

12. Finite state machines and vector recurrences

Combinatorics

Fall 2026

At the end of [Lecture Note 11](#) we met counting problems where a recurrence for the total count is hard to find directly: strings with overlapping forbidden substrings, and tilings of a $3 \times n$ board by dominoes. This note develops a systematic method for such problems. We will carry the following problem through the note.

Problem 1 (Strings avoiding ABA and BC). How many strings of length n over the alphabet $\{A, B, C\}$ contain neither ABA nor BC as a block of consecutive letters?

Call such a string *valid*. As in [Lecture Note 11](#), we think of building a string one letter at a time, from left to right, and ask which letters can be written next. Try it with the valid string CAB: writing A next would complete ABA, and writing C would complete BC, so only B can come next. After CCB, the letters A and B can come next. After CBA, all three letters can. So the number of ways to continue depends on how the string built so far ends, and a single total count of valid strings of length $n - 1$ does not tell us the total for length n . Instead, we will keep a separate count for each kind of ending, find equations linking these counts, and only at the end combine them into a recurrence for the total.

After studying these notes, you will be able to do the following.

- Choose states that record the information needed to decide which extensions of a string, walk, or partial tiling are allowed.
- Construct a finite state machine and its transition table for a list of forbidden substrings.
- Convert a transition table into a transition matrix and a vector recurrence, with the correct initial vector and final sum.
- Use the Cayley–Hamilton theorem to derive a scalar recurrence from a vector recurrence, and check where it holds.
- Compute a matrix power by repeated squaring and count the multiplications it uses.
- Set up boundary states and transition matrices for tilings of a $3 \times n$ board by dominoes, and by dominoes and angles.

12.1 Finite state machines

12.1.1 Strings

A **string** over a finite alphabet Σ is a finite sequence of letters from Σ , written without commas; its **length** is the number of letters. The string with no letters at all is called the **empty string**; it has length 0, and we denote it by ε (the Greek letter epsilon), since a blank space on the page would be invisible. Every string is built from ε by writing its letters one at a time. We write wx for the string w followed by the letter x . A **substring** of w is a block of consecutive letters of w , a **prefix** is a block at the start, and a **suffix** is a block at the end; the empty string counts as all three. A prefix is **proper** if it is not the whole string. For example, the suffixes of CAB are ε , B, AB, CAB, and the proper prefixes of ABA are ε , A, AB.

12.1.2 Keeping notes while building: walks

We start with an example where the bookkeeping is simple. In [Lecture Note 11](#) we counted walks in $\mathbb{Z}^2$ with steps $L = (-1, 0)$, $U = (0, 1)$, $R = (1, 0)$ that never contain LR or RL. A walk is a string over $\{L, U, R\}$, built one step at a time.

While building a walk, we keep a one-symbol note: the last step taken. Before the first step there is no last step, and we write the note ε . Once a forbidden pair has been written, the walk is ruined for good, and we write the note $\dagger$ (read “dagger”). Each new step updates the note according to this table, whose row is the current note and whose column is the step written next:

note	L	U	R
ε	L	U	R
L	L	U	$\dagger$
U	L	U	R
R	$\dagger$	U	R
$\dagger$	$\dagger$	$\dagger$	$\dagger$

For example, building LUR produces the notes ε, L, U, R , and the walk is valid. Building ULR produces $\varepsilon, U, L, \dagger$, and the walk is not valid.

Why does this work? Whether the next step is allowed depends only on the last step, and the note records exactly the last step. So the note always tells us what we need, and we never have to look further back in the walk.

Definition 2 (Deterministic finite state machine). A **deterministic finite state machine** over an alphabet Σ consists of a finite set of **states** (the possible notes), a **start state**, a set of **accepting states**, and a **transition function** δ that assigns to each state p and letter x exactly one next state $\delta(p, x)$. The machine reads a string from left to right, starting in the start state and moving from p to $\delta(p, x)$ on each letter x . The state reached at the end is the **state of the string**, and the machine **accepts** the string if that state is accepting.

The table above is the **transition table** of the walk machine: its states are $\varepsilon, L, U, R, \dagger$, the start state is ε , and every state except $\dagger$ is accepting. *Deterministic* means that the current state and the next letter decide the next state. Many strings share a state; that is the point, since the state should remember only what affects the future.

12.1.3 A machine for strings avoiding ABA and BC

For [Problem 1](#), the last letter is not a good enough note: CB and AB both end in B, but A can be written after the first and not after the second. Instead, the note records how much of a forbidden block sits at the end of the string built so far: nothing (ε), A, B, or AB. The machine is

state	A	B	C
ε	A	B	ε
A	A	AB	ε
B	A	B	$\dagger$
AB	$\dagger$	B	$\dagger$
$\dagger$	$\dagger$	$\dagger$	$\dagger$

with start state ε , rejecting state $\dagger$, and the other four states accepting. For example, building CABBA produces

$$\varepsilon \xrightarrow{C} \varepsilon \xrightarrow{A} A \xrightarrow{B} AB \xrightarrow{B} B \xrightarrow{A} A,$$

so CABBA is valid; building CABA ends with $AB \xrightarrow{A} \dagger$, so CABA is not.

How was the table made? Each entry comes from writing the letter after the state’s name and reading off the new ending. From AB, the letter C gives ABC, which ends in BC, so the entry is $\dagger$. From B, the letter A gives BA; this contains no forbidden block, and the part of a forbidden block at its end is A, so the entry is A.

From A, the letter C gives AC, which ends in no part of a forbidden block, so the entry is ε . In general, the states are the proper prefixes of the forbidden blocks,

$$P = \{\varepsilon, A, B, AB\},$$

and from a state $p \in P$ the letter x leads to $\dagger$ if px ends in a forbidden block, and otherwise to the longest suffix of px that lies in P .

Why does this work? A forbidden block is completed by the letter just written, so just before that letter, a proper prefix of the block (an element of P) was sitting at the end of the string. The note records the longest such ending, so it contains every shorter one, and nothing earlier in the string can matter. For instance, CCAB and BBCAB both have note AB and can be continued in exactly the same ways as the string AB itself. Therefore the machine accepts exactly the valid strings. Note that state ε does not mean the string is empty: after the first letter, it means the string ends in C.

12.2 Counting with a transition matrix

12.2.1 Strings avoiding ABA and BC

We now count valid strings separately by state. Let

$$\varepsilon_n, \quad A_n, \quad B_n, \quad AB_n$$

be the numbers of valid strings of length n in states ε, A, B, AB . These symbols denote numbers, not strings: for instance AB_n is the number of valid strings of length n ending in AB, and ε_n counts the empty string when $n = 0$ and the valid strings ending in C when $n \geq 1$. The total is $s_n = \varepsilon_n + A_n + B_n + AB_n$.

The counts satisfy

$$\begin{aligned} \varepsilon_n &= \varepsilon_{n-1} + A_{n-1}, & A_n &= \varepsilon_{n-1} + A_{n-1} + B_{n-1}, \\ B_n &= \varepsilon_{n-1} + B_{n-1} + AB_{n-1}, & AB_n &= A_{n-1}, \end{aligned} \quad (n \geq 1),$$

with $\varepsilon_0 = 1$ and $A_0 = B_0 = AB_0 = 0$ (the empty string is in state ε). Let us use them before explaining them:

n	ε_n	A_n	B_n	AB_n	s_n
0	1	0	0	0	1
1	1	1	1	0	3
2	2	3	2	1	8
3	5	7	5	3	20

We can check $s_3 = 20$ by hand: of the 27 strings of length 3, the six of the form BCx or xBC and the string ABA are invalid.

Where do the equations come from? Look at the transition table for the entries equal to B: they are in the rows of ε, B , and AB, in column B. So a valid string of length n in state B is built by writing B after a valid string of length $n - 1$ in state ε, B , or AB. Each such string is built in exactly one way, since removing its last letter recovers the shorter string. Hence $B_n = \varepsilon_{n-1} + B_{n-1} + AB_{n-1}$, and the other three equations come from the other three states in the same way. Strings in state $\dagger$ never contribute, since no letter leads from $\dagger$ back to a valid state.

The four equations are one matrix equation. With the states in the order ε, A, B, AB ,

$$v_n = \begin{pmatrix} \varepsilon_n \\ A_n \\ B_n \\ AB_n \end{pmatrix}, \quad v_n = M v_{n-1} \quad (n \geq 1), \quad M = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \quad v_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

This is a **vector recurrence**, and M is the **transition matrix**. You can read it directly off the transition table: the entry in the row of state q and the column of state p is the number of letters leading from p to q . For instance, the last column has a single 1, in the row of B, because from AB only the letter B keeps the string valid. For other machines the recipe is the same, and an entry can exceed 1 when several letters lead from p to the same q .

Since $v_n = M^n v_0$, Sage can compute far ahead. In [SageCell](#), ZZ denotes the integers and a caret denotes a power. Later code blocks use matrices defined in earlier ones, so paste the earlier blocks first.

```
M = matrix(ZZ, [[1, 1, 0, 0],
                [1, 1, 1, 0],
                [1, 0, 1, 1],
                [0, 1, 0, 0]])
v0 = vector(ZZ, [1, 0, 0, 0])
show([sum(M^n * v0) for n in range(9)])
```

Sage displays $(s_0, \dots, s_8) = (1, 3, 8, 20, 49, 120, 294, 720, 1763)$.

12.2.2 Walks

For walks, let L_n, U_n, R_n be the numbers of valid walks of length n ending in L, U, R, and let t_n be the total. Reading the walk table the same way,

$$\begin{pmatrix} L_n \\ U_n \\ R_n \end{pmatrix} = W \begin{pmatrix} L_{n-1} \\ U_{n-1} \\ R_{n-1} \end{pmatrix} \quad (n \geq 2), \quad W = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}, \quad \begin{pmatrix} L_1 \\ U_1 \\ R_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

This gives $t_1 = 3, t_2 = 7, t_3 = 17$, as in [Lecture Note 11](#). We start at $n = 1$ because the start state ε is used only by the empty walk.

12.3 From a vector recurrence to a scalar recurrence

The vector recurrence computes s_n , but to use the methods of [Lecture Note 12](#) we want a recurrence for (s_n) alone. Here is how to get one. Ask Sage for the characteristic polynomial of M :

```
show(M.charpoly('x'))
```

Sage displays $x^4 - 3x^3 + 2x^2 - 2x + 1$. Read its coefficients as a recurrence of order 4:

$$s_n = 3s_{n-1} - 2s_{n-2} + 2s_{n-3} - s_{n-4} \quad (n \geq 4), \quad s_0 = 1, s_1 = 3, s_2 = 8, s_3 = 20.$$

Let us test it on the values computed above:

$$s_4 = 3 \cdot 20 - 2 \cdot 8 + 2 \cdot 3 - 1 = 49, \quad s_5 = 3 \cdot 49 - 2 \cdot 20 + 2 \cdot 8 - 3 = 120.$$

Both agree. This recurrence answers [Problem 1](#). Its negative coefficients do not count anything; they appear because four separate counts have been combined into one equation.

Why does this work? Recall that the **characteristic polynomial** of a $d \times d$ matrix Q is $\chi_Q(x) = \det(xI - Q)$. We need one theorem from linear algebra, which we use without proof.

Theorem 3 (Cayley–Hamilton). Every square matrix Q satisfies its own characteristic polynomial: if $\chi_Q(x) = x^d + c_{d-1}x^{d-1} + \dots + c_1x + c_0$, then

$$Q^d + c_{d-1}Q^{d-1} + \dots + c_1Q + c_0I = 0,$$

the zero matrix.

For example, $F = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ has $\chi_F(x) = x^2 - x - 1$, and indeed $F^2 = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} = F + I$. For our matrix the theorem says $M^4 - 3M^3 + 2M^2 - 2M + I = 0$. Multiply this equation on the right by v_{n-4} , where $n \geq 4$. Since $Mv_{n-4} = v_{n-3}$, $M^2v_{n-4} = v_{n-2}$, and so on, we get

$$v_n - 3v_{n-1} + 2v_{n-2} - 2v_{n-3} + v_{n-4} = 0.$$

This is an equation between vectors, so it holds in every coordinate: the sequences (ε_n) , (A_n) , (B_n) , (AB_n) all satisfy the same recurrence. Adding the four coordinates gives it for s_n . The same argument works for any matrix.

Theorem 4 (Scalar recurrence from a transition matrix). Suppose $v_n = Qv_{n-1}$ for $n \geq 1$, where Q is a $d \times d$ matrix with $\chi_Q(x) = x^d + c_{d-1}x^{d-1} + \dots + c_0$. Let a_n be a fixed combination of the coordinates of v_n (their sum, or a single coordinate, for example). Then

$$a_n + c_{d-1}a_{n-1} + \dots + c_1a_{n-d+1} + c_0a_{n-d} = 0 \quad (n \geq d).$$

Two cautions, both visible for walks. Sage gives $\chi_W(x) = x^3 - 3x^2 + x + 1$, hence $t_n = 3t_{n-1} - t_{n-2} - t_{n-3}$. First, our vector recurrence for walks starts one step late (its first vector is the one for length 1), so the theorem guarantees this only for $n \geq 4$; the case $n = 3$ must be checked by hand, and it holds: $3 \cdot 7 - 3 - 1 = 17 = t_3$. Second, the recurrence need not be the shortest one: in [Lecture Note 11](#) we found $t_n = 2t_{n-1} + t_{n-2}$. The two agree, because $\chi_W(x) = (x-1)(x^2 - 2x - 1)$ and the cubic recurrence is the difference of two consecutive instances of the quadratic one.

Remark 5 (Companion matrices). Conversely, every recurrence becomes a vector recurrence if we store several consecutive terms. For example, $t_n = 2t_{n-1} + t_{n-2}$ becomes

$$\begin{pmatrix} t_n \\ t_{n-1} \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} t_{n-1} \\ t_{n-2} \end{pmatrix}.$$

The first row computes the new term; the second row shifts the old term down. This matrix, the **companion matrix** of the recurrence, has characteristic polynomial $x^2 - 2x - 1$, the characteristic polynomial of the recurrence. The same works for any order d , with a $d \times d$ matrix whose first row holds the coefficients and whose other rows shift.

12.4 Computing matrix powers

Computing v_n one step at a time takes n multiplications by M . For very large n there is a faster way to compute M^n . Here it is for Q^{13} . Square three times,

$$Q^2 = Q \cdot Q, \quad Q^4 = Q^2 \cdot Q^2, \quad Q^8 = Q^4 \cdot Q^4,$$

and then multiply twice, $Q^{13} = Q \cdot Q^4 \cdot Q^8$. That is five matrix multiplications instead of the twelve in $Q \cdot Q \dots Q$. This works because $13 = 1 + 4 + 8$ in binary (1101_2) , and $Q^a Q^b = Q^{a+b}$.

Definition 6 (Repeated squaring). To compute Q^n for $n \geq 1$:

1. Set $P = I$, $S = Q$, and $r = n$.
2. If r is odd, replace P by PS .
3. Replace r by $\lfloor r/2 \rfloor$. If $r = 0$, stop and output P .
4. Replace S by S^2 and go to step 2.

For $n = 13$ the procedure carries out exactly the computation above:

r	S	P after step 2
13	Q	Q
6	Q^2	Q
3	Q^4	Q^5
1	Q^8	Q^{13}

Why does it work? S runs through $Q, Q^2, Q^4, \dots$; checking whether r is odd and then halving it reads the binary digits of n from right to left; and P collects the powers S whose digit is 1. If n has $k + 1$ binary digits, there are k squarings and at most k other multiplications, so at most $2 \log_2 n$ matrix multiplications in all. We say the count is $O(\log n)$, meaning at most a constant times $\log n$. Only P , S , and r are stored.

This counts matrix multiplications, not work on digits. Our counts grow exponentially, so their number of digits grows in proportion to n ; for example s_{100} has 40 digits. If we only need counts modulo a fixed integer, reducing after every multiplication keeps the entries small.

12.5 Tilings of three-row boards

12.5.1 Tiles and boundary states

A $3 \times n$ board has rows 0, 1, 2 from top to bottom and columns $1, \dots, n$. A **tiling** covers every cell exactly once by tiles lying on the board; tiles may be rotated, and tiles of the same shape are not distinguished. The 3×0 board has one tiling, the empty one. We use **dominoes** (two cells sharing a side) and **angles** (three cells of a 2×2 square), in all orientations (Figure 1).

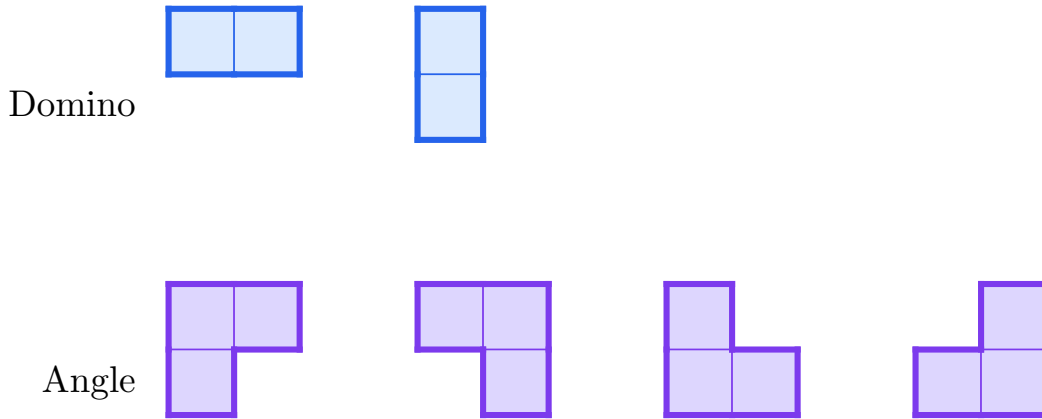


Figure 1: The two domino orientations and the four angle orientations. Thin lines separate cells within a tile; thick colored lines outline the tile. The tiling diagrams below use the same convention.

Figure description: Two rows of shapes show both orientations of a domino and all four rotations of an L-shaped angle.

Thick outlines distinguish a whole tile from its constituent cells.

We build a tiling column by column, the way we built strings letter by letter. Figure 2 follows a domino tiling of the 3×4 board. After column 1 is finished, a horizontal domino in the top row already covers the top cell of column 2. Next we fill the rest of column 2 with two horizontal dominoes, in the middle and bottom rows; these stick out and cover the middle and bottom cells of column 3. And so on, until the board is full and nothing sticks out.

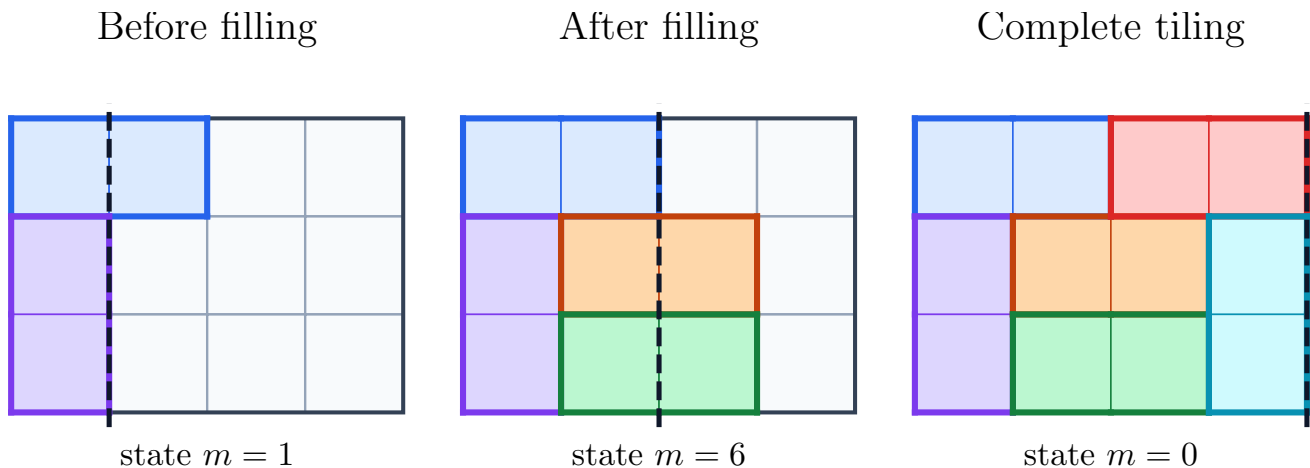


Figure 2: Moving the cut through a 3×4 domino tiling. The dashed line separates completed columns on the left from the columns still to be processed on the right; only the tiles already placed are colored.

Figure description: Three panels of the same four-column board. A cut after column one leaves the next top cell occupied, state one. A cut after column two leaves the next middle and bottom cells occupied, state six. The complete six-domino tiling has its cut at the right edge and state zero.

The note we keep is the set of cells in the next column that are already covered; we call it the **boundary state**. We record it as a number,

$$m = b_0 + 2b_1 + 4b_2, \quad b_r = 1 \text{ if the cell in row } r \text{ is covered,}$$

so in the figure the states are 1 (top cell covered), then $2 + 4 = 6$ (middle and bottom covered), then 0 at the end. All eight states are shown in [Figure 3](#). Why is this the right note? Tiles placed later only need to avoid the cells already covered; which tiles covered them does not matter. Since dominoes and angles are at most two columns wide, a tile that starts in the current column can reach only the next column, so nothing is covered further to the right.

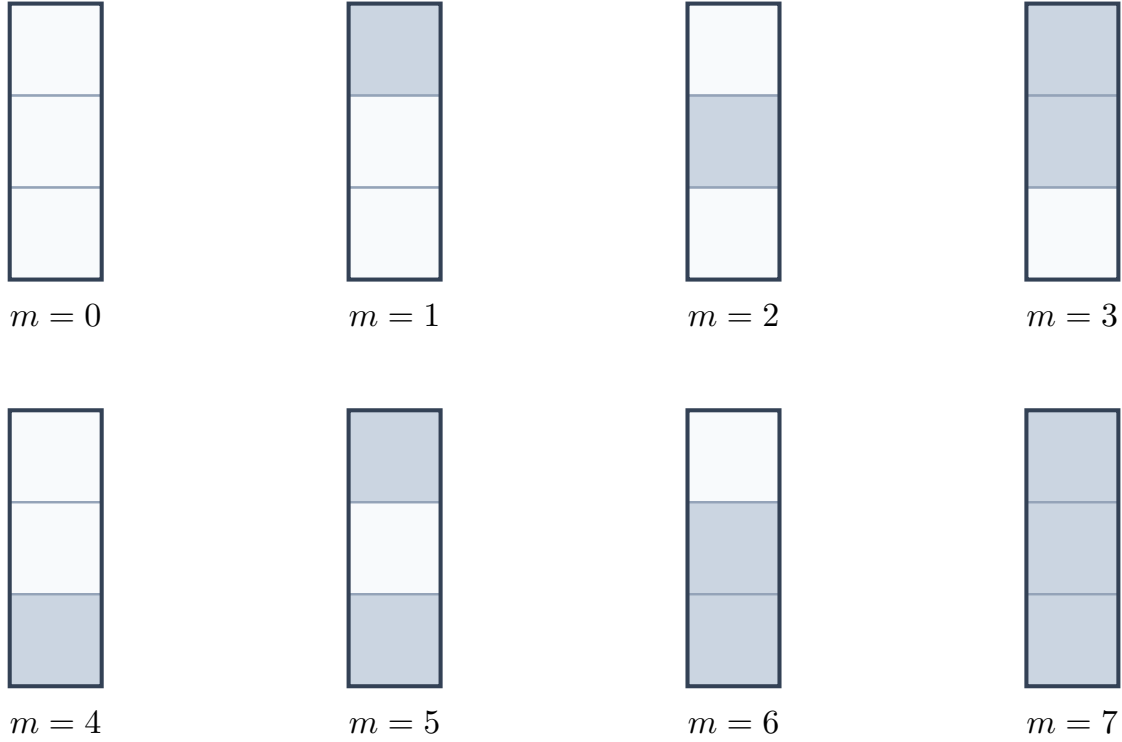


Figure 3: All eight one-column states, in numerical order. Gray cells are already covered by tiles crossing the cut from the left; white cells still need to be covered. The top, middle, and bottom covered cells contribute 1, 2, and 4, respectively.

Figure description: Eight three-cell columns labelled m equals zero through seven. Gray cells mark the subset encoded by each mask: none; top; middle; top and middle; bottom; top and bottom; middle and bottom; all three.

One step of the building process places the tiles whose leftmost column is the current column. They cover its uncovered cells and may stick out into the next column. We call such a set of tiles a **filling from m to q** , where m is the current state and q , the **outgoing state**, records the cells they cover in the next column. In the figure, two horizontal dominoes form a filling from 1 to 6.

Now we count exactly as for strings. Let $A_{q,m}$ be the number of fillings from m to q , and let A be the 8×8 matrix with these entries. Then

$$\text{number of tilings of the } 3 \times n \text{ board} = (A^n)_{0,0}, \quad (1)$$

the entry in row 0 and column 0 of A^n . The reason is the same as for strings: if X_k is the vector of counts of partially built tilings with k finished columns, sorted by state, then $X_k = AX_{k-1}$, because each partially built tiling with k columns comes from exactly one with $k-1$ columns plus one filling. We start with nothing placed, in state 0, and a finished tiling is one with n columns and state 0, since nothing may stick out past the board. Unlike letters, several fillings can join the same two states, so entries of A can exceed 1.

To list the fillings from a state without repeats, we always place next the tile covering the topmost uncovered cell of the column, trying every placement that fits and avoids covered cells.

12.5.2 Dominoes

Let f_n be the number of domino tilings of the $3 \times n$ board. Start with an empty column (state 0). Its top cell can be covered by a horizontal domino or by a vertical domino. Trying both, there are three fillings: three horizontal dominoes (outgoing state 7); a vertical domino on the top two cells and a horizontal one in the bottom row (state 4); and a horizontal domino in the top row with a vertical one below it (state 1).

Next take state 1, where the top cell is already covered: either a vertical domino fills the rest (state 0), or two horizontal dominoes do (state 2 + 4 = 6). Doing this for every state gives the list $\mathcal{R}(m)$ of outgoing states, shown in Figure 4:

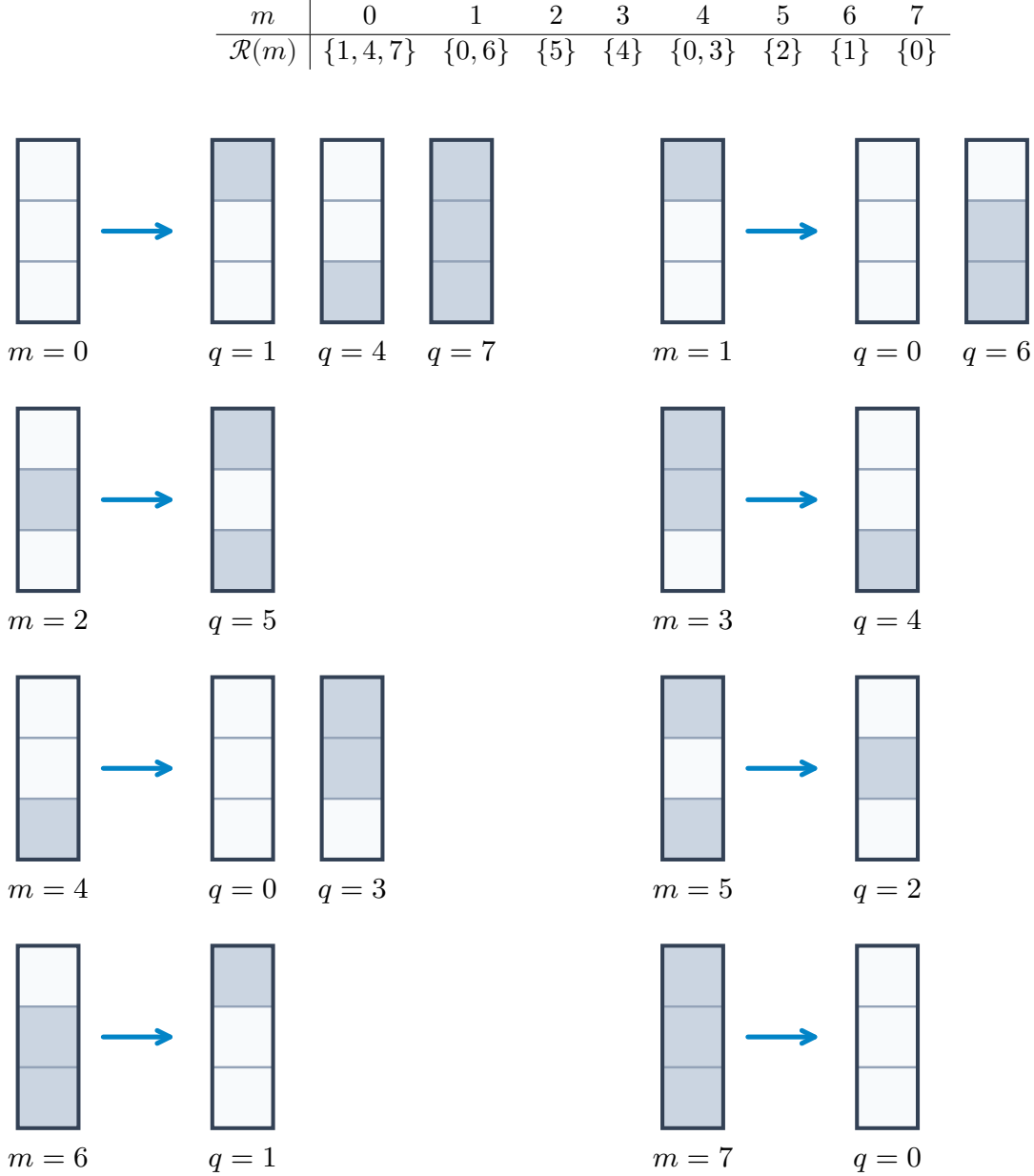


Figure 4: All domino fillings, one panel for each starting state m . Each panel shows the starting pattern on the left and every outgoing state q of a filling on the right.

Figure description: Eight panels of gray-cell occupancy patterns. Starting state zero has destinations one, four, and seven; one has zero and six; two has five; three has four; four has zero and three; five has two; six has one; and seven has zero. Arrows connect each starting pattern to its complete list of one-column destinations.

There is a quick way to check this table. The cells a domino filling covers in the next column come from horizontal dominoes, one per row of q , so q cannot use a row that is already covered in m . Every other uncovered cell of the current column needs a vertical domino, which works only if those cells are none, the top two, or the bottom two; and then in only one way. So each pair m, q has at most one filling, and column

m of the transition matrix D has a 1 in each row $q \in \mathcal{R}(m)$:

$$D = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad f_n = (D^n)_{0,0}.$$

```
D = matrix(ZZ, [[0, 1, 0, 0, 1, 0, 0, 1],
                [1, 0, 0, 0, 0, 0, 1, 0],
                [0, 0, 0, 0, 0, 1, 0, 0],
                [0, 0, 0, 0, 1, 0, 0, 0],
                [1, 0, 0, 1, 0, 0, 0, 0],
                [0, 0, 1, 0, 0, 0, 0, 0],
                [0, 1, 0, 0, 0, 0, 0, 0],
                [1, 0, 0, 0, 0, 0, 0, 0]])
show([(D^n)[0, 0] for n in range(11)])
show(D.charpoly('x').factor())
```

The counts are $(f_0, \dots, f_{10}) = (1, 0, 3, 0, 11, 0, 41, 0, 153, 0, 571)$; odd widths give 0 because $3n$ must be even. [Figure 5](#) and [Figure 6](#) show the tilings of widths 2 and 4.

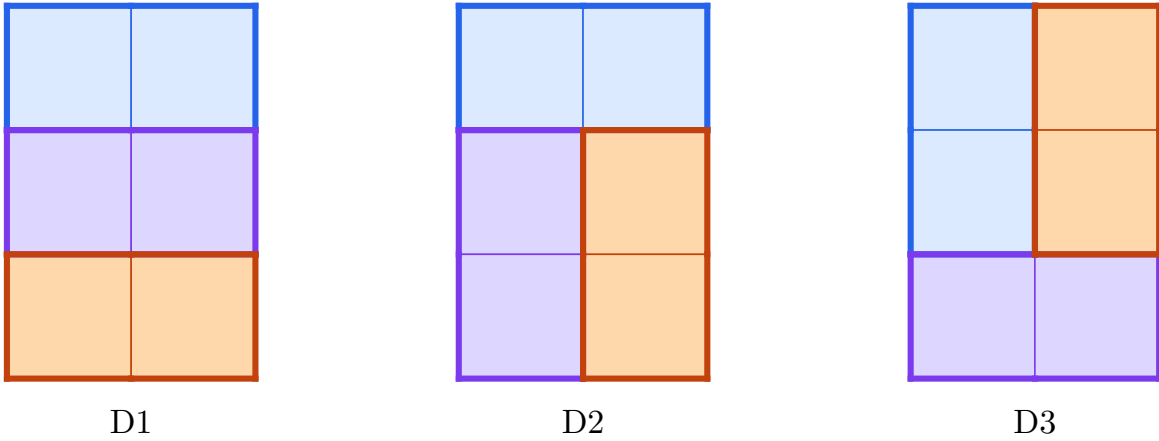


Figure 5: All three domino tilings of a 3×2 board: $f_2 = 3$.

Figure description: Three fully covered three-by-two boards. One has three horizontal dominoes; the other two have a horizontal domino in the top or bottom row and two vertical dominoes in the other rows.

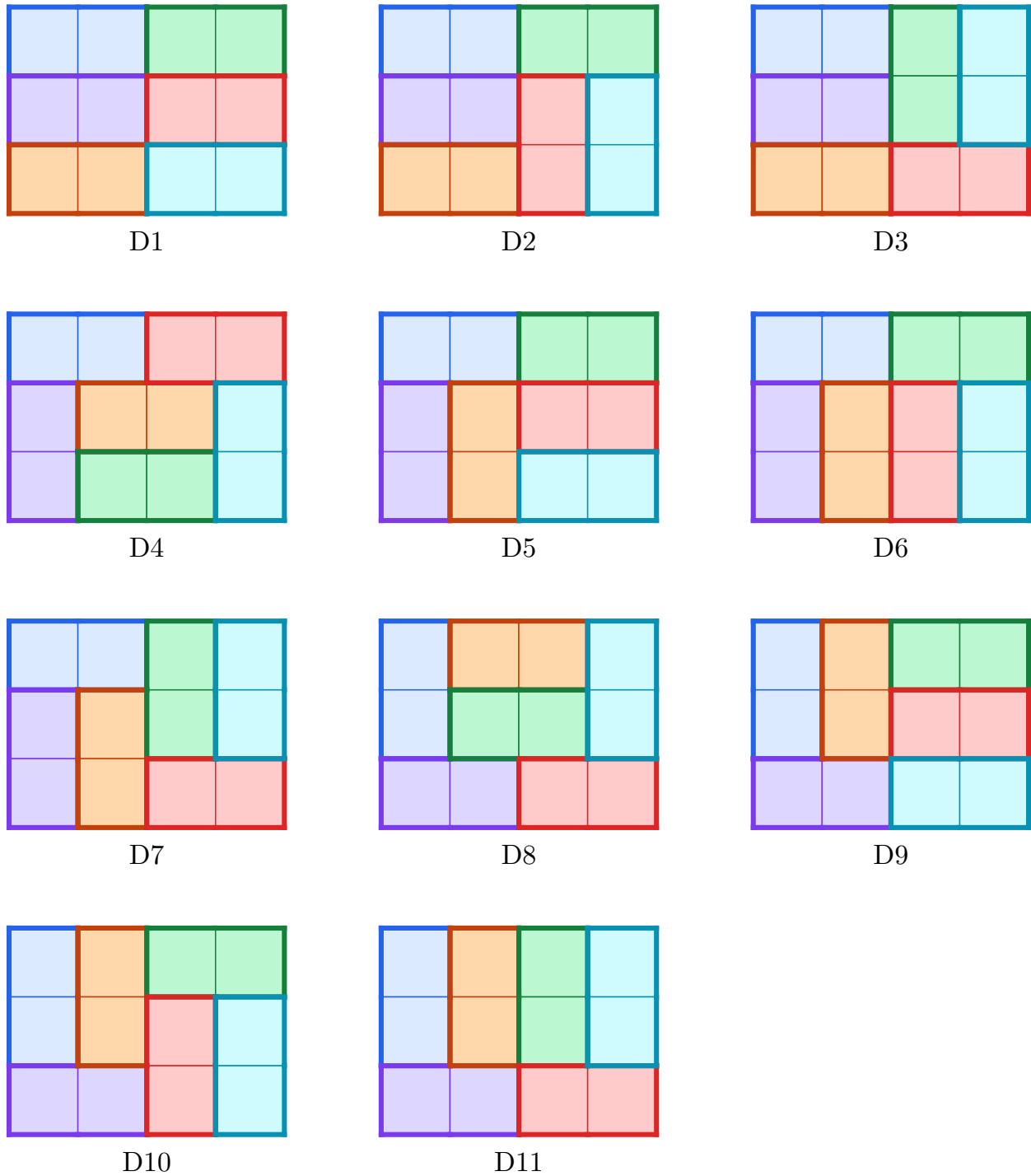


Figure 6: All eleven domino tilings of a 3×4 board: $f_4 = 11$.

Figure description: Eleven complete three-by-four boards labelled D1 through D11. Each has six dominoes distinguished by colored outlines; thin lines within dominoes show the original cell grid.

Sage factors $\chi_D(x) = (x-1)^2(x+1)^2(x^4 - 4x^2 + 1)$, so [Theorem 4](#) gives a recurrence of order 8. The last factor suggests something shorter, and indeed

$$\boxed{f_n = 4f_{n-2} - f_{n-4} \quad (n \geq 4),} \quad (f_0, f_1, f_2, f_3) = (1, 0, 3, 0).$$

Check: $4 \cdot 11 - 3 = 41$ and $4 \cdot 41 - 11 = 153$. This answers the question left open in [Lecture Note 11](#).

Why does the shorter recurrence hold? Write $X_k(m)$ for the number of partially built tilings with k finished columns in state m , so $f_k = X_k(0)$. Starting from state 0, the table shows that only states 0, 1, 3, 4, 6, 7 ever occur, and flipping a tiling upside down shows that states 1 and 4 have equal counts, as do 3 and 6. So rows 0, 1, 3, 7 of $X_k = DX_{k-1}$ become

$$X_k(0) = 2X_{k-1}(1) + X_{k-1}(7), \quad X_k(1) = X_{k-1}(0) + X_{k-1}(3), \quad X_k(3) = X_{k-1}(1), \quad X_k(7) = X_{k-1}(0).$$

Substituting gives $f_k = 3f_{k-2} + 2X_{k-2}(3)$ and $X_k(3) = f_{k-2} + X_{k-2}(3)$, and eliminating the $X(3)$ terms gives $f_k = 4f_{k-2} - f_{k-4}$.

12.5.3 Dominoes and angles

Let g_n count tilings by dominoes and angles. The same eight states work; only the fillings change. [Figure 7](#) shows the eleven fillings from state 0. Some outgoing states repeat: two fillings reach state 3, one using a top horizontal domino and an angle, the other an angle in the top row and a vertical domino below it. Both count, so the matrix entry in row 3, column 0 is 2.

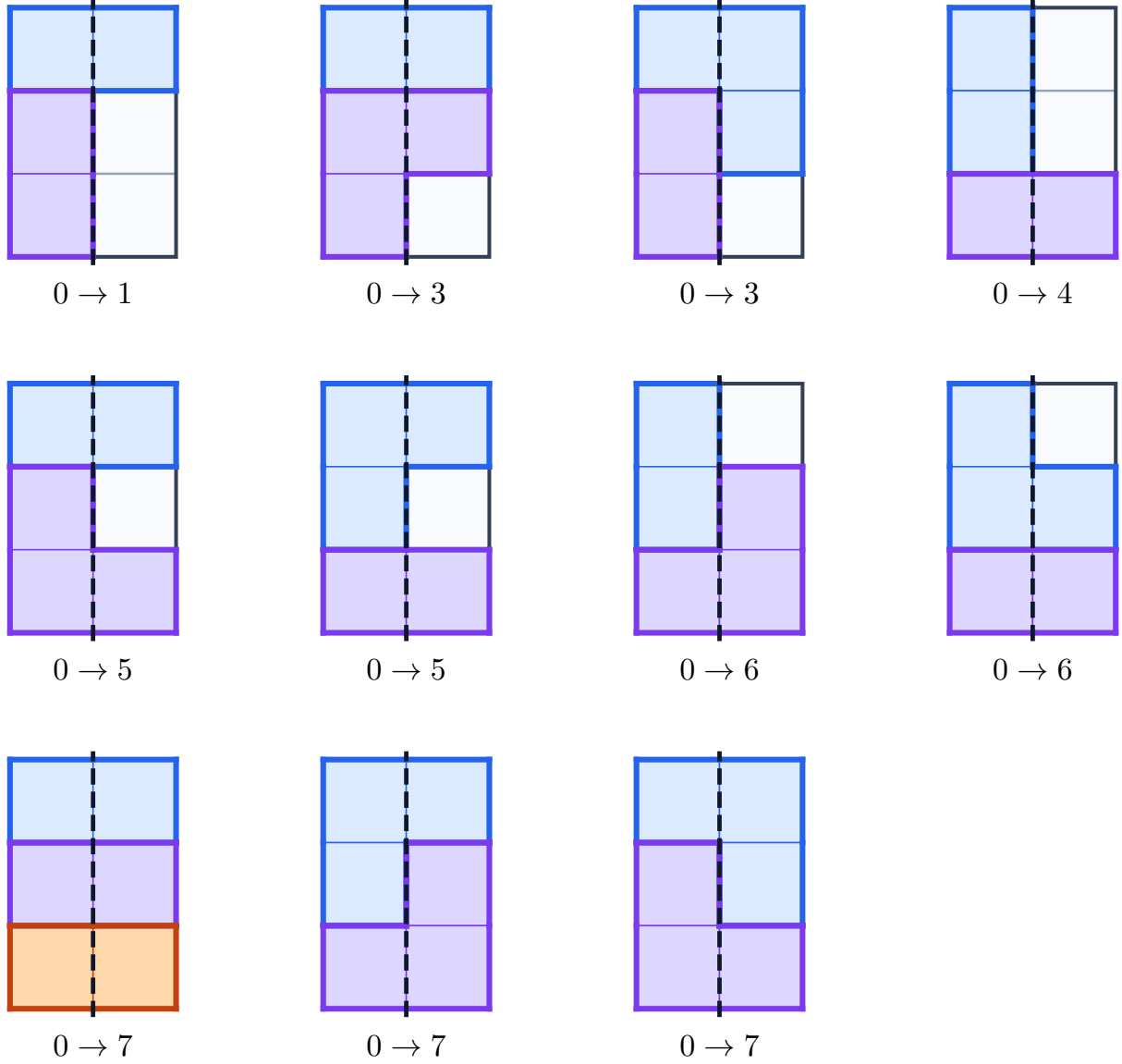


Figure 7: All eleven domino-and-angle fillings from state 0. Outgoing states that occur more than once give entries of E larger than 1.

Figure description: Eleven two-column panels show one filling to state one, two fillings to state three, one to state four, two to state five, two to state six, and three to state seven. Colored outlines identify each domino or angle and a dashed line separates the filled column from the outgoing occupied cells.

Listing the fillings from every state in the same way gives

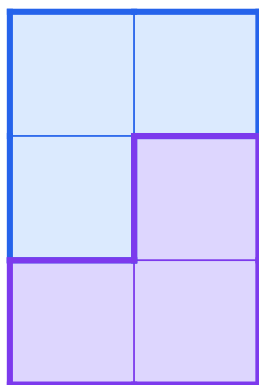
$$E = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 3 & 1 & 2 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}, \quad g_n = (E^n)_{0,0}.$$

```

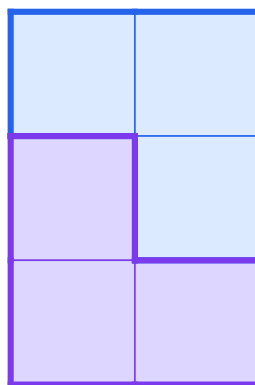
E = matrix(ZZ, [[0, 1, 0, 0, 1, 0, 0, 1],
                [1, 0, 0, 0, 1, 0, 1, 0],
                [0, 1, 0, 0, 1, 1, 0, 0],
                [2, 0, 0, 0, 1, 1, 1, 0],
                [1, 1, 0, 1, 0, 0, 0, 0],
                [2, 0, 1, 0, 0, 0, 0, 0],
                [2, 1, 0, 1, 0, 1, 0, 0],
                [3, 1, 2, 0, 1, 0, 0, 0]])
show([(E^n)[0, 0] for n in range(9)])
show(E.charpoly('x'))

```

The counts begin $(g_0, \dots, g_8) = (1, 0, 5, 8, 55, 140, 633, 1984, 7827)$. On the 3×2 board the three domino tilings are joined by two tilings with two angles (Figure 8). On the 3×3 board every tiling uses one angle and three dominoes (Figure 9), so an odd number of cells is no longer an obstacle.



A1



A2

Figure 8: The two 3×2 tilings by two angles. Together with the three domino tilings, they give $g_2 = 5$.
Figure description: Two three-by-two boards each tiled by two angles. The top-row angle extends into the middle row at the left in one board and at the right in the other; the bottom-row angle covers the complementary cells.

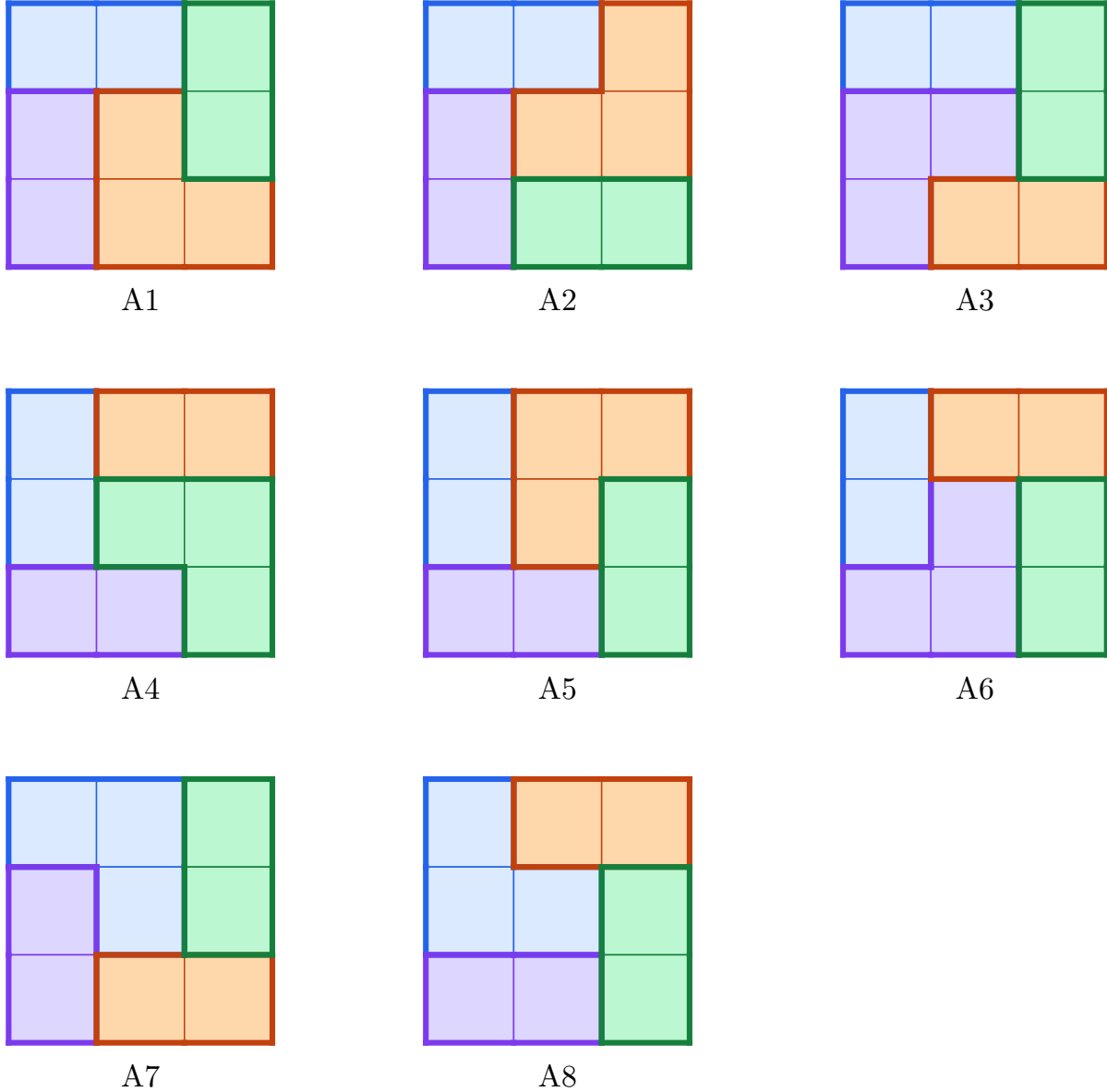


Figure 9: All eight domino-and-angle tilings of a 3×3 board: $g_3 = 8$. Each consists of one angle and three dominoes.

Figure description: Eight complete three-by-three boards labelled A1 through A8. Every panel contains one angle and three dominoes; thick colored outlines separate the four tiles.

Sage gives $\chi_E(x) = x^8 - 10x^6 - 8x^5 - 3x^4 - 24x^3 - 4x^2$. Reading off the coefficients as before gives

$$g_n = 10g_{n-2} + 8g_{n-3} + 3g_{n-4} + 24g_{n-5} + 4g_{n-6} \quad (n \geq 8).$$

The terms g_{n-7} and g_{n-8} are missing because χ_E has no x or constant term, so this is really a recurrence of order 6; but the theorem proves it only for $n \geq 8$. For $n = 6, 7$ we check it directly: $633 = 10 \cdot 55 + 8 \cdot 8 + 3 \cdot 5 + 4 \cdot 1 = g_6$ and $1984 = 10 \cdot 140 + 8 \cdot 55 + 3 \cdot 8 + 24 \cdot 5 = g_7$. So it holds for all $n \geq 6$.

12.6 Summary of the method

1. Build the objects one piece at a time: a letter, a step, or a column filling.

2. Choose states that record what decides the allowed next pieces: the longest suffix that is a proper prefix of a forbidden string, the last step, or the covered cells beyond the cut.
3. Form the transition matrix, whose entry in row q and column p counts the pieces leading from p to q , together with the initial vector and the coordinates to add at the end.
4. Use the characteristic polynomial to get a scalar recurrence, and check where it starts.

Problem 7 (Overlapping forbidden substrings). Let w_n be the number of strings of length n over $\{a, b, c, d\}$ avoiding $abca$, $bcab$, and $cabc$, as at the end of [Lecture Note 11](#). List the proper prefixes of the forbidden strings, build the machine as in [Section 12.1.3](#), write its transition matrix, and use Sage to compute $w_0, \dots, w_{10}$ and a recurrence for (w_n) .