RationalWordHeight.lean`.

```
theorem normalized_height {H : ℕ} (A Z : Mat ℤ d) (hA : A.det ≠ 0)
  (ha : ∀ i j, |A i j| ≤ (2H : ℤ)) (hz : ∀ i j, |Z i j| ≤ (2H : ℤ)) :
  HasHeight (d*(H+logBound d)+1)
    ((A.map (Int.castRingHom ℚ))-1 * Z.map (Int.castRingHom ℚ))
```

Source: `TC10/RationalWordHeight.lean`.

```
theorem code_height_bound (U : Fin 2 → (Mat ℚ d)×) (b : ℕ)
  (hU : ∀ i, HasHeight b (U i : Mat ℚ d) ∧ HasHeight b (↑(U i)-1 : Mat
    ℚ d))
  (w : Word) : HasHeight (2*(w.length+1)*(b+logBound d)) (codeUnit U w :
    Mat ℚ d)
```

Source: `TC10/RationalWordHeight.lean`.

```
theorem decodeOutput_encoding {n : ℕ} (hd : 0 < d)
  (W : Fin n → Word) (Us : List (Fin 2 → (Mat ℚ d)×)) (hne : Us ≠ [])
  (b k : ℕ) (hW : ∀ i, (W i).length ≤ k)
  (hU : ∀ U ∈ Us, ∀ i,
    HasHeight b (U i : Mat ℚ d) ∧ HasHeight b (↑(U i)-1 : Mat ℚ d)) :
  (encodeOutput (decodeOutput .zero hd W Us hne)).length ≤
    2*logBound d + 2*logBound Us.length + 2 +
    Us.length*n*d*d*(4*(2*(k+1)*(b+logBound d)+1)+3)
```

S4. EXACT FORMAL CONTRACT AND FOUNDATIONAL PREDICATES

These excerpts retain the original source spelling and quantifiers, with namespace context stated alongside. Unicode symbols are mechanically escaped for portable typesetting using standard mathematical fonts. The excerpts are navigation material, not a replacement Lean module. Some theorem proof bodies and unrelated lemmas are omitted.

No proof hole or assumed axiom is inserted in the distributed project. `scripts/check_supplement.py` compares every displayed source block with its pinned source slice; `scripts/verify.py` also rebuilds the project and checks its transitive axioms.

S4.1. Trees, size, inversion height and strict evaluation. In namespace `TC10`, the universes are `u v w`. Inside `Formula`, `F : Type u` and the variable counts are natural numbers. Inverse evaluation explicitly requires both inverse identities; multiplication retains both child evaluations even at zero.

Source: `TC10/Syntax.lean`.

```
inductive Formula (F : Type u) (n : ℕ) : Type u where
  | var : Fin n → Formula F n
  | const : F → Formula F n
  | add : Formula F n → Formula F n → Formula F n
  | mul : Formula F n → Formula F n → Formula F n
  | inv : Formula F n → Formula F n

namespace Formula

variable {F : Type u} {n m : ℕ}

def size : Formula F n → ℕ
  | .var _ | .const _ => 1
  | .add f g | .mul f g => f.size + g.size + 1
  | .inv f => f.size + 1

def height : Formula F n → ℕ
  | .var _ | .const _ => 0
  | .add f g | .mul f g => max f.height g.height
  | .inv f => f.height + 1
```

Source: `TC10/Syntax.lean`.

```
variable {R : Type v} [Ring F] [Ring R]

/-- An inverse is an actual two-sided inverse in the target ring. -/
inductive Eval (c : F →+* R) (X : Fin n → R) : Formula F n → R → Prop
  where
  | var (i) : Eval c X (.var i) (X i)
  | const (a) : Eval c X (.const a) (c a)
  | add {f g a b} : Eval c X f a → Eval c X g b → Eval c X (.add f g) (a + b)
  | mul {f g a b} : Eval c X f a → Eval c X g b → Eval c X (.mul f g) (a * b)
  | inv {f a b} : Eval c X f a → a * b = 1 → b * a = 1 → Eval c X (.inv f) b

def Defined (c : F →+* R) (X : Fin n → R) (f : Formula F n) : Prop :=
  ∃ a, f.Eval c X a
```

S4.2. Matrix semantics and operational rank. The following source uses universes $u \ v$. The first block is in TC10 and enters `ExprMatrix`; the second is again in TC10. Positive rectangular shapes belong to the type. Admissibility is individual: a common witness is subsequently proved. Source: TC10/MatrixSemantics.lean.

```
abbrev Mat (F : Type u) (d : ℕ) := Matrix (Fin d) (Fin d) F
abbrev Tuple (F : Type u) (n d : ℕ) := Fin n → Mat F d

def scalarEmbedding {F : Type u} {E : Type v} [Ring F] [Ring E]
  (ι : F →+* E) (d : ℕ) : F →+* Mat E d :=
  (Matrix.scalar (Fin d)).comp ι

def Evaluates {F : Type u} {E : Type v} [Ring F] [Ring E]
  {n d : ℕ} (ι : F →+* E) (f : Formula F n) (X : Tuple E n d) (A : Mat
    E d) : Prop :=
  f.Eval (scalarEmbedding ι d) X A

/-- Every entry is an actual tree; positive rectangular shapes are part of
the type. -/
structure ExprMatrix (F : Type u) (n : ℕ) where
  rows : ℕ
  cols : ℕ
  rows_pos : 0 < rows
  cols_pos : 0 < cols
  entry : Fin rows → Fin cols → Formula F n

namespace ExprMatrix

variable {F : Type u} {n : ℕ}

def size (M : ExprMatrix F n) : ℕ :=
  ∑ i, ∑ j, (M.entry i j).size

def HeightLE (M : ExprMatrix F n) (h : ℕ) : Prop :=
  ∀ i j, (M.entry i j).height ≤ h

abbrev Value (M : ExprMatrix F n) (E : Type v) (d : ℕ) :=
  Matrix (Fin M.rows × Fin d) (Fin M.cols × Fin d) E

/-- A value is the ordinary expanded block matrix, not a matrix over an
abstract skew field. -/
def Eval {E : Type v} [Ring F] [Ring E] (M : ExprMatrix F n)
  {d : ℕ} (ι : F →+* E) (X : Tuple E n d) (A : M.Value E d) : Prop :=
  ∀ i j, Evaluates ι (M.entry i j) X (fun a b => A (i, a) (j, b))

def Defined {E : Type v} [Ring F] [Ring E] (M : ExprMatrix F n)
  {d : ℕ} (ι : F →+* E) (X : Tuple E n d) : Prop := ∃ A, M.Eval ι X A

/-- Extensions are explicit field embeddings; no common initial witness is
required. -/
def Admissible [Field F] (M : ExprMatrix F n) : Prop :=
```

```

  ∃ (E : Type u) (ι : Field E) (ι : F →+* E) (d : ℕ),
    0 < d ∧ ∃ (X : Tuple E n d) (A : M.Value E d), M.Eval ι X A

```

Source: TC10/MatrixSemantics.lean.

```

def familySize {F : Type u} {n : ℕ} (Ms : List (ExprMatrix F n)) : ℕ :=
  (Ms.map ExprMatrix.size).sum

/-- Supremum-free universal upper rank, including every positive dimension
and extension. -/
def UpperRank {F : Type u} [Field F] {n : ℕ} (M : ExprMatrix F n) (ρ : ℕ)
  : Prop :=
  ∀ (E : Type u) [Field E] (ι : F →+* E) (d : ℕ), 0 < d →
    ∀ (X : Tuple E n d) (A : M.Value E d), M.Eval ι X A → A.rank ≤ d * ρ

/-- A single evaluation with an integral attained rank and a universal
upper bound. -/
def Attains {F : Type u} [Field F] {n D : ℕ} (M : ExprMatrix F n)
  (X : Tuple F n D) (ρ : ℕ) : Prop :=
  ρ ≤ min M.rows M.cols ∧
  (∃ A, M.Eval (RingHom.id F) X A ∧ A.rank = D * ρ) ∧ UpperRank M ρ

```

S4.3. Characteristic tags and canonical prime-field maps. The positive characteristic is a promised prime, represented by a natural number and its primality certificate. This does not charge a primality test to the generator. The complete short source file is reproduced here.

Source: TC10/PrimeField.lean.

```

import TC10.MatrixSemantics
import Mathlib.Data.ZMod.Basic

namespace TC10

universe u

/-- Only promised characteristics are inputs; no primality test is charged
to the generator. -/
inductive Characteristic where
  | zero
  | prime (p : ℕ) (isPrime : p.Prime)

namespace Characteristic

def tag : Characteristic → ℕ
  | .zero => 0
  | .prime p _ => p

def PrimeField : Characteristic → Type
  | .zero => ℚ
  | .prime p _ => ZMod p

```

```

noncomputable instance primeFieldField (p : Characteristic) : Field p.
  PrimeField := by
  cases p with
  | zero => exact inferInstanceAs (Field  $\mathbb{Q}$ )
  | prime p hp =>
    letI : Fact p.Prime := ⟨hp⟩
    exact inferInstanceAs (Field (ZMod p))

instance primeFieldChar (p : Characteristic) : CharP p.PrimeField p.tag :=
  by
  cases p with
  | zero => exact inferInstanceAs (CharP  $\mathbb{Q}$  0)
  | prime p hp =>
    let : Fact p.Prime := ⟨hp⟩
    exact inferInstanceAs (CharP (ZMod p) p)

def embedding (p : Characteristic) (F : Type u) [Field F] [CharP F p.tag] :
  p.PrimeField →+* F := by
  cases p with
  | zero =>
    letI : CharZero F := CharP.charP_to_charZero F
    exact Rat.castHom F
  | prime p hp => exact ZMod.castHom (dvd_refl p) F

def mapTuple (p : Characteristic) {n D :  $\mathbb{N}$ } (F : Type u) [Field F] [CharP
  F p.tag]
  (X : Tuple p.PrimeField n D) : Tuple F n D :=
  fun i a b => p.embedding F (X i a b)

end Characteristic
end TC10

```

S4.4. Literal encodings and deterministic machine execution. Both binary symbols are nonblank. The halting condition specifies the tape from the head rightward and all trailing blanks; it makes no assertion about cells to the left. `PaperContract.tape_word_injective` proves that no binary suffix disappears when represented as a blank-ended tape.

Source: TC10/Encoding.lean.

```

import TC10.PrimeField
import Mathlib.Data.Nat.Digits.Defs
import Mathlib.Data.Nat.Log
import Mathlib.Computability.TuringMachine.PostTuringMachine

namespace TC10

/--! Literal encodings in the paper, including the one-bit representation
  of zero. -/

def binaryDigits (t :  $\mathbb{N}$ ) : List Bool :=

```

```

    if t = 0 then [false] else ((Nat.digits 2 t).reverse.map (fun b => b !=
    0))

def encodeNat (t : ℕ) : List Bool :=
  List.replicate (binaryDigits t).length true ++ [false] ++ binaryDigits t

def encodeRat (q : ℚ) : List Bool :=
  [decide (q.num < 0)] ++ encodeNat q.num.natAbs ++ encodeNat q.den

def encodeScalar (p : Characteristic) → p.PrimeField → List Bool
  | .zero, q => encodeRat q
  | .prime _, z => encodeNat z.val

def encodeTuple (p : Characteristic) {n D : ℕ} (X : Tuple p.PrimeField n D
) : List Bool :=
  (List.finRange n).flatMap fun i =>
    (List.finRange D).flatMap fun a =>
      (List.finRange D).flatMap fun b => encodeScalar p (X i a b)

structure Output (p : Characteristic) (n D : ℕ) where
  dimension_pos : 0 < D
  tuples : List (Tuple p.PrimeField n D)
  nonempty : tuples ≠ []

def encodeOutput {p : Characteristic} {n D : ℕ} (H : Output p n D) : List
Bool :=
  encodeNat D ++ encodeNat H.tuples.length ++ H.tuples.flatMap (
  encodeTuple p)

def encodeInput (p : Characteristic) (n s h : ℕ) : List Bool :=
  encodeNat p.tag ++ List.replicate n true ++ [false] ++
  List.replicate s true ++ [false] ++ List.replicate h true ++ [false]

def logBound (t : ℕ) : ℕ := Nat.clog 2 (t + 2)

/-- Blank, zero, one, work symbol; finite alphabet independent of every
input. -/
abbrev Alphabet := Fin 4
abbrev Machine (m : ℕ) := Turing.TM0.Machine Alphabet (Fin (m + 1))
abbrev Configuration (m : ℕ) := Turing.TM0.Cfg Alphabet (Fin (m + 1))

def tapeBit (b : Bool) : Alphabet := if b then 2 else 1

def run {m : ℕ} (G : Machine m) : ℕ → Configuration m → Option (
  Configuration m)
  | 0, a => some a
  | t + 1, a => (Turing.TM0.step G a).bind (run G t)

/-- Ordinary deterministic transitions, halting, and the complete literal
output. -/
def OutputsWithin {m : ℕ} (G : Machine m) (p : Characteristic) (n s h
bound : ℕ)
  {D : ℕ} (H : Output p n D) : Prop :=

```

```

    ∃ t ≤ bound, ∃ a : Configuration m,
      run G t (Turing.TM0.init ((encodeInput p n s h).map tapeBit)) = some a
    ∧
      Turing.TM0.step G a = none ∧
      a.Tape.right0 = Turing.ListBlank.mk ((encodeOutput H).map tapeBit)
end TC10

```

S4.5. The unchanged main statement. In namespace TC10, with universe u , the dimension function has no variable-count or height argument. The machine and constants precede every characteristic and input; the output precedes every coefficient field and target family. A separate size budget for each family member would be a different proposition.

Source: TC10/MainTheorem.lean.

```

def Structured {p : Characteristic} {n D : ℕ} (H : Output p n D) : Prop :=
  ∀ X ∈ H.tuples,
    (∀ i, IsUnit (X i)) ∧
    ∀ i j, ∃ U : (Mat p.PrimeField D)×,
      X i = (U : Mat p.PrimeField D) * X j * (↑U-1 : Mat p.PrimeField D)

/-- The same point works for the whole family; the family has one total
    budget. -/
def SimultaneousRanks {p : Characteristic} {n D : ℕ} (s h : ℕ)
  (H : Output p n D) : Prop :=
  ∀ (F : Type u) [Field F] [CharP F p.tag] (Ms : List (ExprMatrix F n)),
    Ms ≠ [] → familySize Ms ≤ s →
    (∀ M ∈ Ms, M.HeightLE h) → (∀ M ∈ Ms, M.Admissible) →
    ∃ X ∈ H.tuples, ∀ M ∈ Ms,
      ∃ ρ : ℕ, Attains M (p.mapTuple F X) ρ

def dimensionBound (p : Characteristic) (C s : ℕ) : ℕ :=
  if p.tag = 2 then C * s ^ 2 * logBound s ^ 3
  else C * s ^ 4 * logBound s ^ 4

def lengthBound (p : Characteristic) (C n s h : ℕ) : ℕ :=
  if p.tag = 2 then C * (h + 1) * s ^ 8 * logBound s ^ 6 * logBound n
  else C * (h + 1) * s ^ 10 * logBound s ^ 7 * logBound n

def bitBound (p : Characteristic) (C k n s : ℕ) : ℕ :=
  C * (n + s + logBound p.tag + 1) ^ k

```

Source: TC10/MainTheorem.lean.

```
def MainStatement : Prop :=
  ∃ (m : ℕ) (G : Machine m) (D : Characteristic → ℕ → ℕ) (C k : ℕ),
    0 < C ∧ 0 < k ∧
    ∀ (p : Characteristic) (n s h : ℕ), 0 < n → 0 < s → h ≤ s →
      ∃ H : Output p n (D p s),
        D p s ≤ dimensionBound p C s ∧
        H.tuples.length ≤ lengthBound p C n s h ∧
        (encodeOutput H).length ≤ bitBound p C k n s ∧
        OutputsWithin G p n s h (bitBound p C k n s) H ∧
        Structured H ∧ SimultaneousRanks.{u} s h H
```

S4.6. One named generator, independent of field universes. This complete file constructs its compilation certificate from a proved existence theorem. The standard `Classical.choice` chooses that certificate once. The selected machine and bounds have no universe parameters: they are not chosen again for each coefficient field. The audit checks the universe-parameter lists of all six named witnesses.

Source: TC10/Runtime/FixedGenerator.lean.

```
import TC10.Runtime.MainGenerator

/--! One named finite machine, chosen independently of all field universes.
The unchanged MainStatement is recovered below. The new specification
exposes
the same witnesses at every universe level without changing the generator,
the semantic predicates, the output encoding, or any quantitative bound. -/

namespace TC10.Runtime.FixedGenerator
universe u
noncomputable section

/-- A compilation certificate has no coefficient-field or universe
parameter. -/
structure Compilation where
  states : ℕ
  machine : TC10.Machine states
  C : ℕ
  k : ℕ
  C_pos : 0 < C
  k_pos : 0 < k
  correct : ∀ (p : Characteristic) (n s h : ℕ), h ≤ s →
    let w := (MainGenerator.finish (p.tag,⟨n⟩,⟨s⟩,⟨h⟩)).val
    w.length ≤ bitBound p C k n s ∧
    ∃ t ≤ bitBound p C k n s, ∃ cfg,
      run machine t (Turing.TM0.init ((encodeInput p n s h).map tapeBit))
    = some cfg ∧
      Turing.TM0.step machine cfg = none ∧
      cfg.Tape.right₀ = Turing.ListBlank.mk (w.map tapeBit)

theorem compilation_exists : Nonempty Compilation := by
```

```

obtain ⟨m,G,C,k,hC,hk,hG⟩ := LiteralGenerator.compile MainGenerator.
data_poly
exact ⟨⟨m,G,C,k,hC,hk,hG⟩⟩

noncomputable def program : Compilation := Classical.choice
  compilation_exists
def stateCount : ℕ := program.states
noncomputable def generator : TC10.Machine stateCount := program.machine
def constant : ℕ := program.C + MainGenerator.boundConstant
def exponent : ℕ := program.k
def dimension : Characteristic → ℕ → ℕ := MainGenerator.dimension

theorem constant_pos : 0 < constant := by
  have := program.C_pos
  unfold constant
  omega
theorem exponent_pos : 0 < exponent := program.k_pos

/-- The witnesses in this statement are the same named constants at every
coefficient-field universe. No universe-dependent choice of machine occurs.
-/
theorem specification (p : Characteristic) (n s h : ℕ)
  (hn : 0 < n) (hs : 0 < s) (hh : h ≤ s) :
  ∃ H : Output p n (dimension p s),
    dimension p s ≤ dimensionBound p constant s ∧
    H.tuples.length ≤ lengthBound p constant n s h ∧
    (encodeOutput H).length ≤ bitBound p constant exponent n s ∧
    OutputsWithin generator p n s h (bitBound p constant exponent n s) H
  ∧
  Structured H ∧ SimultaneousRanks.{u} s h H := by
obtain ⟨H,hD,hL,hS,hR,hE⟩ := MainGenerator.output.{u} p n s h hs hh
obtain ⟨hbits,t,ht,cfg,hr,hhalt,hout⟩ := program.correct p n s h hh
change (MainGenerator.finish (p.tag,⟨n⟩,⟨s⟩,⟨h⟩)).val.length ≤ _ at hbits
change cfg.Tape.right₀ = Turing.ListBlank.mk
  ((MainGenerator.finish (p.tag,⟨n⟩,⟨s⟩,⟨h⟩)).val.map tapeBit) at hout
rw [hE] at hbits hout
have hbb : bitBound p program.C exponent n s ≤ bitBound p constant
  exponent n s :=
  Nat.mul_le_mul_right _ (Nat.le_add_right _ _)
refine ⟨H,?,_,hbits.trans hbb,⟨t,ht.trans hbb,cfg,hr,hhalt,hout⟩,hS,hR⟩
· apply hD.trans
  unfold dimensionBound constant
  split_ifs <;> gcongr <;> omega
· apply hL.trans
  unfold lengthBound constant
  split_ifs <;> gcongr <;> omega

/-- The original theorem follows with these fixed, universe-free witnesses.
-/
theorem main : MainStatement.{u} :=
  ⟨stateCount,generator,dimension,constant,exponent,constant_pos,
  exponent_pos,specification⟩

```

```

end
end TC10.Runtime.FixedGenerator

```

S4.7. Expanded contract and independent extension universes.

In namespace `TC10.PaperContract`, the universes `u v w` are independent. Generic specialization proves that an admissibility witness in any universe yields the original predicate, and transfers the upper-rank bound to every extension universe. The two bridge signatures and the expanded conclusion are given below.

Source: `TC10/PaperContract.lean`.

```

theorem admissible_of_witness {F : Type u} [Field F] {E : Type v} [Field E]
]
  {n d : ℕ} (M : ExprMatrix F n) (ι : F →++ E) (hd : 0 < d)
  (X : Tuple E n d) (A : M.Value E d) (hA : M.Eval ι X A) : M.Admissible

```

Source: `TC10/PaperContract.lean`.

```

theorem upperRank_any_universe {F : Type u} [Field F] {E : Type v} [Field E]
]
  {n d ρ : ℕ} {M : ExprMatrix F n} (h : UpperRank M ρ)
  (ι : F →++ E) (hd : 0 < d) (X : Tuple E n d) (A : M.Value E d)
  (hA : M.Eval ι X A) : A.rank ≤ d * ρ

```

Source: `TC10/PaperContract.lean`.

```

theorem main :
  ∃ (m : ℕ) (G : Machine m) (D : Characteristic → ℕ → ℕ) (C k : ℕ),
    0 < C ∧ 0 < k ∧
    ∀ (p : Characteristic) (n s h : ℕ), 0 < n → 0 < s → h ≤ s →
      ∃ H : Output p n (D p s),
        D p s ≤ (if p.tag = 2 then C * s ^ 2 * (Nat.clog 2 (s + 2)) ^ 3
          else C * s ^ 4 * (Nat.clog 2 (s + 2)) ^ 4) ∧
        H.tuples.length ≤ (if p.tag = 2 then
          C * (h + 1) * s ^ 8 * (Nat.clog 2 (s + 2)) ^ 6 * Nat.clog 2 (n
            + 2)
          else C * (h + 1) * s ^ 10 * (Nat.clog 2 (s + 2)) ^ 7 * Nat.
            clog 2 (n + 2)) ∧
        (encodeOutput H).length ≤ C * (n + s + Nat.clog 2 (p.tag + 2) +
          1) ^ k ∧
        (∃ t ≤ C * (n + s + Nat.clog 2 (p.tag + 2) + 1) ^ k,
          ∃ a : Configuration m,
            run G t (Turing.TMO.init ((encodeInput p n s h).map tapeBit))
            = some a ∧
            Turing.TMO.step G a = none ∧
            a.Tape.right0 = Turing.ListBlank.mk ((encodeOutput H).map
              tapeBit)) ∧
        (∀ X ∈ H.tuples, (∀ i, IsUnit (X i)) ∧
          ∀ i j, ∃ U : (Mat p.PrimeField (D p s))×,
            X i = (U : Mat p.PrimeField (D p s)) * X j *
              (↑U-1 : Mat p.PrimeField (D p s))) ∧

```

```

(∀ (F : Type u) [Field F] [CharP F p.tag] (Ms : List (ExprMatrix
F n)),
  Ms ≠ [] →
  (Ms.map (fun M => ∑ i, ∑ j, (M.entry i j).size)).sum ≤ s →
  (∀ M ∈ Ms, ∀ i j, (M.entry i j).height ≤ h) →
  (∀ M ∈ Ms, ∃ (E : Type v) (ι : Field E) (ι : F →+* E) (d : ℕ)
  ,
    0 < d ∧ ∃ (X : Tuple E n d) (A : M.Value E d), M.Eval ι X A)
→
  ∃ j : Fin H.tuples.length, ∀ i : Fin Ms.length, ∃ ρ : ℕ,
  ρ ≤ min (Ms.get i).rows (Ms.get i).cols ∧
  (∃ A, (Ms.get i).Eval (RingHom.id F)
    (p.mapTuple F (H.tuples.get j)) A ∧ A.rank = D p s * ρ) ∧
  (∀ (E : Type w) [Field E] (ι : F →+* E) (d : ℕ), 0 < d →
    ∀ (X : Tuple E n d) (A : (Ms.get i).Value E d),
    (Ms.get i).Eval ι X A → A.rank ≤ d * ρ))

```

S5. FURTHER CONSEQUENCES AND EXPLICIT EXAMPLES

S5.1. The exact lower-bound contract. Proposition 2.4 uses the theorem’s strongest quantifiers: the same list must work after any coefficient field and family have been chosen. The proof of $L \geq \lfloor s/3 \rfloor + 1$ selects the algebraic closure of the prime field and an eigenvalue for each listed first coordinate. Each target $x_1 - c_j$ is an addition of two leaves and has operational rank one. Their common total budget is $3L$. This proves a limitation on full-rank attainment, even for division-free targets; it is not an obstruction to a single point that merely detects nonzero values.

For $p > 0$ and $s \geq 4$, the four-leaf matrix $\begin{pmatrix} x_i & x_k \\ 1 & 1 \end{pmatrix}$ forces distinct coordinate signatures. Counting all $D \times D$ prime-field matrices gives $n \leq p^{D^2 L}$. At fixed p, s , the dimension function $D(p, s)$ is fixed, so logarithmic growth in n is necessary. Neither proof uses the structured-output requirement. The bounds leave a substantial polynomial gap in s and establish no optimality of the construction.

The following declarations are in `TC10.UniversalLowerBounds`, with implicit natural numbers `n D`. Their explicit `SimultaneousRanks` hypothesis is discharged by the main theorem; it is not a new assumption on the generator.

Source: `TC10/UniversalLowerBounds.lean`.

```

theorem list_length {p : Characteristic} {s h : ℕ} (hn : 0 < n) (H :
  Output p n D)
  (hRanks : SimultaneousRanks.{0} s h H) : s / 3 + 1 ≤ H.tuples.length

```

Source: TC10/UniversalLowerBounds.lean.

```
theorem naming_bound (p : ℕ) (hp : p.Prime) {s h : ℕ}
  (H : Output (.prime p hp) n D) (hs : 4 ≤ s)
  (hRanks : SimultaneousRanks.{0} s h H) : n ≤ p ^ (D * D * H.tuples.
length)
```

S5.2. Universal tests for algebraic branching programs. Corollary.

Let P be an r -vertex noncommutative algebraic branching program over F with affine edge labels in n variables, and put $S = (r + 1)^2(4n + 1)$. The list $\mathcal{H}_{p,n,S,0}$ depends only on p, n, r . The program has a nonzero finite-matrix evaluation over an extension of F if and only if its value is invertible at some listed tuple transported to F . It therefore gives universal nonadaptive tests in every characteristic, with the dimension, list length and generator runtime obtained by substituting S for s in Theorem 2.1.

Proof. Topologically order the vertices and let $A(x)$ be the strictly upper triangular affine adjacency matrix. Matrix multiplication retains the order of edge labels along paths. At every matrix substitution, $A^r = 0$ and

$$Q = I + A + \cdots + A^{r-1} = (I - A)^{-1}, \quad f = e_a^\top Q e_b,$$

where a, b are the source and sink. Thus f is exactly the finite path-sum value. The affine border

$$B(x) = \begin{pmatrix} I - A(x) & e_b \\ e_a^\top & 0 \end{pmatrix}$$

is invertible exactly when f is invertible, by its Schur complement $-f$. There is no restriction on the substitution needed to invert $I - A$. If a nonzero value exists, expand the finite path sum into a division-free formula and apply Lemma 3.2 to obtain a unit witness. The border then has a unit witness. Its $(r + 1)^2$ affine entries each have a literal tree of size $4n + 1$ and height zero. Theorem 2.1 at budget S supplies an invertible listed border, hence an invertible program value. The reverse implication is immediate. The potentially large path-sum formula is used only to prove existence of a witness; its size is not used in S or by the generator.

The triangular representation is classical; see Hrubeš–Wigderson, *Non-commutative arithmetic circuits with division*, Section 6.1, Proposition 6.5, pp. 374–375, [doi:10.4086/toc.2015.v011a014](https://doi.org/10.4086/toc.2015.v011a014). Our corollary applies the present universal list to that representation. The budget S includes the affine labels’ variable count, so the substituted bounds are polynomial in n, r ; a variable-independent ABP dimension is not asserted by this conversion. The statement uses operational nonzero, as does the paper’s scalar corollary. An identification with any abstract free-algebra encoding is not required.

In TC10.BranchingPrograms, vertices are `Fin r`; the coefficient field has universe `u`. A missing edge has zero affine label. `value` is the selected entry

of the finite sum of adjacency powers. The following source specifies the graph and operational nonzero exactly.

Source: TC10/BranchingPrograms.lean.

```
structure Program (F : Type u) (n r : ℕ) [Field F] where
  labels : Matrix (Fin r) (Fin r) (Affine.Form F n)
  acyclic : ∀ i j, j ≤ i → labels i j = 0
  source : Fin r
  sink : Fin r
```

Source: TC10/BranchingPrograms.lean.

```
def Nonzero (P : Program F n r) : Prop :=
  ∃ (E : Type u) (c : Field E) (c : F →+* E) (d : ℕ), 0 < d ∧
    ∃ X : Tuple E n d, value P (scalarEmbedding c d) X ≠ 0
```

Source: TC10/BranchingPrograms.lean.

```
theorem hitting {p : Characteristic} [CharP F p.tag] {D s : ℕ}
  (H : Output p n D) (hRanks : SimultaneousRanks.{u} s 0 H)
  (P : Program F n r) (hs : (r+1)^2*(4*n+1) ≤ s) :
  Nonzero P ↔ ∃ X ∈ H.tuples, IsUnit (value P (scalarEmbedding (RingHom.
    id F) D)
    (p.mapTuple F X))
```

The complete source also checks `acyclic_nilpotent`, `paths_inverse`, `border_value_unit_iff`, the exact `affineMatrix_size`, and the expanded formula's evaluation. Both lower-bound lemmas and `hitting` are named entries in the transitive axiom audit. Their proofs introduce no additional axioms, and do not change the main statement or literal machine encoding.

S5.3. Literature pinpoints and provenance. The comparison in the introduction uses these exact versions:

- Chatterjee, Ghosh, Gurjar, Raj and Thierauf, *Bipartite Matching is in NC*, ECCC TR26-100, revision 2, 15 July 2026, Theorem 5.3, printed p. 37: “ A has full nc-rank $\iff N(A)$ has full rank.” Here A is their block-symbolic matrix; $N(A)$ is their augmented rank-certificate matrix, constructed with their stated parameters. [Pinned revision](#).
- Brakensiek, Chen, Dhar and Zhang, *From Random to Explicit via Subspace Designs With Applications to Local Properties and Matroids*, ECCC TR25-146 (2025), Remark 6.4, printed p. 31, identifies further boundary roots and says “we defer such arguments to future work.” The full boundary multiplicity bound used here is proved in Lemma 5.2, rather than assumed from that remark. [Cited manuscript](#). The work also appears as [arXiv:2510.13777](#) and was accepted at [STOC 2026](#); the pinpoint above refers specifically to the ECCC version.

The main formal theorem assumes no unproved result from the September OpenAI preprints. Adapted arguments and source files are credited and rechecked. This statement does not extend the formalization to the classical free-field identification or the general-tree strengthening identified in Section S1. The result includes a deterministic machine and verified polynomial runtime; it makes no claim of practical efficiency.

S5.4. Exact literal cost of the logarithmic dictionary. For every signed word w , including the empty word, the existing literal tree $\text{code}(w) = w v w^{-1}$ has exactly $5|w| + 5$ nodes. The inverse word reverses the letters and flips their signs; it is not a single inverse gate around the whole word. Both word products retain their constant 1 leaves. The exact recursive syntax is displayed below. A letter and its opposite sign contribute three nodes together, and the two word-product nodes contribute two more. The terminal leaves, central variable and outer products contribute five. This is an exact count for this representation, not a minimality claim among all equivalent formulas. The following proof needs only an element 1 in the coefficient type; its conclusion is purely syntactic. Source: TC10/ConjugateCoding.lean.

```
abbrev Letter := Fin 2 × Bool
abbrev Word := List Letter

def letter {F : Type*} (l : Letter) : Formula F 2 :=
  if l.2 then .var l.1 else .inv (.var l.1)

def word {F : Type*} [One F] : Word → Formula F 2
| [] => .const 1
| l :: w => .mul (letter l) (word w)

def inverseWord {F : Type*} [One F] : Word → Formula F 2
| [] => .const 1
| l :: w => .mul (inverseWord w) (letter (l.1,!l.2))

def code {F : Type*} [One F] (w : Word) : Formula F 2 :=
  .mul (word w) (.mul (.var 1) (inverseWord w))
```

Source: TC10/ExactCodeSize.lean.

```

theorem word_pair_size_exact (w : Word) :
  (word (F := F) w).size + (inverseWord (F := F) w).size = 5 * w.length +
  2 := by
  induction w with
  | nil => simp [word, inverseWord, Formula.size]
  | cons l w ih =>
    rcases l with ⟨i, s⟩
    cases s <|> simp_all [word, inverseWord, letter, Formula.size] <|>
    omega

/-- The exact size of the literal tree, for arbitrary signed words,
    including
the empty word. No cancellation, sharing or algebraic simplification is
used. -/
theorem code_size_exact (w : Word) :
  (code (F := F) w).size = 5 * w.length + 5 := by
  have hw := word_pair_size_exact (F := F) w
  simp only [code, Formula.size]
  omega

```

At dictionary size n , every word has length $\Lambda(n) + 1$, so its literal leaf cost is $5(\Lambda(n) + 1) + 5$. Entrywise substitution therefore satisfies the sharper local estimate below, where `code` is the existing `TC10.ShortWords.code`. The executed generator retains its original larger budgets. Its parameters, dimension, list-length and runtime claims are unchanged.
Source: TC10/ExactCodeSize.lean.

```

theorem size_le_exact_cost {F : Type*} [Field F] {n : ℕ} (M : ExprMatrix F
n) :
  (code M).size ≤ (5 * (logBound n + 1) + 5) * M.size

```

S5.5. A reproducible characteristic-two integration example. The matrices U, V, P are those in the main paper's example. To show the normalization and literal decoding, take the homogeneous raw triple

$$(Z_0, Z_1, Z_2) = (P, PU, PV), \quad (Z_0^{-1}Z_1, Z_0^{-1}Z_2) = (U, V).$$

For $n = 2$, the actual dictionary words are $g_0 = u^3$ and $g_1 = vu^2$. Since $U^2 = V^2 = I_2$, decoding yields

$$(g_0(U, V)Vg_0(U, V)^{-1}, g_1(U, V)Vg_1(U, V)^{-1}) = (P, V).$$

Both coordinates are units and conjugate. Direct multiplication gives $PV + VP = I_2$, so the original height-one tree $f(x, y) = (xy + yx)^{-1}$ evaluates to I_2 with rank two and normalized rank one. Its tree has eight nodes; each dictionary leaf has twenty. At (I_2, I_2) the inverse operand is zero, and even $0 \cdot f$ remains undefined. These identities use the actual Lean dictionary and strict evaluation relation in `TC10/CharacteristicTwoExample.lean`.

Source: TC10/CharacteristicTwoExample.lean.

```
theorem dictionary : ShortWords.word (0 : Fin 2) = [(0, true), (0, true),
  (0, true)] ∧
  ShortWords.word (1 : Fin 2) = [(1, true), (0, true), (0, true)] := by
  decide

theorem decoded_eq : decoded = ![P, V] := by decide
```

Source: TC10/CharacteristicTwoExample.lean.

```
theorem target_eval : target.Eval (scalarEmbedding (RingHom.id K) 2)
  decoded 1 := by
  rw [decoded_eq]
  have hg : operand.Eval (scalarEmbedding (RingHom.id K) 2) ![P, V] (P * V
    + V * P) :=
    .add (.mul (.var 0) (.var 1)) (.mul (.var 1) (.var 0))
  rw [commutator] at hg
  exact .inv hg (one_mul 1) (one_mul 1)

theorem cancelled_target_undefined :
  ¬ ((Formula.const (0 : K)).mul target).Defined
  (scalarEmbedding (RingHom.id K) 2) (fun _ => 1) := by
  rw [Formula.defined_mul_iff]
  rintro ⟨_, a, ha⟩
  have hg : operand.Eval (scalarEmbedding (RingHom.id K) 2) (fun _ => 1) 0
    := by
    have hz : (1 : R) * 1 + 1 * 1 = 0 := by decide
    exact hz ▸ Formula.Eval.add (Formula.Eval.mul (.var 0) (.var 1))
      (Formula.Eval.mul (.var 1) (.var 0))
  cases ha with
  | inv he hr _ =>
    have hv := he.unique hg
    rw [hv, zero_mul] at hr
    exact zero_ne_one hr
```

The added correspondence checks in TC10/CharacteristicTwoAudit.lean evaluate the actual substituted tree at (U, V) and prove exact domain pullback there. That tree has 84 nodes and height two; the original tree still has size eight and height one. The same file checks the decoded leaf size, structured output and rank of the value. It also proves that f has no scalar domain over *any* characteristic-two field, with arbitrary coefficient embedding, rather than only testing $\mathbb{F}_2$.

Source: TC10/CharacteristicTwoAudit.lean.

```
theorem coded_target_eval :
  (target.subst (fun i => ConjugateCoding.code (ShortWords.word i))).
  Eval
    (scalarEmbedding (RingHom.id K) 2) (fun i => (baseUnits i : R)) 1
```

Source: TC10/CharacteristicTwoAudit.lean.

```
theorem scalar_target_undefined {E : Type*} [Field E] [CharP E 2]
  (c : K →** E) (X : Fin 2 → E) : ¬ target.Defined c X
```

For a single decoded pair (P, V) , the literal output contains the codes of $D = 2$, $L = 1$, then the row-major entries $(1, 0, 1, 1, 1, 1, 0, 1)$. It is the 32-bit word

11010101101100101101101101100101.

The characteristic and number of variables are already known from the input, exactly as in the main theorem. The source checks the production encoder directly; the preceding natural numbers in the entry list are encoded individually, not packed as raw field bits.

Source: TC10/CharacteristicTwoExample.lean.

```
theorem sample_encoding : encodeOutput sampleOutput =
  encodeNat 2 ++ encodeNat 1 ++
  [1, 0, 1, 1, 1, 1, 0, 1].flatMap encodeNat := by decide

theorem sample_encoding_length : (encodeOutput sampleOutput).length = 32
:= by decide
```

Source: TC10/CharacteristicTwoAudit.lean.

```
theorem sample_encoding_bits :
  String.ofList ((encodeOutput sampleOutput).map (fun b => if b then '1'
    else '0')) =
  "11010101101100101101101101100101" := by decide
```

Finite enumeration. The script `scripts/characteristic_two_example.py` enumerates all 256 pairs (Z_1, Z_2) of 2×2 binary matrices, in row-major lexicographic order, with $Z_0 = P$ fixed. It normalizes, retains the 36 unit pairs, and decodes each using u^3, vu^2 . It preserves list order and repeated outputs. No target or target certificate is an input to this generation. Only afterwards does it evaluate f and $0 \cdot f$ on the resulting list. Exactly 12 entries lie in the strict domain of f , all with value I_2 ; the other 24 also make $0 \cdot f$ undefined. The complete dense list has 882 encoded bits. The script checks its round trip, two nonblank tape-bit symbols, trailing blanks, and rejection of truncated, noncanonical or trailing-data encodings. Its 289 finite checks and the complete literal output are recorded by the verifier.

This deliberately bounded example starts from supplied raw matrices. It does not run the universal quotient and determinant-curve core, finite-extension construction, naming table or compiled Turing machine. In particular, $D = 2$ and this 36-entry list do not claim the universal guarantee of Theorem 2.1. The small Lean witness checks the displayed identities and its encoding; the 36-entry enumeration is an independent Python diagnostic. The proof's target-dependent certificates establish correctness and are never inputs to the actual universal generator.

S6. REPRODUCTION, PROVENANCE AND MEASURED SCOPE

The clean package contains both standalone papers, all 366 production Lean files, pinned dependency metadata, upstream references and licenses, and verification tools. From `formal/tc10/`, run

```
lake exe cache get
python3 -B scripts/verify.py
```

The verifier checks the source, contract, encoding, excerpt and dependency pins; builds every module; and checks transitive axioms, declaration names and six fixed witnesses. It runs 8,047 arithmetic regressions, 289 characteristic-two checks, quotient/curve diagnostics at $q = 2, 3$, and parser/excerpt regressions. Only `propext`, `Classical.choice`, and `Quot.sound` are allowed. The finite diagnostics supplement the kernel proof; they include a scalar-singular 3×3 skew-symmetric pencil with an invertible size-two evaluation.

S6.1. Build measurements and portability. The historical fresh local-source build, using pinned dependency caches, completed 3,631 Lake jobs in 429.19 seconds on macOS arm64; its peak memory was unavailable. See `validation/fresh-source-build.json`. Current verification times and maximum child-process RSS appear in `validation/verification.json`. This RSS is not aggregate concurrent memory, and cached timing is not fresh-build timing. A separately recorded isolated rebuild covers the byte-identical Lean sources without prior local proof artifacts; see `validation/publication-source-build.json`.

The package’s `README.txt` lists the installation endpoints. With the pinned toolchain and dependencies installed, proof checks can run offline. Linux verification was excluded and not run; the optional manual workflow remains available.

S6.2. Trust, attribution and publication state. The source hashes in `validation/contract.json` and the excerpt map in `validation/supplement-excerpts.json`, together with the package’s `SHA256SUMS`, identify the exact distributed versions. These checksums establish file consistency, not a publisher’s digital signature. An executable proof check still requires the pinned Lean toolchain. The original OpenAI modules and Apache 2.0 notices are retained in `reference/openai/`; `THIRD_PARTY_NOTICES.txt` and `TC10/Runtime/PROVENANCE.txt` distinguish adapted files from originals.

The preparation of this manuscript and its computational companion benefited from assistance by AI systems developed by OpenAI and Anthropic. Responsibility for the mathematical claims, implementation, and presentation rests with the author.

The reported audit comprises kernel checks and an internal correspondence review. Independent referee review, public archival deposit and venue-specific disclosure remain publication steps.