

On a non-basis of the coinvariant algebra

GPT-5.6 Sol and GPT-6 Astra,
prompted by Darij Grinberg

19 August 2026; rewritten 28 September 2026

Abstract

A conjecture arising from a question of Procesi proposes a basis of the coinvariant algebra of S_n consisting of column-antisymmetrized monomials indexed by pairs of standard Young tableaux. We show that the proposed family need not even span an S_n -subrepresentation: for $n = 8$, a generator of degree 15 is sent outside the span by the adjacent transposition $(4, 5)$. The proof is an exact finite computation with an explicit separating functional, requiring neither a rank computation nor Gröbner reduction. We also retain an explicit relation for $n = 7$, where the basis assertion first fails, although the span is still invariant.

1 The proposed family and the main result

Let $n \geq 1$, let $\mathbf{k}$ be a field of characteristic 0, and define

$$\begin{aligned} &\text{the polynomial ring } \mathcal{P}_n = \mathbf{k}[x_1, \dots, x_n], \\ &\text{the ideal } \mathcal{J}_n = (e_1, \dots, e_n), \quad \text{and} \\ &\text{the quotient algebra } R_n = \mathcal{P}_n / \mathcal{J}_n, \end{aligned}$$

where e_i is the i -th elementary symmetric polynomial. The symmetric group S_n acts by $\sigma(x_i) = x_{\sigma(i)}$ on $\mathcal{P}_n$, and this action descends to the coinvariant algebra R_n . A bar denotes the image of a polynomial in R_n .

We use English notation for Young tableaux: rows get shorter from top to bottom, and standard tableaux increase along rows and down columns. For a partition $\lambda \vdash n$, let $\text{SYT}(\lambda)$ be the set of standard tableaux of shape λ , and let $Y(\lambda)$ be its Young diagram. A *descent* of $S \in \text{SYT}(\lambda)$ is an $i \in \{1, \dots, n-1\}$ such that $i+1$ lies in a lower row than i . The *cocharge tableau* $\text{cc}(S)$ replaces each entry i of S by the number of descents smaller than i . Write $d(S)$ for the sum of the entries of $\text{cc}(S)$.

Example 1.1. For instance,

$$\text{if } S = \begin{array}{|c|c|c|c|} \hline 1 & 2 & 5 & 7 \\ \hline 3 & 4 & 8 & \\ \hline 6 & & & \\ \hline \end{array}, \text{ then } \text{cc}(S) = \begin{array}{|c|c|c|c|} \hline 0 & 0 & 1 & 2 \\ \hline 1 & 1 & 3 & \\ \hline 2 & & & \\ \hline \end{array}$$

(and the descents of S are 2, 5, 7).

For $S, T \in \text{SYT}(\lambda)$, set

$$\mathbf{x}_T^S = \prod_{c \in Y(\lambda)} x_{T(c)}^{\text{cc}(S)(c)}.$$

Let $\text{Col } T$ be the subgroup of S_n consisting of all permutations that preserve each column of T as a set. Define

$$\nabla_{\text{Col } T}^- = \sum_{\sigma \in \text{Col } T} \text{sgn}(\sigma) \sigma \in \mathbf{k}[S_n], \quad F_T^S = \nabla_{\text{Col } T}^- \mathbf{x}_T^S.$$

These polynomials F_T^S are homogeneous of degree $d(S)$. For example, when S is the tableau filled consecutively along its rows, $\text{cc}(S)$ is constant with value $i - 1$ in row i , and the F_T^S are the usual Specht–Vandermonde polynomials.

Allen records the conjecture that

$$\left(\overline{F_T^S} \right)_{\lambda \vdash n, S, T \in \text{SYT}(\lambda)} \quad (1)$$

is a basis of R_n [Allen93, (I.5)]. He attributes the basis problem and a proposed construction to Procesi, while describing the basis assertion as an informally circulated conjecture [Allen92]. Two closely related constructions do give bases: row symmetrization alone [Allen93, Theorem I.1], and row symmetrization followed by column antisymmetrization [ArTeYa97, Theorem 1(1)]. In our notation these are, respectively,

$$\left(\nabla_{\text{Row } T} \mathbf{x}_T^S \right)_{\lambda, S, T} \quad \text{and} \quad \left(\nabla_{\text{Col } T}^- \nabla_{\text{Row } T} \mathbf{x}_T^S \right)_{\lambda, S, T},$$

where $\nabla_{\text{Row } T}$ is the sum of the permutations preserving each row of T as a set, and the indexing sets are as in (1).

The failure of (1) is stronger than linear dependence. For any $d \geq 0$, set

$$W_{n,d} = \text{span}_{\mathbf{k}} \left\{ \overline{F_T^S} : \lambda \vdash n, S, T \in \text{SYT}(\lambda), d(S) = d \right\} \subseteq (R_n)_d.$$

The span of the whole family (1) is the graded space $W_n = \bigoplus_{d \geq 0} W_{n,d}$.

Theorem 1.2. *The space $W_{8,15}$ is not S_8 -stable. More explicitly, take*

$$S = \begin{array}{|c|c|c|c|} \hline 1 & 3 & 4 & 7 \\ \hline 2 & 5 & 8 & \\ \hline 6 & & & \\ \hline \end{array}, \quad T = \begin{array}{|c|c|c|c|} \hline 1 & 2 & 4 & 5 \\ \hline 3 & 6 & 7 & \\ \hline 8 & & & \\ \hline \end{array}, \quad \tau = (4, 5).$$

Then $d(S) = 15$ and

$$\overline{F_T^S} \in W_{8,15}, \quad \tau \overline{F_T^S} \notin W_{8,15}.$$

Consequently, the span W_8 of the entire proposed family is not an S_8 -subrepresentation of R_8 .

If the definition of F_T^S is extended to arbitrary bijective fillings T , then $\sigma F_T^S = F_{\sigma T}^S$. But σT need not be standard. The theorem shows that restricting to standard T can destroy invariance, even after passing to the coinvariant algebra.

2 An explicit certificate of non-invariance

We prove Theorem 1.2 by constructing a linear functional on $(R_8)_{15}$ that vanishes on $W_{8,15}$ but not on $\tau \overline{F}_T^S$. All the identities needed to check this certificate are identities over the integers.

The alternating pairing

For general n , put

$$N = \binom{n}{2}, \quad \Delta = \prod_{1 \leq i < j \leq n} (x_j - x_i), \quad \text{Alt}(h) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \sigma(h)$$

for any polynomial $f \in \mathcal{P}_n$. Every alternating polynomial is divisible by Δ . In particular, an alternating polynomial of degree less than N is zero.

Lemma 2.1. *Let $0 \leq d \leq N$, and let $g \in \mathcal{P}_n$ be homogeneous of degree $N - d$. For every homogeneous f of degree d , there is a unique scalar $L_g(f) \in \mathbf{k}$ such that*

$$\text{Alt}(fg) = L_g(f)\Delta. \quad (2)$$

The linear functional L_g vanishes on $(\mathcal{J}_n)_d$, and therefore defines a linear functional on $(R_n)_d$.

Proof. The existence and uniqueness follow from divisibility by Δ and $\deg(fg) = N$. Now, recall that the map $\text{Alt} : \mathcal{P}_n \rightarrow \mathcal{P}_n$ is $\mathcal{P}_n^{S_n}$ -linear, where $\mathcal{P}_n^{S_n} \subseteq \mathcal{P}_n$ is the subalgebra consisting of the symmetric polynomials. In particular, $\text{Alt}(e_i p) = e_i \text{Alt}(p)$ for any $i \geq 1$ and any $p \in \mathcal{P}_n$. Thus, if $f = e_i h$ is homogeneous of degree d , then

$$\text{Alt}(fg) = e_i \text{Alt}(hg) = 0,$$

since $\deg(hg) = N - i < N$ (and Alt annihilates anything of degree $< N$). But products of this form $e_i h$ span $(\mathcal{J}_n)_d$. Thus, L_g vanishes on $(\mathcal{J}_n)_d$, as desired. $\square$

Here is a coefficient formula for this functional. For $v \in \mathbb{Z}_{\geq 0}^n$, define $\varepsilon(v)$ to be the sign of the permutation v of $(0, 1, \dots, n-1)$ if v is such a permutation, and to be 0 otherwise. If $g = \sum_b g_b x^b \in (\mathcal{P}_n)_{N-d}$, then

$$L_g(x^a) = \sum_b g_b \varepsilon(a+b) \quad \text{for all } a \in \mathbb{Z}_{\geq 0}^n \text{ with } \sum_i a_i = d. \quad (3)$$

Indeed, this follows from the equality

$$\text{Alt}(x^c) = \varepsilon(c)\Delta \quad \text{for all } c \in \mathbb{Z}_{\geq 0}^n \text{ satisfying } \sum_i c_i \leq N,$$

which is very easy to prove: A degree- N monomial is sent to 0 by Alt unless its exponents are distinct. The only n distinct nonnegative integers summing to N are $0, 1, \dots, n-1$, and any monomial x^c whose exponents are $0, 1, \dots, n-1$ is sent by Alt to $\varepsilon(c)\Delta$.

The separating polynomial

Now take $n = 8$ and $d = 15$, so that $N - d = 13$. For $I = \{i_1 < \dots < i_m\}$ and nonnegative integers $a_1, \dots, a_m$, write

$$D_I(a_1, \dots, a_m) = \det (x_{i_r}^{a_s})_{1 \leq r, s \leq m}.$$

Set

$$A = \{1, 5, 6\}, \quad B = \{2, 7, 8\}, \quad C = \{3, 4\}.$$

Our certificate is the degree-13 polynomial

$$g = D_A(0, 1, 2) q \in (\mathcal{P}_n)_{N-d}, \tag{4}$$

where

$$\begin{aligned} q = & D_B(0, 1, 2)(103D_C(0, 7) + 48D_C(1, 6)) \\ & - 32D_B(0, 2, 3)D_C(2, 3) + 105D_B(1, 2, 3)D_C(1, 3) \\ & + (-32D_B(0, 1, 6) + 77D_B(1, 2, 4))D_C(1, 2) \\ & - 60D_B(0, 1, 7)D_C(0, 2) + 77D_B(0, 2, 7)D_C(0, 1). \end{aligned}$$

Thus g is specified by eight products of determinants of sizes at most 3×3 .

Verification and conclusion

For the tableaux in Theorem 1.2,

$$\text{cc}(S) = \begin{array}{|c|c|c|c|} \hline 0 & 1 & 1 & 3 \\ \hline 1 & 2 & 4 & \\ \hline 3 & & & \\ \hline \end{array}, \quad \mathbf{x}_T^S = x_2 x_3 x_4 x_5^3 x_6^2 x_7^4 x_8^3.$$

In particular, $d(S) = 15$. With g as in (4), we now claim that

$$L_g(F_{T'}^{S'}) = 0 \quad \text{for all } \lambda \vdash 8, \quad S', T' \in \text{SYT}(\lambda), \quad d(S') = 15, \tag{5}$$

$$L_g((4, 5)F_T^S) = -1440. \tag{6}$$

There are 3736 indexed pairs in (5). The standalone script supplied with this note [Verify26] verifies every one of these identities using (3): it expands the small determinants in g , generates standard tableaux recursively by removing the largest entry at a corner, and sums the signed monomials in each $F_{T'}^{S'}$. All operations are exact integer operations. In particular, this verification uses no rank computation, modular inference, or Gröbner basis. Formula (3) also makes the check reproducible without the supplied implementation.

By Lemma 2.1, L_g descends to $(R_8)_{15}$. Equation (5) says that it annihilates $W_{8,15}$, whereas (6) shows that it does not annihilate $(4, 5)\overline{F_T^S}$. Since $-1440 \neq 0$ in every characteristic-0 field, this proves Theorem 1.2 over every such field.

Remark 2.2 (Small cases). The supplementary checks [Verify26] show that (1) is a basis for $n \leq 6$. For $n = 7$, its only failure is in degree 10, where $W_{7,10}$ has codimension 21 but is nevertheless S_7 -stable. Also, $W_{8,10}$ is S_8 -stable. Thus invariance can persist even when the basis assertion fails, but it is not a general property of the construction. None of these supplementary checks is needed for Theorem 1.2.

A An explicit relation for seven variables

For $n = 7$, the following relation is already sufficient to show that the family (1) is not a basis of R_n . Let

$$\mathcal{A} = \{U \in \text{SYT}(3, 2, 1) : \text{the entry 3 occupies cell } (1, 2) \text{ or } (2, 1) \text{ in } U\}.$$

This set has 12 elements. For $\mu \in \{(3, 3, 1), (3, 2, 2), (3, 2, 1, 1)\}$, let U^μ be the tableau obtained from U by adjoining the unique cell of $\mu/(3, 2, 1)$ and filling it with 7. Define

$$E_\mu(S) = \sum_{U \in \mathcal{A}} \overline{F_{U^\mu}^S} \quad \text{for all } S \in \text{SYT}(\mu).$$

Writing tableaux by their rows, separated by vertical bars, we have in R_7 :

$$\begin{aligned} 0 = & 5E_{(3,2,1,1)}[134 \mid 26 \mid 5 \mid 7] + 3E_{(3,2,1,1)}[127 \mid 34 \mid 5 \mid 6] \\ & + 9E_{(3,2,1,1)}[124 \mid 37 \mid 5 \mid 6] - 2E_{(3,2,1,1)}[126 \mid 35 \mid 4 \mid 7] \\ & - 4E_{(3,3,1)}[134 \mid 256 \mid 7] + 16E_{(3,3,1)}[126 \mid 357 \mid 4] \\ & + 24E_{(3,3,1)}[124 \mid 357 \mid 6] - 12E_{(3,2,2)}[134 \mid 26 \mid 57] \\ & - 24E_{(3,2,2)}[124 \mid 35 \mid 67] - 8E_{(3,2,2)}[126 \mid 35 \mid 47]. \end{aligned}$$

All 120 summands have degree 10 and distinct indices (S, T) , so the relation is nontrivial. The raw sum reduces to zero over $\mathbb{Z}$ using the Gröbner basis

$$h_i(x_i, x_{i+1}, \dots, x_7) \quad \text{for all } 1 \leq i \leq 7$$

of $\mathcal{J}_7$ for lexicographic order $x_1 > \dots > x_7$, whose leading monomials are x_i^i . The supplementary script [Verify26] checks this reduction. The first-failure and stability assertions in Remark 2.2 require separate checks; they do not follow from this one relation.

Manifest. Written by GPT-5.6 Sol and subsequently revised and extended by GPT-6 Astra, at Darij Grinberg's prompting. DG has verified all claims and proofs using his own SageMath code (see Subsection B.8) written independently of GPT's .py files.

References

- [Allen92] Edward E. Allen, *A conjecture of Procesi and the straightening algorithm of Rota*, Proc. Natl. Acad. Sci. USA **89** (1992), no. 9, pp. 3980–3984.
- [Allen93] Edward E. Allen, *A Conjecture of Procesi and a New Basis for the Decomposition of the Graded Left Regular Representation of S_n* , Advances in Mathematics **100** (1993), no. 2, pp. 262–292.
- [ArTeYa97] Susumu Ariki, Tomohide Terasoma, Hiro-Fumi Yamada, *Higher Specht polynomials*, Hiroshima Mathematical Journal **27** (1997), pp. 177–188.
- [Verify26] Accompanying verification scripts and exact outputs in the Appendix (Section B). The standalone non-invariance check is `procesi-n8-nonstability-verification.py`; see `README.txt` for the supplementary checks and instructions.

B Ancillary files

B.1 README.txt

```
THE SPAN OF PROCESI'S PROPOSED BASIS NEED NOT BE INVARIANT
Rewritten 28 September 2026
=====

MANUSCRIPT
-----
procesi-noninvariant-span.tex
procesi-noninvariant-span.pdf

The rewritten note centers on non-invariance of  $W_{\{8,15\}}$ . It gives the
witness tableaux, the transposition (4,5), the full separating polynomial
as eight products of small determinants, and the alternating-pairing
argument that turns the exact calculation into a proof.

Small-case stability is confined to Remark 2.2. The seven-variable
relation is retained in Appendix A. The block-repair and polynomial-orbit
discussions have been removed from the manuscript.

The TeX file is self-contained; no separate bibliography file is needed.
Compile twice with a standard TeX installation:

    pdflatex procesi-noninvariant-span.tex
    pdflatex procesi-noninvariant-span.tex

The source uses amsmath, amssymb, amsthm, microtype, ytableau, geometry,
hyperref, and T1 font encoding. The compiled PDF has five pages at 12pt.
The earlier manuscript has not been overwritten.

MAIN RESULT: INDEPENDENT, STANDALONE CHECK
-----
Files:
    procesi-n8-nonstability-verification.py
    procesi-n8-nonstability-results.txt

Requirements: Python 3.10 or later and NumPy. From this directory, run:

    python procesi-n8-nonstability-verification.py

This script has no dependency on either of the other verification scripts.
Its recipe for  $g$  is written directly in the determinant notation of the
rewritten manuscript. It gives exactly the same integer polynomial as the
Schur-function presentation in the preceding version of the certificate.

The script checks that the witness tableaux are standard, the witness
has degree 15, the separator has degree 13, all 3736 indexed generators
are annihilated, and the permuted witness has functional value -1440.

No floating-point arithmetic, modular rank, Groebner reduction, or large
nullspace calculation is used. Fixed-width integer accumulation has an
```

explicit checked bound; remaining sums use arbitrary-precision integers. This check proves the main theorem independently of all small-case data. The exact dimension of $W_{\{8,15\}}$ is not needed and is not asserted.

OPTIONAL: SEVEN-VARIABLE RELATION AND SMALL CASES

Files:

procesi-conjecture-19-1-verification.py
procesi-conjecture-19-1-verification-results.txt

Requirements: Python 3.10 or later, NumPy, and SymPy. Run:

python procesi-conjecture-19-1-verification.py --all-degrees

Do not run this script with Python's -O option: it uses assertions. This is the earlier, unchanged audit. It checks the explicit $n=7$ relation over the integers, the exact degree-10 deficiency, stability, and full rank in all other degrees for $n \leq 7$. It also retains older block-repair and polynomial-orbit checks that are not discussed or needed in the rewritten manuscript.

OPTIONAL: DEGREE 10 FOR EIGHT VARIABLES

Files:

procesi-n8-degree10-stability-verification.py
procesi-n8-degree10-stability-results.txt

Requirements: Python 3.10 or later, NumPy, and SymPy. Keep all three .py files in the same directory, and run:

python procesi-n8-degree10-stability-verification.py

This proves that $W_{\{8,10\}}$ is stable, using complete orbits of annihilating covectors, exact annihilation identities, and complementary modular lower bounds. It does not infer a characteristic-zero upper bound from a small modular rank. This is only the optional small-case observation in Remark 2.2 and is not part of the proof of non-invariance in degree 15.

REPRODUCIBILITY AND SCOPE

All three .txt results files were regenerated successfully for this rewrite. The main result is an explicit counterexample at $n=8$, $d=15$; there is no assertion here that non-invariance holds for every $n \geq 8$ or in every degree. The five-page note contains all formulas needed to reproduce the main certificate without using the supplied implementation.

B.2 procesi-n8-nonstability-verification.py

```
#!/usr/bin/env python3
"""Exact counterexample to stability of the column-only span, n=8, degree=
15.
```

Run with Python 3.10+ and NumPy:

```
python procesi-n8-nonstability-verification.py
```

This is a standalone certificate. It does not use floating-point arithmetic, modular ranks, a Groebner basis, or a large nullspace computation.

For $I=\{i_1<\dots<i_m\}$, put $D_I(a_1,\dots,a_m)=\det(x_{\{i_r\}}^{\{a_s\}})$.

Let $A=\{1,5,6\}$, $B=\{2,7,8\}$, $C=\{3,4\}$. The separating polynomial is

$g = D_A(0,1,2) * q$,

where q is the following sum of eight products of small determinants:

$$\begin{aligned} &103 D_B(0,1,2) D_C(0,7) + 48 D_B(0,1,2) D_C(1,6) \\ &-32 D_B(0,2,3) D_C(2,3) +105 D_B(1,2,3) D_C(1,3) \\ &-32 D_B(0,1,6) D_C(1,2) + 77 D_B(1,2,4) D_C(1,2) \\ &-60 D_B(0,1,7) D_C(0,2) + 77 D_B(0,2,7) D_C(0,1). \end{aligned}$$

The determinant recipe matches the separating polynomial in the rewritten note,

"The span of Procesi's proposed basis need not be invariant."

Let $\text{Alt}(f)=\sum_{\sigma \in S_8} \text{sign}(\sigma)*\sigma(f)$ and $\Delta=\prod_{1 \leq i < j \leq 8} (x_j-x_i)$. For homogeneous f of degree 15, define $L(f)$ by $\text{Alt}(f*g)=L(f)*\Delta$. Since $\deg(g)=13$ and $\deg(\Delta)=28$, L annihilates the degree-15 part of the symmetric ideal: if $f=e_i*h$, then $\text{Alt}(f*g)=e_i*\text{Alt}(h*g)=0$ because $\deg(h*g)=28-i < 28$.

For a monomial x^a , $L(x^a)=\sum_b g_b*\epsilon(a+b)$, where $\epsilon(v)$ is the sign of the permutation v when v permutes $0,\dots,7$, and is zero otherwise. The NumPy table below computes these INTEGERS exactly.

The program checks all 3736 indexed column-only polynomials of degree 15, and then checks a specific generator F with $L((4,5)F)=-1440 \neq 0$. Thus $W_{\{8,15\}}$ is not S_8 -stable, over every characteristic-zero field.

Rows of tableaux are listed from top to bottom, and permutations act by $x_i \rightarrow x_{\{\sigma(i)\}}$, as in `procesi-noninvariant-span.tex`.

"""

```
from __future__ import annotations
```

```
from functools import lru_cache
```

```
from itertools import permutations, product
```

```
from time import perf_counter
```

```
from typing import Iterator
```

```
try:
```

```
    import numpy as np
```

```
except ImportError as exc:
```

```
    raise SystemExit('This certificate requires NumPy.') from exc
```

```
N = 8
```

```
Exponent = tuple[int, ...]
```

```
Tableau = tuple[tuple[int, ...], ...]
```

```
Polynomial = dict[Exponent, int]
```

```

# (integer coefficient, increasing exponents on B, increasing exponents on
#      C).
RECIPE = (
    (103, (0, 1, 2), (0, 7)),
    (48, (0, 1, 2), (1, 6)),
    (-32, (0, 2, 3), (2, 3)),
    (105, (1, 2, 3), (1, 3)),
    (-32, (0, 1, 6), (1, 2)),
    (77, (1, 2, 4), (1, 2)),
    (-60, (0, 1, 7), (0, 2)),
    (77, (0, 2, 7), (0, 1)),
)
BLOCKS = ((0, 4, 5), (1, 6, 7), (2, 3)) # zero-based A, B, C
S: Tableau = ((1, 3, 4, 7), (2, 5, 8), (6,))
T: Tableau = ((1, 2, 4, 5), (3, 6, 7), (8,))

def require(condition: bool, message: str) -> None:
    if not condition:
        raise RuntimeError(message)

def sign(p: tuple[int, ...]) -> int:
    return -1 if sum(p[i] > p[j] for i in range(len(p))
                    for j in range(i + 1, len(p))) % 2 else 1

@lru_cache(None)
def signed_permutations(m: int) -> tuple[tuple[tuple[int, ...], int], ...]:
    :
    return tuple((p, sign(p)) for p in permutations(range(m)))

def partitions(n: int, maximum: int | None = None) -> Iterator[tuple[int,
...]]:
    if n == 0:
        yield ()
        return
    if maximum is None:
        maximum = n
    for m in range(min(maximum, n), 0, -1):
        for tail in partitions(n - m, m):
            yield (m,) + tail

@lru_cache(None)
def tableaux(shape: tuple[int, ...]) -> tuple[Tableau, ...]:
    if not shape:
        return ((),)
    n = sum(shape)
    result = []
    for r, length in enumerate(shape):
        if r + 1 < len(shape) and length == shape[r + 1]:

```

```

        continue
    smaller = list(shape)
    smaller[r] -= 1
    if smaller[-1] == 0:
        smaller.pop()
    for t in tableaux(tuple(smaller)):
        rows = list(t)
        if r == len(rows):
            rows.append((n,))
        else:
            rows[r] = rows[r] + (n,)
        result.append(tuple(rows))
    return tuple(result)

@lru_cache(None)
def cocharge(t: Tableau) -> Tableau:
    n = sum(map(len, t))
    row = [0] * (n + 1)
    for r, entries in enumerate(t):
        for x in entries:
            row[x] = r
    labels = [0] * (n + 1)
    for i in range(2, n + 1):
        labels[i] = labels[i - 1] + int(row[i] > row[i - 1])
    return tuple(tuple(labels[x] for x in entries) for entries in t)

def degree(t: Tableau) -> int:
    return sum(map(sum, cocharge(t)))

def column_polynomial(s: Tableau, t: Tableau) -> Polynomial:
    exp = [0] * N
    for labels, entries in zip(cocharge(s), t):
        for c, x in zip(labels, entries):
            exp[x - 1] = c
    f = {tuple(exp): 1}
    for j in range(len(t[0])):
        block = tuple(row[j] - 1 for row in t if len(row) > j)
        out: Polynomial = {}
        for exp, coeff in f.items():
            for p, parity in signed_permutations(len(block)):
                permuted = list(exp)
                for i, k in enumerate(p):
                    permuted[block[k]] = exp[block[i]]
                key = tuple(permuted)
                out[key] = out.get(key, 0) + parity * coeff
        f = {exp: c for exp, c in out.items() if c}
    return f

def separating_polynomial() -> Polynomial:
    g: Polynomial = {}

```

```

for scalar, b_powers, c_powers in RECIPE:
    local = []
    for block, powers in zip(BLOCKS, ((0, 1, 2), b_powers, c_powers)):
        m = len(block)
        require(len(powers) == m, 'Determinant has the wrong size.')
        # Expand  $\det(x_i^{\text{powers}_j})$  by its signed permutations.
        local.append(tuple((tuple(powers[j] for j in p), parity)
                           for p, parity in signed_permutations(m)))
    for terms in product(*local):
        exp = [0] * N
        coeff = scalar
        for block, (powers, parity) in zip(BLOCKS, terms):
            coeff *= parity
            for i, power in zip(block, powers):
                exp[i] = power
        key = tuple(exp)
        g[key] = g.get(key, 0) + coeff
    return {exp: c for exp, c in g.items() if c}

def functional_table(g: Polynomial) -> tuple[np.ndarray, np.ndarray]:
    """Return  $L(x^a)$ , indexed by the base-8 encoding of  $a$  in  $\{0, \dots, 7\}^8$ .
    """
    perms = np.array(list(permutations(range(N))), dtype=np.int64)
    signs = np.array([sign(tuple(map(int, p))) for p in perms], dtype=np.int64)
    weights = np.array([N ** i for i in range(N)], dtype=np.int64)
    values = np.zeros(N ** N, dtype=np.int64)
    # For any fixed  $a$ , at most one contribution from each monomial of  $g$ 
    # reaches this table entry. This bounds every intermediate partial sum
    bound = sum(abs(c) for c in g.values())
    require(bound < 2 ** 63, 'Integer accumulator bound exceeds int64.')
    for exp, coeff in g.items():
        b = np.array(exp, dtype=np.int64)
        valid = np.all(perms >= b, axis=1)
        differences = perms[valid] - b
        indices = differences @ weights
        np.add.at(values, indices, coeff * signs[valid])
    require(int(np.max(np.abs(values))) <= bound, 'Accumulator bound
    violated.')
    return values, weights

def evaluate(f: Polynomial, values: np.ndarray, weights: np.ndarray) ->
    int:
    answer = 0 # Python integer: arbitrary precision.
    for exp, coeff in f.items():
        if any(e >= N or e < 0 for e in exp):
            continue
        index = sum(e * int(w) for e, w in zip(exp, weights))
        answer += coeff * int(values[index])
    return answer

```

```

def main() -> None:
    start = perf_counter()
    shape = tuple(map(len, S))
    require(S in tableaux(shape) and T in tableaux(shape), 'Witness
                                                    tableaux are not standard.')
    require(degree(S) == 15, 'Witness has the wrong cocharge degree.')
    g = separating_polynomial()
    require(all(sum(exp) == 13 for exp in g), 'Separator has the wrong
                                                    degree.')
    print(f'Separating polynomial: {len(g)} integer monomials, degree 13.'
          )
    values, weights = functional_table(g)
    print('Alternating top-coefficient functional constructed with exact
          integers.')

    count = 0
    for shape in partitions(N):
        ts = tableaux(shape)
        for s in ts:
            if degree(s) != 15:
                continue
            for t in ts:
                result = evaluate(column_polynomial(s, t), values, weights
                                )
                require(result == 0, f'Annihilation failed for S={s}, T={t
                                     }: {result}.')

                count += 1
    require(count == 3736, f'Unexpected number of family members: {count}.
                          ')
    print(f'All {count} indexed degree-15 column-only polynomials are
          annihilated.')

    f = column_polynomial(S, T)
    acted = {}
    for exp, coeff in f.items():
        e = list(exp)
        e[3], e[4] = e[4], e[3]
        acted[tuple(e)] = coeff
    before = evaluate(f, values, weights)
    after = evaluate(acted, values, weights)
    require(before == 0, 'Original witness is not annihilated.')
    require(after == -1440, f'Unexpected permuted-witness value: {after}.
                             )

    print('S = [1347 | 258 | 6]')
    print('T = [1245 | 367 | 8]')
    print('cc(S) = [0113 | 124 | 3]')
    print(f'L(F_T^S) = {before}')
    print(f'L((4,5) F_T^S) = {after}')
    print('CONCLUSION: W_{8,15} is not S_8-stable in characteristic zero.'
          )

    print(f'All checks passed in {perf_counter() - start:.2f} seconds.')

if __name__ == '__main__':
    main()

```

B.3 procesi-n8-nonstability-results.txt

```
Separating polynomial: 576 integer monomials, degree 13.
Alternating top-coefficient functional constructed with exact integers.
All 3736 indexed degree-15 column-only polynomials are annihilated.
S = [1347 | 258 | 6]
T = [1245 | 367 | 8]
cc(S) = [0113 | 124 | 3]
L(F_T^S) = 0
L((4,5) F_T^S) = -1440
CONCLUSION: W_{8,15} is not S_8-stable in characteristic zero.
All checks passed in 1.84 seconds.
```

B.4 procesi-n8-degree10-stability-verification.py

```
#!/usr/bin/env python3
"""Exact verification that W_{8,10} IS stable, despite failure in degree
    15.

Requirements: Python 3.10+, NumPy, SymPy. Keep these three scripts
    together:
    procesi-conjecture-19-1-verification.py
    procesi-n8-nonstability-verification.py
    procesi-n8-degree10-stability-verification.py
Run this script directly. No C/C++ compiler is required.

For k=2,3, respectively, use the degree-18 polynomials
    Delta_{1,...,6} Delta_{7,8} s_{(1,1)}(x7,x8),
    Delta_{1,...,5} Delta_{6,7,8} s_{(2,2,1)}(x6,x7,x8).
Their alternating top-coefficient functionals descend to R_{8,10}.
Each orbit, up to signs, is indexed by k-element subsets of {1,...,8}.
The script checks that all these covectors annihilate the proposed family.
Their combined rank is 48 modulo 1000003. The family rank is 2445 modulo
that prime, and 2445+48=2493=dim R_{8,10}. Hence both ranks are EXACT over
    Q,
and these orbit covectors form the entire annihilator of W_{8,10}.
That annihilator is stable, being a span of complete group orbits;
    therefore
W_{8,10} is stable as well. Extension of scalars covers characteristic
    zero.
"""
from __future__ import annotations
import importlib.util
from pathlib import Path
from itertools import combinations
from time import perf_counter
import numpy as np
```

```

ROOT = Path(__file__).resolve().parent

def load(name: str, filename: str):
    path = ROOT / filename
    if not path.is_file():
        raise SystemExit(f'Missing companion file: {filename}')
    spec = importlib.util.spec_from_file_location(name, path)
    module = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(module)
    return module

audit = load('original_audit', 'procesi-conjecture-19-1-verification.py')
sep = load('alternating_pairing', 'procesi-n8-nonstability-verification.py')

def canonical_separator(k: int, lam: tuple[int, ...]):
    m = 8 - k
    padded = lam + (0,) * (k - len(lam))
    e = tuple(j + padded[k - 1 - j] for j in range(k))
    return {tuple(p) + tuple(e[j] for j in q): s * t
            for p, s in sep.signed_permutations(m)
            for q, t in sep.signed_permutations(k)}

def main():
    start = perf_counter()
    M, labels, ring = audit.matrix(8, 10)
    sep.require(M.shape == (2493, 2493), 'Wrong family or ambient
                                         dimension.')
    A = np.array(ring.basis, dtype=np.int64)
    covector_blocks = []
    for k, lam in ((2, (1, 1)), (3, (2, 2, 1))):
        g = canonical_separator(k, lam)
        sep.require(all(sum(exp) == 18 for exp in g), 'Wrong separator
                                                         degree.')
        values, weights = sep.functional_table(g)
        columns = []
        for subset in combinations(range(8), k):
            order = tuple(j for j in range(8) if j not in subset) + subset
            # These are the orbit covectors, with harmless columnwise
            # signs
            # omitted. The inverse permutation on a monomial permutes its
            # exponent tuple by order.
            columns.append(values[A[:, order] @ weights])
        K = np.array(columns, dtype=np.int64).T
        bound = int(np.max(np.abs(M))) * M.shape[1] * int(np.max(np.abs(K))
                                                             )
        product = M @ K if bound < 2**63 else M.astype(object) @ K.astype(
            object)
        sep.require(not np.any(product), 'Exact annihilation failed.')
        print(f'k={k}: all {K.shape[1]} orbit covectors exactly annihilate
              the family.')
        covector_blocks.append(K)
    del values

```

```

K = np.hstack(covector_blocks)
lower_kernel = audit.rank(K)
lower_family = audit.rank(M)
sep.require(lower_kernel == 48 and lower_family == 2445, 'Unexpected
                modular ranks.')
print('Modular lower bounds: rank(M) >= 2445, rank(K) >= 48.')
print('Exact identity M K = 0 and 2445 + 48 = 2493 certify both ranks
                over Q.')
print('The full annihilator is spanned by complete orbits, hence is
                stable.')
print('CONCLUSION: W_{8,10} is S_8-stable, of dimension 2445 and
                codimension 48.')
print(f'All checks passed in {perf_counter() - start:.2f} seconds.')

if __name__ == '__main__':
    main()

```

B.5 procesi-n8-degree10-stability-results.txt

```

k=2: all 28 orbit covectors exactly annihilate the family.
k=3: all 56 orbit covectors exactly annihilate the family.
Modular lower bounds: rank(M) >= 2445, rank(K) >= 48.
Exact identity M K = 0 and 2445 + 48 = 2493 certify both ranks over Q.
The full annihilator is spanned by complete orbits, hence is stable.
CONCLUSION: W_{8,10} is S_8-stable, of dimension 2445 and codimension 48.
All checks passed in 25.01 seconds.

```

B.6 procesi-conjecture-19-1-verification.py

```

#!/usr/bin/env python3
"""Independent audit of procesi-conjecture-19-1.tex.

Requirements: Python 3.10+, NumPy, SymPy.
Run:
    python procesi-conjecture-19-1-verification.py
    python procesi-conjecture-19-1-verification.py --all-degrees

The first command verifies the relation, exact degree-10 rank, all six
block repairs, both alternative full families, exact S_7-stability,
the identity action of the transposition class sum on the quotient,
and exact dimensions of the two raw polynomial orbits. The second
also verifies all degreewise rank claims for 1 <= n <= 7.

All polynomials and normal forms are computed over the integers.
Modular full-rank minors give characteristic-zero lower bounds.
Upper bounds and stability are checked by exact integer identities.
In particular, small modular rank is NEVER used as an upper bound.
Rational reconstruction is only a candidate generator: the resulting

```

```

annihilator is checked against the original integer matrix.

Conventions: rows are written from top to bottom; permutations send
x_i to x_{sigma(i)}; lexicographic order has x_1 > ... > x_n.
The reduction basis is h_i(x_i,...,x_n), with leading term x_i^i.

Do not run Python with -O: this audit deliberately uses assertions.
The source manuscript is neither read nor modified by this script.
"""

from functools import lru_cache
from itertools import product, permutations
from collections import defaultdict, Counter
from fractions import Fraction
from math import gcd, isqrt, lcm
import argparse
import gc
import time

try:
    import numpy as np
    import sympy as sp
except ImportError as exc:
    raise SystemExit("Install NumPy and SymPy before running this audit.")
    from exc

@lru_cache(None)
def tableaux(shape):
    if not shape: return ((),)
    n=sum(shape); result=[]
    for r, length in enumerate(shape):
        if r+1<len(shape) and length==shape[r+1]: continue
        smaller=list(shape); smaller[r]-=1
        if smaller[-1]==0: smaller.pop()
        for t in tableaux(tuple(smaller)):
            rows=list(t)
            if r==len(rows): rows.append((n,))
            else: rows[r]=rows[r]+(n,)
            result.append(tuple(rows))
    return tuple(result)

def partitions(n,m=None):
    if n==0:
        yield (); return
    if m is None: m=n
    for i in range(min(m,n),1-1,-1):
        for t in partitions(n-i,i): yield (i,)+t

@lru_cache(None)
def cocharge(t):
    n=sum(map(len,t)); row=[0]*(n+1)
    for r,rr in enumerate(t):
        for x in rr: row[x]=r
    cc=[0]*(n+1)
    for x in range(2,n+1): cc[x]=cc[x-1]+(row[x]>row[x-1])

```

```

    return tuple(tuple(cc[x] for x in rr) for rr in t)

def deg(t): return sum(map(sum,cocharge(t)))
def text(t): return ','.join(''.join(map(str,r)) for r in t)
def parse(s): return tuple(tuple(map(int,r)) for r in s.split(','))

@lru_cache(None)
def comps(d,n):
    if n==0: return ((),) if d==0 else ()
    if n==1: return ((d,),)
    return tuple((i,)+c for i in range(d+1) for c in comps(d-i,n-1))

class Ring:
    def __init__(self,n,d):
        self.n=n; self.d=d
        self.basis=tuple(a for a in product(*(range(i+1) for i in range(n)
                                                )) if sum(a)==d)
        self.index={a:i for i,a in enumerate(self.basis)}
        self.tail={i:tuple((0,)*i+c for c in comps(i+1,n-i) if c[0]<i+1)
                    for i in range(n)}
        self.nf=lru_cache(None)(self._nf)
    def _nf(self,a):
        # x_n^n=0; prune before any other reduction.
        if a[-1]>=self.n: return ()
        for i in range(self.n):
            if a[i]>i: break
        else: return ((self.index[a],1),)
        base=list(a); base[i]-=i+1
        out=defaultdict(int)
        for b in self.tail[i]:
            c=tuple(x+y for x,y in zip(base,b))
            for j,coef in self.nf(c): out[j]-=coef
        return tuple((j,c) for j,c in out.items() if c)
    def reduce(self,f):
        out=[0]*len(self.basis)
        for a,c in f.items():
            if c:
                for j,v in self.nf(a): out[j]+=c*v
        return out

@lru_cache(None)
def signed_perms(n):
    out=[]
    for p in permutations(range(n)):
        sign=(-1)**sum(p[i]>p[j] for i in range(n) for j in range(i+1,n))
        out.append((p,sign))
    return tuple(out)

def add(f,g,scale=1):
    out=defaultdict(int,f)
    for a,c in g.items(): out[a]+=scale*c
    return {a:c for a,c in out.items() if c}

def action(f,p):

```

```

out={}
for a,c in f.items():
    b=[0]*len(a)
    for i,j in enumerate(p): b[j]=a[i]
    out[tuple(b)]=c
return out

def symmetrize(f,t,kind):
    n=sum(map(len,t))
    if kind=='col': blocks=[tuple(row[j]-1 for row in t if len(row)>j) for
                             j in range(len(t[0]))]
    else: blocks=[tuple(x-1 for x in row) for row in t]
    for block in blocks:
        if len(block)==1: continue
        out=defaultdict(int)
        for a,c in f.items():
            for p,s in signed_perms(len(block)):
                b=list(a)
                for i,j in enumerate(p): b[block[j]]=a[block[i]]
                out[tuple(b)]+=c*(s if kind=='col' else 1)
        f={a:c for a,c in out.items() if c}
    return f

def poly(s,t,kind='col'):
    n=sum(map(len,s)); a=[0]*n
    for cr,tr in zip(cocharge(s),t):
        for c,x in zip(cr,tr): a[x-1]=c
    f={tuple(a):1}
    if kind=='both': return symmetrize(symmetrize(f,t,'row'),t,'col')
    return symmetrize(f,t,kind)

def matrix(n,d,kind='col',ring=None):
    if ring is None: ring=Ring(n,d)
    rows=[]; labels=[]
    for shape in partitions(n):
        ts=tableaux(shape)
        ss=[s for s in ts if deg(s)==d]
        for s in ss:
            for t in ts:
                rows.append(ring.reduce(poly(s,t,kind)))
                labels.append((s,t))
    return np.array(rows,dtype=np.int64),labels,ring

def rref(a,p=1000003,full=False):
    a=np.array(a,dtype=np.int64)%p; r=0; piv=[]
    for col in range(a.shape[1]):
        candidates=np.flatnonzero(a[r:,col])
        if len(candidates)==0: continue
        j=r+int(candidates[0])
        if j!=r: a[[r,j]]=a[[j,r]]
        a[r]=(a[r]*pow(int(a[r,col]),-1,p))%p
        if full:
            ii=np.flatnonzero(a[:,col]); ii=ii[ii!=r]
        else:

```

```

        ii=np.flatnonzero(a[r+1:,col])+r+1
        # Individual modular operations stay below p^2 (int64 safe).
        if len(ii): a[ii,col:]=(a[ii,col:]-a[ii,col,None]*a[r,None,col:])%
            p
        piv.append(col); r+=1
        if r==a.shape[0]: break
    return a[:r],piv

def rank(a,p=1000003): return len(rref(a,p)[1])

TERMS=[(5,'134|26|5|7'),(3,'127|34|5|6'),(9,'124|37|5|6'),(-2,'126|35|4|7'),
        (-4,'134|256|7'),(16,'126|357|4'),(
        24,'124|357|6'),(-12,'134|26|57'),(-
        24,'124|35|67'),(-8,'126|35|47')]

def relation():
    aset=[t for t in tableaux((3,2,1)) if t[0][1]==3 or t[1][0]==3]
    h={}; terms=[]
    for c,st in TERMS:
        s=parse(st); shape=tuple(map(len,s))
        for u in aset:
            t=[list(r) for r in u]
            for i,l in enumerate(shape):
                if i==len(t): t.append([7])
                elif len(t[i])<1: t[i].append(7)
            t=tuple(map(tuple,t)); assert t in tableaux(shape)
            terms.append((c,s,t)); h=add(h,poly(s,t),c)
    return h,terms,aset

def trans(f,i,j):
    n=len(next(iter(f)))
    p=list(range(n)); p[i],p[j]=p[j],p[i]
    return action(f,p)

def zact(f):
    n=len(next(iter(f))); out={}
    for i in range(n):
        for j in range(i+1,n): out=add(out,trans(f,i,j))
    return out

def reconstruct(x,p):
    x=int(x)%p
    if x==0: return Fraction(0)
    bound=isqrt(p//2)
    a,b=p,x; u,v=0,1
    while abs(b)>bound:
        q=a//b; a,b=b,a-q*b; u,v=v,u-q*v
    if abs(v)>bound or not v or gcd(b,v)!=1: raise ValueError((x,p,b,v))
    z=Fraction(b,v)
    if (z.numerator-x*z.denominator)%p: raise ValueError('bad residue')
    return z

def orbit_certificate(f):

```

```

n=len(next(iter(f))); allmon=comps(sum(next(iter(f))),n); idx={a:i for
                                i,a in enumerate(allmon)}
p=1000003
vectors=[]; original=[]; echelon=[]; piv=[]
def consider(g):
    v=np.zeros(len(allmon),dtype=np.int64)
    for a,c in g.items(): v[idx[a]]=c
    w=v%p
    for row,col in zip(echelon,piv):
        if w[col]: w=(w-w[col]*row)%p
    support=np.flatnonzero(w)
    if not len(support): return
    col=int(support[0]); w=(w*pow(int(w[col]),-1,p))%p
    vectors.append(v); original.append(g); echelon.append(w); piv.
                                append(col)

consider(f)
i=0
while i<len(original):
    for j in range(n-1): consider(trans(original[i],j,j+1))
    i+=1
    if len(original)>60: raise RuntimeError('unexpected large orbit')
B=np.array(vectors,dtype=np.int64)
print('modular orbit dimension',len(original),'coordinate pivots',piv,
      flush=True)
inv=sp.Matrix(B[:,piv].tolist()).inv(method='DM')
den=sp.ilcm(*[v.q for v in inv]); invnum=np.array([[int(v) for v in
                                                    row] for row in (inv*den).tolist
                                                    ()],dtype=object)

BB=B.astype(object)
for g in original:
    for j in range(n-1):
        gg=trans(g,j,j+1); v=np.zeros(len(allmon),dtype=object)
        for a,c in gg.items(): v[idx[a]]=c
        coeff=v[piv]*invnum
        assert not np.any(coeff@BB-int(den)*v)
print('exact orbit dimension',len(original),'inverse denominator',den,
      flush=True)

return B,piv

def exact_annihilator(M, p=1000003):
    """Return an independent integer basis of the rational right kernel.
    """
    B, piv = rref(M, p, full=True)
    free = [i for i in range(M.shape[1]) if i not in piv]
    K = np.zeros((M.shape[1], len(free)), dtype=object)
    for j, col in enumerate(free):
        vals = [reconstruct(-B[i, col], p) for i in range(len(piv))]
        den = lcm(*(v.denominator for v in vals))
        K[col, j] = den
        for i, row in enumerate(piv):
            K[row, j] = int(vals[i] * den)
    # Reconstruction alone is not evidence: verify the exact identity.

```

```

assert not np.any(M.astype(object) @ K), "Exact annihilator check
                                     failed"
# The free-coordinate submatrix is diagonal and nonsingular.
assert all(K[col, j] > 0 for j, col in enumerate(free))
return K, free, len(piv)

def audit_degree10():
    start = time.time()
    R = Ring(7, 10)
    h, terms, aset = relation()
    assert len(aset) == 12 and len(terms) == 120
    assert len(set((s, t) for c, s, t in terms)) == 120
    assert all(deg(s) == 10 for c, s, t in terms)
    assert h and not any(R.reduce(h))
    print("Explicit relation: 120 distinct indexed summands; exact normal
                                     form zero.")

    zh = zact(h)
    assert zact(zh) == h
    hp, hm = add(h, zh), add(h, zh, -1)
    assert hp and hm
    print("Raw H: 1359 monomials; Z^2 H = H and both H+ and H- are nonzero
                                     .")

    M, labels, _ = matrix(7, 10, ring=R)
    assert M.shape == (573, 573)
    K, free, r = exact_annihilator(M)
    assert r == 552 and len(free) == 21
    print("Exact rank over Q: 552 (modular minor plus 21 exact
                                     annihilators).")
    # Also independently construct the stated 21 relations among the
                                     family.
    L, _, _ = exact_annihilator(M.T, p=1000000007)
    assert L.shape == (573, 21)
    print("Constructed and exactly verified 21 independent integer
                                     relations.")

    diagonal = [int(K[col, j]) for j, col in enumerate(free)]
    quotient_actions = []
    for i in range(6):
        acted = []
        for a in R.basis:
            b = list(a)
            b[i], b[i + 1] = b[i + 1], b[i]
            acted.append(R.reduce({tuple(b): 1}))
        # object arrays hold Python integers: no fixed-width overflow.
        A = np.array(acted, dtype=object)
        AK = A @ K
        assert not np.any(M.astype(object) @ AK), "W is not invariant"
        D = sp.Matrix([
            [sp.Rational(int(AK[col, k]), diagonal[k])
              for k in range(len(free))]
            for col in free
        ])

```

```

    assert sp.trace(D) == 1
    quotient_actions.append(D)
print("All six adjacent transpositions preserve W, checked exactly.")
print("Their quotient traces are all exactly 1.")
Z = sp.zeros(21)
for i in range(6):
    transposition = quotient_actions[i]
    for j in range(i + 1, 7):
        if j > i + 1:
            transposition = (quotient_actions[j - 1] * transposition
                             * quotient_actions[j - 1])
        Z += transposition
assert Z == sp.eye(21)
content_one = [sh for sh in partitions(7)
                if sum(c - r for r, length in enumerate(sh)
                      for c in range(length)) == 1]
assert content_one == [(3, 3, 1)]
assert len(tableaux((3, 3, 1))) == 21
print("Z acts exactly as the identity on the quotient; its type is (3,
      3,1).")

s1, s2, s3 = [parse(x) for x in
              ('126|357|4', '124|357|6', '134|256|7')]
assert set(t for t in tableaux((3, 3, 1)) if deg(t) == 10) == {s1, s2,
s3}

replacement = {
    kind: np.array([R.reduce(poly(s2, t, kind))
                    for t in tableaux((3, 3, 1))], dtype=np.int64)
    for kind in ('row', 'both')
}
for s in (s1, s2, s3):
    reduced = M[[ss != s for ss, t in labels]]
    assert reduced.shape == (552, 573)
    assert rank(reduced) == 552
    for kind in ('row', 'both'):
        assert rank(np.vstack([reduced, replacement[kind]])) == 573
    print(f"Remove block {text(s)}: rank 552; either stated repair has
          rank 573.")

for kind in ('row', 'both'):
    N, _, _ = matrix(7, 10, kind, R)
    assert rank(N) == 573
    print(f"Full {kind} family has rank 573.")

print("Exact raw orbit check for H+:")
B, _ = orbit_certificate(hp)
assert B.shape[0] == 21
print("Exact raw orbit check for H-:")
B, _ = orbit_certificate(hm)
assert B.shape[0] == 21
R.nf.cache_clear()
print(f"Degree-10 audit passed in {time.time() - start:.1f} seconds.",
      flush=True)

```

```

def audit_all_degrees():
    start = time.time()
    for n in range(1, 8):
        records = []
        for d in range(n * (n - 1) // 2 + 1):
            R = Ring(n, d)
            M, labels, _ = matrix(n, d, ring=R)
            size, r = len(labels), rank(M)
            assert size == len(R.basis)
            assert r == size - (21 if (n, d) == (7, 10) else 0), (n, d,
                                                                    size, r)

            records.append(f"{d}:{r}/{size}")
            if n == 7:
                print(f"  n=7 degree={d}, rank={r}, dimension={size}",
                      flush=True)

            R.nf.cache_clear()
            del M, labels, R
            gc.collect()
        print(f"n={n}, degree:rank/dimension: " + ", ".join(records),
              flush=True)
    print(f"All degreewise claims passed in {time.time() - start:.1f}
          seconds.")

def main():
    parser = argparse.ArgumentParser(description=__doc__)
    parser.add_argument('--all-degrees', action='store_true',
                        help='also test every degree for 1 <= n <= 7')
    args = parser.parse_args()
    if not __debug__:
        raise SystemExit('Run without -O so that verification assertions
                          execute.')

    audit_degree10()
    if args.all_degrees:
        audit_all_degrees()
    print('ALL REQUESTED CHECKS PASSED.')

if __name__ == '__main__':
    main()

```

B.7 procesi-conjecture-19-1-verification-results.txt

Explicit relation: 120 distinct indexed summands; exact normal form zero.
 Raw H: 1359 monomials; $Z^2 H = H$ and both H^+ and H^- are nonzero.
 Exact rank over $\mathbb{Q}$: 552 (modular minor plus 21 exact annihilators).
 Constructed and exactly verified 21 independent integer relations.
 All six adjacent transpositions preserve W , checked exactly.
 Their quotient traces are all exactly 1.
 Z acts exactly as the identity on the quotient; its type is $(3,3,1)$.
 Remove block 126|357|4: rank 552; either stated repair has rank 573.

```

Remove block 124|357|6: rank 552; either stated repair has rank 573.
Remove block 134|256|7: rank 552; either stated repair has rank 573.
Full row family has rank 573.
Full both family has rank 573.
Exact raw orbit check for H+:
modular orbit dimension 21 coordinate pivots [361, 1076, 397, 367, 1083,
                                              1082, 368, 403, 404, 408, 568, 1112,
                                              1118, 1241, 409, 569, 596, 1247, 1397
                                              , 1311, 1403]
exact orbit dimension 21 inverse denominator 8640
Exact raw orbit check for H-:
modular orbit dimension 21 coordinate pivots [368, 1083, 397, 367, 1076,
                                              1082, 361, 408, 1112, 1241, 1247,
                                              1311, 1397, 3078, 1123, 1408, 3243,
                                              1431, 3084, 3249, 3313]
exact orbit dimension 21 inverse denominator 720
Degree-10 audit passed in 6.3 seconds.
n=1, degree:rank/dimension: 0:1/1
n=2, degree:rank/dimension: 0:1/1, 1:1/1
n=3, degree:rank/dimension: 0:1/1, 1:2/2, 2:2/2, 3:1/1
n=4, degree:rank/dimension: 0:1/1, 1:3/3, 2:5/5, 3:6/6, 4:5/5, 5:3/3, 6:1/
    1
n=5, degree:rank/dimension: 0:1/1, 1:4/4, 2:9/9, 3:15/15, 4:20/20, 5:22/22
    , 6:20/20, 7:15/15, 8:9/9, 9:4/4, 10:
    1/1
n=6, degree:rank/dimension: 0:1/1, 1:5/5, 2:14/14, 3:29/29, 4:49/49, 5:71/
    71, 6:90/90, 7:101/101, 8:101/101, 9:
    90/90, 10:71/71, 11:49/49, 12:29/29,
    13:14/14, 14:5/5, 15:1/1

n=7 degree=0, rank=1, dimension=1
n=7 degree=1, rank=6, dimension=6
n=7 degree=2, rank=20, dimension=20
n=7 degree=3, rank=49, dimension=49
n=7 degree=4, rank=98, dimension=98
n=7 degree=5, rank=169, dimension=169
n=7 degree=6, rank=259, dimension=259
n=7 degree=7, rank=359, dimension=359
n=7 degree=8, rank=455, dimension=455
n=7 degree=9, rank=531, dimension=531
n=7 degree=10, rank=552, dimension=573
n=7 degree=11, rank=573, dimension=573
n=7 degree=12, rank=531, dimension=531
n=7 degree=13, rank=455, dimension=455
n=7 degree=14, rank=359, dimension=359
n=7 degree=15, rank=259, dimension=259
n=7 degree=16, rank=169, dimension=169
n=7 degree=17, rank=98, dimension=98
n=7 degree=18, rank=49, dimension=49
n=7 degree=19, rank=20, dimension=20
n=7 degree=20, rank=6, dimension=6
n=7 degree=21, rank=1, dimension=1
n=7, degree:rank/dimension: 0:1/1, 1:6/6, 2:20/20, 3:49/49, 4:98/98, 5:169
    /169, 6:259/259, 7:359/359, 8:455/455
    , 9:531/531, 10:552/573, 11:573/573,

```

```

12:531/531, 13:455/455, 14:359/359,
15:259/259, 16:169/169, 17:98/98, 18:
49/49, 19:20/20, 20:6/6, 21:1/1
All degreewise claims passed in 14.8 seconds.
ALL REQUESTED CHECKS PASSED.

```

B.8 sagemath-verification.py

This is my (DG)'s independent verification of the claims in this note.

```

n = 8
N = (n*(n-1))//2
d = 15
Pn = PolynomialRing(QQ, "x", n)
xs = Pn.gens()

def Alt_coefficient(h):
    # This is Alt(h) / Delta, assuming that h is a polynomial
    # of degree \leq N in Pn.
    res = 0
    for c, m in h:
        mex = m.exponents()[0]
        if len(set(mex)) != len(mex):
            continue
        per = Permutation([exp+1 for exp in mex])
        res += c * per.sign()
    return res

def cocharge_tableau(T):
    # Cocharge tableau of a standard tableau T.
    # Encoded as a tuple of tuples to save time.
    n = sum(len(row) for row in T)
    dess = T.standard_descents()
    cocharges = [sum(1 for i in dess if i < k)
                  for k in range(1, n+1)]
    return tuple(tuple(cocharges[ent-1] for ent in row)
                  for row in T)

S = StandardTableau([[1,3,4,7],[2,5,8],[6]])
T = StandardTableau([[1,2,4,5],[3,6,7],[8]])
T45 = Tableau([[1,2,5,4],[3,6,7],[8]])

def FTS(T, ccS):
    # the polynomial F_T^S,
    # taking the cocharge tableau ccS of S as input instead of S
    ColT = T.column_stabilizer()
    res = Pn.zero()
    for w in ColT:
        res += w.sign() * Pn.prod(xs[w(Tent)-1]**(ccSent)
                                   for (Trow, ccSrow) in zip(T, ccS)
                                   for (Tent, ccSent) in zip(Trow, ccSrow))
    return res

```

```

def DIa(I, a):
    # D_I(a), where I is a subset (as tuple) and a is a tuple.
    I = sorted(I)
    M = Matrix(Pn, [[xs[i_r-1]**a_s for i_r in I]
                    for a_s in a])
    return M.det()

q = DIa((2,7,8), (0,1,2)) * (103 * DIa((3,4), (0,7)) + 48 * DIa((3,4), (1,
6))) \
- 32 * DIa((2,7,8), (0,2,3)) * DIa((3,4), (2,3)) \
+ 105 * DIa((2,7,8), (1,2,3)) * DIa((3,4), (1,3)) \
+ (-32 * DIa((2,7,8), (0,1,6)) + 77*DIa((2,7,8), (1,2,4))) * DIa((3,4),
(1,2)) \
- 60 * DIa((2,7,8), (0,1,7)) * DIa((3,4), (0,2)) \
+ 77 * DIa((2,7,8), (0,2,7)) * DIa((3,4), (0,1))

g = DIa((1,5,6), (0,1,2)) * q

def Lg(h):
    return Alt_coefficient(g*h)

assert(Lg(FTS(T45, cocharge_tableau(S))) == -1440) # non-vanishing
for lam in Partitions(8):
    for S2 in StandardTableaux(lam):
        ccS2 = cocharge_tableau(S2)
        if sum(ent for row in ccS2 for ent in row) != 15:
            continue # wrong degree
        for T2 in StandardTableaux(lam):
            assert(Lg(FTS(T2, ccS2)) == 0)
    print("done with " + str(lam))
print("done")

```