The Syntax of Sudoku (Grids) – Roadmap

This is a preliminary draft / brainstorming
and will be updated step by step;
be patient!

Alex Pfaff
nonintersective@gmail.com

September 4, 2025

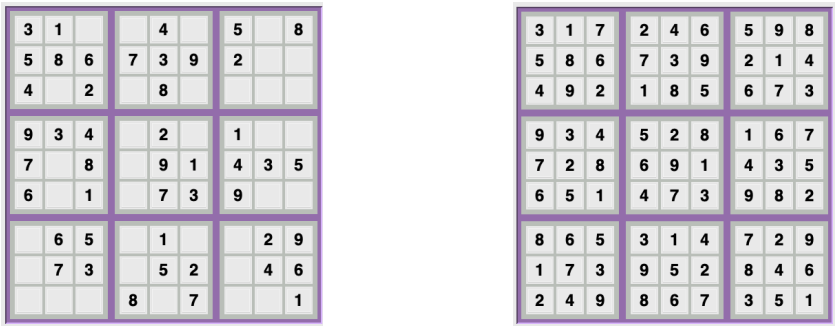
comments and feedback welcome!

1 Basics & Basic Terminology

A (classical) Sudoku is a 9×9 matrix or grid structure subdivided into nine 3×3 boxes. The resulting 81 cells (or *slots*) must be filled with the digits 1–9 in such a way that no digit repeats within any row, column, or 3×3 box. While *Sudoku Puzzles*—where some digits are given and the remaining cells are left blank—are the form most commonly encountered, in the following we will be concerned with fully filled *Sudoku Grids*.[1]

Individual cells can be identified by their $(\text{row}, \text{column})$ coordinates: for instance, $(1, 1)$ points to the top-left cell (value: 3), while $(4, 7)$ identifies the cell at the intersection of row 4 and column 7 (value: 1). Boxes are numbered from 1 to 9, also starting in the top-left corner and proceeding left to right, top to bottom. In addition, we will use the terms *boxrow* and *boxcolumn* to refer to sets of rows or columns spanning the width of a box.

(1) **Sudoku Puzzle** **Sudoku Grid**



(2) **Grid Sequence**
(abbreviated)

(3, 1, 7, 2, 4, 6, 5, 9, 8, 5, 8, 6, 7, 3, 9, 2, 1, 4, 4, 9, 2, . . .
. . . 4, 7, 2, 9, 1, 7, 3, 9, 5, 2, 8, 4, 6, 2, 4, 9, 8, 6, 7, 3, 5, 1)

An alternative way to represent a grid is as a *Grid Sequence* of length 81, obtained by parsing the grid row by row from top-left to bottom-right.[2] A grid sequence encodes exactly the same information as the two-dimensional grid, apart from the explicit geometry. We will occasionally refer to the translation from grid to sequence as *Linear Reduction*, and the reverse operation as *Geometric Expansion*.

[1] Procedurally, a grid can be seen as the solution to a given puzzle. Conceptually, however, the grid precedes the puzzle: a well-formed puzzle must admit exactly one unique solution, which is the grid itself.

[2] This choice of ordering is, of course, conventional, but a natural one: it mirrors the reading direction of most Latin-alphabet texts and coincides with the output of a NumPy instruction such as `grid.flatten()`.

2 Some Basic Numbers

1.9×10^{77}	9^{81}	**No rules** – $v \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ – $9 \times 9 = 81$ slots
5.3×10^{70}	$\frac{81!}{(9!)^9}$	**Cardinality Constraint** – $v \in \{1, 2, \ldots, 9\}^9$ – $9 \times 9 = 81$ slots
6.7×10^{21}	6670903752021072936960	**Valid Sudoku Grids** – $v \in \{1, 2, \ldots, 9\}^9$ – $9 \times 9 = 81$ slots distribution of v according to Sudoku rules
$6.7 \times 10^{21} \times \binom{81}{k}$		**Hypothetical Sudoku Puzzles with k blanks (upper bound)**

Table 1: Number of possible and valid Sudoku grids (and related counts).

2.1 Top-Down: Demarcations

Starting point: a 9×9 grid $\rightarrow$ 81 slots. We want to assign values $v = (v_1, v_2, \ldots, v_{81})$ to these slots, where $v \in \{1, 2, \ldots, 9\}$.

Case 1: No constraints. If there are no further restrictions (that is, each slot can be filled independently with any of the nine digits, and repetitions are unrestricted), we obtain

$$9^{81} \approx 1.9 \times 10^{77}$$

possible grid assignments.

Case 2: Cardinality constraint. Suppose instead we impose the requirement that each digit 1–9 occurs *exactly nine times* in the completed grid. In this case, the number of possible assignments is

$$\frac{81!}{(9!)^9} \approx 5.3 \times 10^{70}.$$

Case 3: Valid Sudoku grids. Finally, if we demand full compliance with Sudoku's structural rules (no repeats within any row, column, or 3×3 box), then the number of

valid grids reduces dramatically. Felgenhauer and Jarvis (2005) showed that there are exactly

$$6,670,903,752,021,072,936,960 \;\approx\; 6.7 \times 10^{21}$$

valid Sudoku grids.[3]

Case 4: Hypothetical Sudoku puzzles. On the basis of this result, one can estimate an upper bound on the number of *hypothetical* Sudoku puzzles by allowing some cells to remain blank. For k blanks, the number of possible such assignments is

$$\sum_{k=1}^{81} \left(6.7 \times 10^{21}\right) \times \binom{81}{k}.$$

Here k ranges over the number of blank slots.

(i) McGuire et al. (2014) proved that, in a classical Sudoku (with no additional constraints), the minimum number of given digits required for a puzzle to admit a unique solution is 17, i.e. $81 - 17 = 64$ blanks. Thus, the summation index should be reduced to $1 \leq k \leq 64$.

(ii) Even under this restriction, not every "blanked grid" yields a valid puzzle (with a unique solution). For example, if all nine instances of two digits are missing (say, no 3s and no 7s), the grid admits at least two completions. Hence, for $18 \leq k \leq 64$, the actual number of valid puzzles with a unique solution is *strictly less* than $6.7 \times 10^{21} \times \binom{81}{k}$. This is why we refer to these counts as *hypothetical Sudoku puzzles*.

See Table 1 for a numerical summary of these cases.

2.2 Bottom-up: Expansions

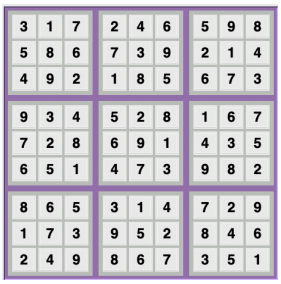
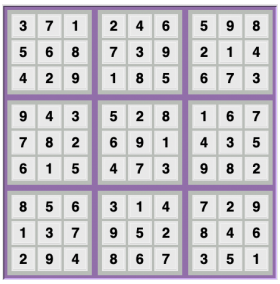
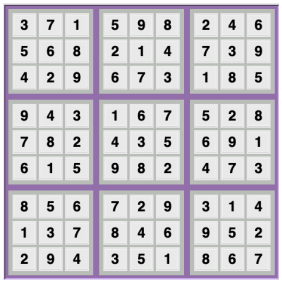
Starting from a valid grid, we can generate further valid grids by permuting its rows and columns in a constrained fashion:

Column operations. If we swap two columns *within the same boxcolumn*, we obtain another valid grid (see illustration below, left and middle). The same holds for swaps within each of the three boxcolumns. Furthermore, we may swap entire boxcolumns with one another (see below, middle and right).

[3] See also:

`https://en.wikipedia.org/wiki/Mathematics_of_Sudoku`

`http://sudopedia.enjoysudoku.com/Mathematics_of_Sudoku.html`

These permutations can be combined. Exhaustively applying all possible combinations yields

$$3!^4 = 1,296$$

valid column permutations.

Row operations. Exactly the same reasoning applies to rows (via boxrows). Hence, altogether we obtain

$$3!^4 \times 3!^4 = 3!^8 = 1,296 \times 1,296 = 1,679,616$$

valid **grid permutations**.

Rotation and reflection. Interestingly, rotation does not generate new grids beyond those already included in the above permutations, but reflection does.[4] Thus every grid in the permutation series can be mapped to its reflection, which is genuinely new. In short, one valid grid gives rise to

$$2 \times 1,679,616 = 3,359,232$$

valid grids through these geometric transformations.

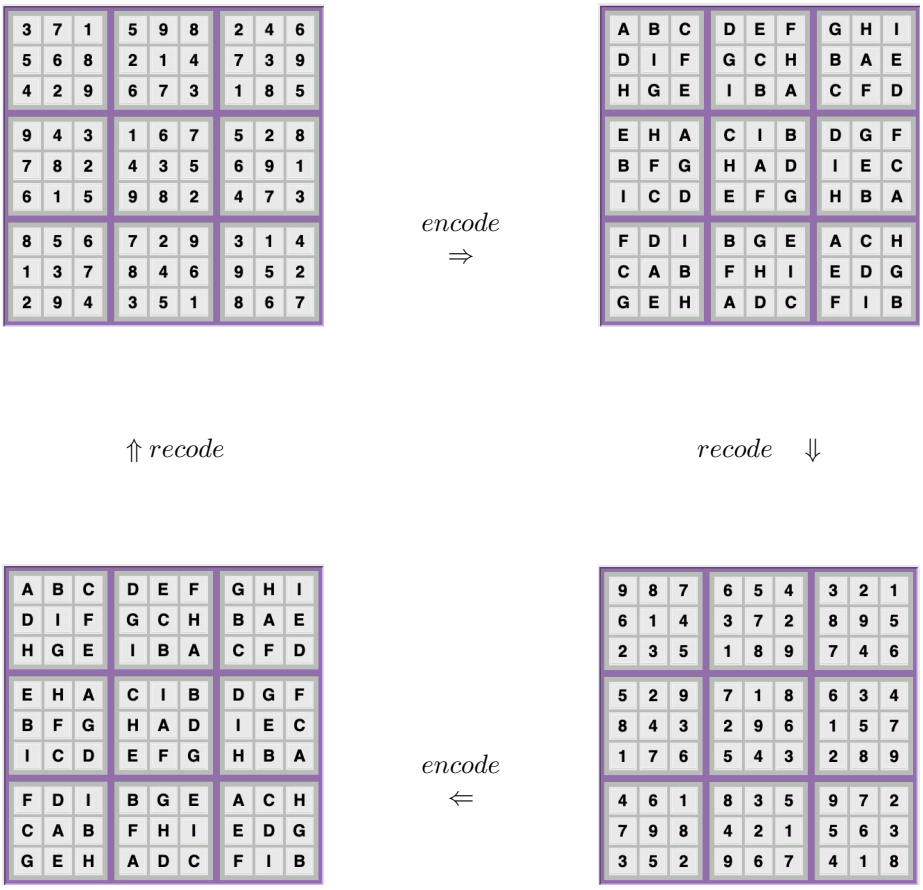
Value substitution and abc-grids. We now turn to substitution, which is easier to describe via an intermediate alphabetic representation: the *abc-grid*. To construct it, we use a fixed key:

$$\text{top\_row\_of\_grid} \;\Leftrightarrow\; (A, B, C, D, E, F, G, H, I)$$

That is, the nine digits appearing in the top row of a numerical grid (*int-grid*) are mapped, in order, to the letters A–I. This key is then applied consistently to the entire grid: each digit is replaced by its corresponding letter. Notice that the mapping works both ways: the fixed letter sequence $(A, \ldots, I)$ may be mapped back onto any valid digit sequence, and the abc-grid can thereby be recoded as an integer grid.

The letter sequence $(A, \ldots, I)$ is constant (immutable), so the encoding step is deterministic. Recoding, however, admits degrees of freedom: *any* valid Sudoku row can serve as the digit sequence aligned with $(A, \ldots, I)$.

[4]This is based on systematic observation rather than formal proof. Generating all grid permutations (1,679,616), then applying four rotations and a reflection to each produces $13,436,928 = 1,679,616 \times 4 \times 2$ grids. Taking the set of these shows only $3,359,232 = \frac{13,436,928}{4} = 1,679,616 \times 2$ distinct outcomes. In other words: Permutation × Rotation × Reflection = Permutation × Reflection.



Distributional equivalence. An abc-grid is distributionally equivalent to the int-grid from which it was created, and any recoding remains distributionally equivalent: the distribution of the value in cell 1 is identical whether it is labelled 3, A, or 9 (see illustration above). The same holds for every cell.

In other words, all int-grids obtainable from a given abc-grid by recoding are equivalent up to relabeling. Since the number of possible recoding keys is 9!, each abc-grid corresponds to

$$9! = 362,880$$

distributionally equivalent int-grids. The abc-grid can thus be seen as the *deep structure* (prototype) for a given distribution of digits, with its associated int-grids as *surface structures* (substitutions).

At face value, two int-grids such as the top-left and bottom-right examples above appear distinct (because the actual digits differ at given positions). At a deeper level,

however, they share the same distributional pattern: both correspond to the same abc-grid.

Vertical vs. Horizontal permutation series. Terminologically, we distinguish two expansion families:

- **Vertical Permutation Series:** grids obtained via row/column permutations (plus reflection).
- **Horizontal Permutation Series:** grids obtained via value substitution, i.e. recoding from a given abc-grid.

Grids in the vertical series are distinct in the sense that structural positions (rows, columns) are permuted. This produces

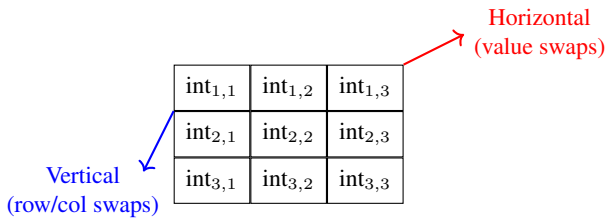
$$3,359,232$$

distinct abc-grids. Each of these can in turn be recoded in $362,880$ ways.

Therefore, one single valid grid ultimately gives rise to

$$3,359,232 \times 362,880 = 1,218,998,108,160 \approx 1.2 \times 10^{12}$$

(superficially) distinct valid grids.



	123456789	...	**987654321**
abc_grid_1	$\text{int\_grid}_{1,1}$	...	$\text{int\_grid}_{1,362880}$
abc_grid_2	$\text{int\_grid}_{2,1}$	...	$\text{int\_grid}_{2,362880}$
...	...	...	...
abc_grid_1679616	$\text{int\_grid}_{1679616,1}$	...	$\text{int\_grid}_{1679616,362880}$

Table 2: Terminology: *Vertical* vs. *Horizontal Permutation Series*. Rows correspond to abc-grids generated by grid permutations; columns correspond to integer keys for recoding.

3 Syntax of Sudoku Grids

We now turn to the central question of this work: understanding the *syntax of Sudoku grids*. Informally, the syntax of Sudoku refers to the set of constraints and regularities that distinguish valid Sudoku grids from arbitrary 9×9 arrangements of digits 1-9. Formally, we may treat this problem within the framework of *formal languages*.

Sudoku as a language. Let us define the **language** $\mathbb{G}$ as the set of all valid Sudoku grids. Each element of $\mathbb{G}$ can be represented either as a 9×9 matrix of digits (the *grid representation*) or as a linear sequence of 81 digits (the *grid sequence*).

We are interested in two canonical tasks:

1. **Recognition (classification):** Given a candidate grid, determine whether it belongs to $\mathbb{G}$.
2. **Generation (production):** Produce elements of $\mathbb{G}$, ideally enumerating all valid grids or sampling from them uniformly.

Linearized sequences and hidden regularity. Linearizing a grid yields a sequence of length 81, e.g. by row-wise concatenation. This sequence obeys deterministic constraints derived from the Sudoku rules, but the constraints are non-local: the same digit can appear multiple times in the sequence if it is in different rows or boxes. A natural question arises: *can we classify or generate valid sequences without reconstructing the 2D grid geometry explicitly?*

In other words, we seek an algorithmic or procedural characterization of $\mathbb{G}$ that might uncover *hidden regularities* in the sequences themselves. A speculative illustration: could there exist a series of "magic matrices" $Mat_1, \ldots, Mat_n$ such that for any grid G, we have

$$G \times Mat_1 \times \cdots \times Mat_n = I \quad \Leftrightarrow \quad G \in \mathbb{G},$$

where I denotes an identity-like structure? While unlikely in practice, this captures the type of abstract thinking we aim to explore: identifying intrinsic structural properties that define valid grids.

Grammars and the Chomsky hierarchy. Viewed as a formal language over the alphabet $\Sigma = \{1, \ldots, 9\}$, the language $\mathbb{G}$ is finite but extremely large ($\sim 6.7 \times 10^{21}$ elements). In principle, any finite set is trivially *regular*, but constructing a regular grammar that generates all and only valid Sudoku grids is infeasible. Context-free grammars similarly fail to capture the necessary inter-row, inter-column, and inter-box dependencies.

Consequently, a context-sensitive (type 1) or even unrestricted (type 0) grammar may be able to generate $\mathbb{G}$, but such grammars are expected to be extremely complex. This motivates our search for alternative, potentially simpler procedural representations of the syntax.

Representations and perspectives. The distinction between *grid* and *sequence* representations highlights different perspectives on the problem:

- Grid representation emphasizes the 2D geometry of rows, columns, and boxes.
- Sequence representation emphasizes the linear ordering of digits and the possibility of discovering non-obvious regularities in sequences.

Similarly, the dual tasks of recognition and generation suggest two complementary research directions:

1. Can we construct an algorithm that decides membership in $\mathbb{G}$ without explicitly checking Sudoku constraints?
2. Can we construct an algorithm that produces all and only valid elements of $\mathbb{G}$ efficiently, possibly revealing hidden structural patterns in the process?

Summary. In short, the *syntax of Sudoku* refers to the abstract, structural properties of grids that define validity. The overarching question we pose is:

Is there an algorithmic or procedural characterization of $\mathbb{G}$ that generates or recognizes all and only valid Sudoku grids, ideally simpler than the explicit Sudoku rules?

The remainder of this work will explore different approaches to this problem, including both grid-based and sequence-based representations, as well as connections to combinatorial and neural methods.

4 Overflow (Modulo) Sudoku Grids

The classical Sudoku geometry lives on a finite 9×9 board: there are 9 rows, 9 columns, and 9 boxes, each box being 3×3. Navigation is ordinarily *bounded* by the edges of this 9×9 structure. By contrast, in this subsection we introduce a *modular wrap-around* ("overflow") perspective on the same grid. Intuitively, one can move along rows or columns *indefinitely* without ever leaving the row or column: stepping "right" from column 9 brings you to column 1 (same row), and stepping "up" from row 1 brings you to row 9 (same column). This view is purely geometric/coordinate-theoretic unless explicitly stated otherwise; it does not, by itself, alter which grids are valid under the classical rules.

In order to describe the wrap-around mechanics precisely, let us introduce a compact notation. We denote the position of a cell in the grid by a pair of coordinates (r, c), where r is the row index and c the column index, both ranging from 1 to 9. Movements in the grid can then be formalized as operators acting on such coordinates.

We use the modular operators $\oplus_9$ and $\ominus_9$ to mean "addition modulo 9" and "subtraction modulo 9," respectively. Thus, for example, $c \oplus_9 1$ means "move one step to the right, wrapping around from column 9 back to column 1." Similarly, $r \ominus_9 1$ means "move one step up, re-entering the grid at row 9 if necessary."

Formally, the four fundamental moves are defined as follows:

$$\begin{aligned} \mathrm{Right}(r,c) &= (r,\ c \oplus_9 1), & \mathrm{Left}(r,c) &= (r,\ c \ominus_9 1),\\ \mathrm{Down}(r,c) &= (r \oplus_9 1,\ c), & \mathrm{Up}(r,c) &= (r \ominus_9 1,\ c). \end{aligned}$$

In words: starting from any cell (r, c), the operator Right takes us to the next column, wrapping around from column 9 to column 1; Left moves in the opposite direction; Down advances one row downward, with wrap-around from row 9 to row 1; and Up moves upward, wrapping from row 1 to row 9.

Index sets and 1-based modular arithmetic. Let $R = C = \{1, \dots, 9\}$ denote the row and column index sets, and let $P = R \times C$ be the set of cell positions. We write

$$a \oplus_9 k \stackrel{\text{def}}{=} 1 + \big((a - 1 + k) \bmod 9\big), \qquad a \ominus_9 k \stackrel{\text{def}}{=} 1 + \big((a - 1 - k) \bmod 9\big),$$

for $a \in \{1, \dots, 9\}$ and $k \in \mathbb{Z}$. Thus $9 \oplus_9 1 = 1$ and $1 \ominus_9 1 = 9$. We extend $\oplus_9, \ominus_9$ componentwise to positions $(r, c) \in P$:

$$(r,c) \oplus (\Delta r, \Delta c) \ = \ \big(r \oplus_9 \Delta r,\ c \oplus_9 \Delta c\big), \quad (r,c) \ominus (\Delta r, \Delta c) \ = \ \big(r \ominus_9 \Delta r,\ c \ominus_9 \Delta c\big).$$

Cyclic moves (wrap-around navigation). Define the four elementary moves on P:

$$\begin{aligned} \mathrm{Right}(r,c) &= (r,\ c \oplus_9 1), & \mathrm{Left}(r,c) &= (r,\ c \ominus_9 1),\\ \mathrm{Down}(r,c) &= (r \oplus_9 1,\ c), & \mathrm{Up}(r,c) &= (r \ominus_9 1,\ c). \end{aligned}$$

By iteration, $(r, c) \mapsto (r,\ c \oplus_9 k)$ cycles through the row, and $(r, c) \mapsto (r \oplus_9 k,\ c)$ cycles through the column. In particular, one can "move along" rows and columns indefinitely. This induces *cyclic neighborhoods* (e.g. a radius-t neighborhood along a row using $\oplus_9, \ominus_9$), which are useful for periodic filtering or convolutional views of the grid.

Linear reduction and circular shift. Under linear reduction (row-major flattening), the coordinate map

$$\ell : P \to \{1, \dots, 81\}, \qquad \ell(r,c) = 9\,(r-1) + c,$$

is a bijection with inverse $\ell^{-1}(i) \ = \ \big(1 + \lfloor (i-1)/9 \rfloor,\ 1 + ((i-1) \bmod 9)\big)$. On $\{1, \dots, 81\}$ we define the 1-based modulo operation

$$i \oplus_{81} k \stackrel{\text{def}}{=} 1 + \big((i - 1 + k) \bmod 81\big).$$

Thus a *circular shift* σ_k of a grid sequence $(x_1, \dots, x_{81})$ is given by $(x_{\,1 \oplus_{81} k}, \dots, x_{\,81 \oplus_{81} k})$. Note that $\oplus_{81}$ produces wrap-around at the *end of the sequence* (position 81 returns to 1), while $\oplus_9$ produces wrap-around within *rows or columns*. The two are related by ℓ but capture different navigation regimes (global vs. per-axis cyclicity).

Why this perspective matters for syntax. From a syntactic viewpoint (Section 3), overflow indexing provides:

- A *coordinate system* that exposes periodic structure: rows and columns form cycles of length 9; sequences form a cycle of length 81.
- A principled way to define *circular features* (e.g. cyclic n-grams along a row/column, or periodic convolutional kernels) without edge effects.
- A clean algebra for *equivalence by rotation of indices* (e.g. comparing patterns up to cyclic shifts), potentially revealing hidden regularities in $\mathbb{G}$ without reconstructing geometry from scratch.

Crucially, this is a representational device: it changes how we navigate and measure regularities, not what counts as a valid grid (unless we explicitly adopt toroidal constraints as a separate variant).

A minimal schematic. The wrap-around behavior can be visualized as follows (right edge connects to left edge in each row; top edge connects to bottom edge in each column).

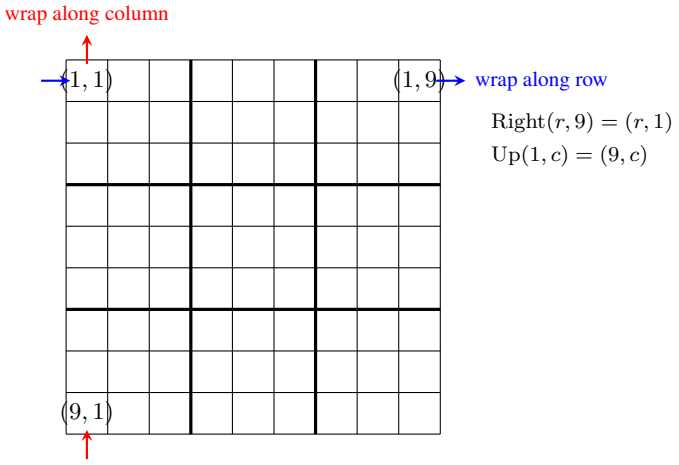


Figure 1: Overflow (modular) navigation on a 9×9 grid: rows and columns are cyclic under $\oplus_9$. Boxes remain classical unless stated otherwise.

Two cautions.

1. Overflow indexing is a *navigation model*. Unless a toroidal constraint system is explicitly introduced, wrap-around does not license cross-boundary equality/inequality constraints for Sudoku validity.

2. Cyclic shifts of indices can expose periodic patterns, but they need not preserve validity under the classical rules. They are tools for *analysis*, not automatic generators of legal transformations (contrast Section 2.2 on legal grid permutations).

In summary, the overflow/modulo view equips us with a compact algebra for periodic navigation in both the matrix and sequence representations (via $\oplus_9$ and $\oplus_{81}$). This will be useful when formulating sequence-based regularities, circular features, and convolution-like procedures that probe the "syntax" of Sudoku without reconstructing geometry at every step.

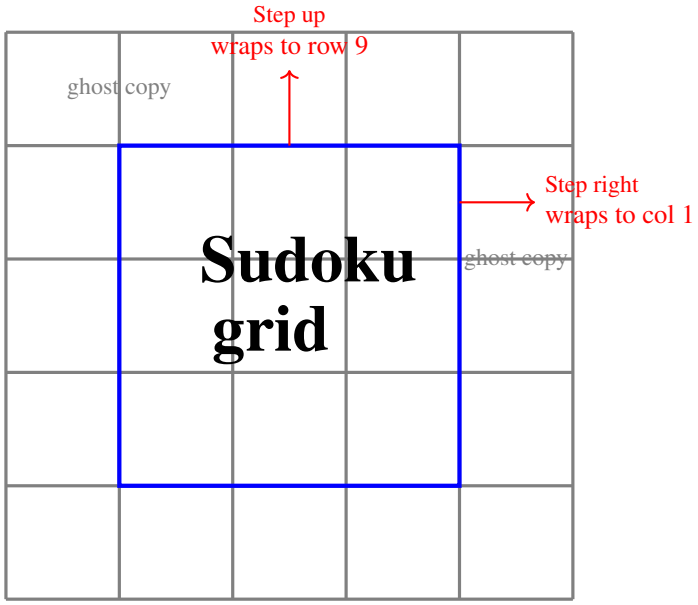


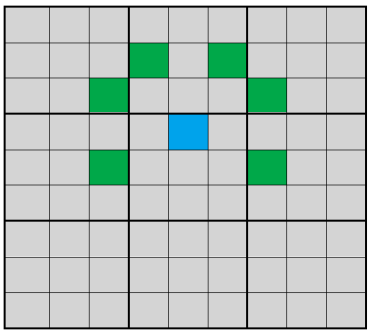
Figure 2: Illustration of modular overflow on a Sudoku grid. The central 9×9 is the actual grid; the surrounding "ghost copies" show what wrap-around moves *look like* when stepping beyond the boundary. In reality, overflow simply reenters the same grid.

4.1 Knight's Move Constraint

Beyond the classical constraints of rows, columns, and boxes, a number of Sudoku variants introduce an additional restriction inspired by chess: the *Knight's Move constraint*. This rule is modeled on the legal movements of a knight in chess, namely two steps in one of the cardinal directions followed by one step orthogonally. Formally, two cells are said to be related by a knight's move if their coordinates differ by $(\pm 2, \pm 1)$ or $(\pm 1, \pm 2)$. The Knight's Move constraint then requires that no two such cells may contain the same digit.

Figure 3 illustrates the classical situation: the blue cell is related by a knight's move to each of the surrounding green cells.

(3)

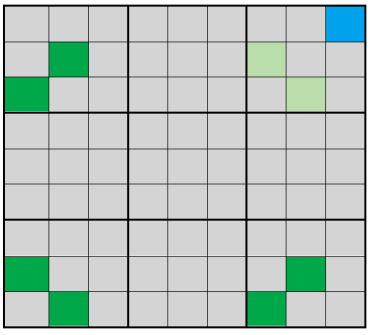


In the standard setting, however, the effective range of knight moves is curtailed by the boundaries of the grid. Just as a knight on the edge of a chessboard has fewer legal moves, so too does a Sudoku cell at the boundary: some of its potential knight's-move neighbors simply lie outside the 9×9 structure.

To extend this idea, we may introduce the *Modulo Knight*. This is a knight operating in a wrap-around, or *overflow*, Sudoku geometry of the type discussed earlier. In this toroidal environment, grid boundaries no longer block movement. Instead, moving beyond an edge simply re-enters the grid from the opposite side. Accordingly, a modular knight enjoys the full complement of eight possible moves, even when starting from a boundary or corner position.

Figure 6 visualizes this extension. The blue cell retains its classical knight-move neighbors (shown in light green), but now also acquires additional neighbors across the boundary (shown in dark green). The modular knight's constraint thus enlarges the web of disallowed equalities across the grid, making the puzzle structure both richer and more challenging.

(4)



4.2 Intermezzo: The travels of a modular knight

The *Knight's Tour* is a classical combinatorial puzzle: can a chess knight move across a board in such a way that every square is visited exactly once, without repetition? This question has fascinated mathematicians for centuries, and countless variations have been studied. Among these variations, one can imagine placing the knight not on an 8×8 chessboard, but on a 9×9 Sudoku grid.

For the sake of exploration—and perhaps a little mischief—let us propose a new constraint for this journey:

> Traverse a Sudoku grid using knight's moves such that
>
> - every cell is visited exactly once, *and*
> - the start and landing positions of every move belong to **different** 3×3 **boxes**.

At first glance, the task looks forbidding. In the classical setting, a knight starting from one of the four corner positions $(1,1), (1,9), (9,1), (9,9)$ cannot even make the first move without violating the box condition. The geometry of the Sudoku grid constrains the knight so tightly that many starting points appear hopeless. (We have not tested whether / how many other journeys are possible.)

Enter the *modular knight*. By adopting the wrap-around geometry described earlier—where rows and columns cycle back on themselves—the knight is liberated from the edges of the grid. Moving "off the board" simply means reappearing on the opposite side, still within the same row or column. Under these modular rules, the knight's horizon expands dramatically: from *every* starting position on the Sudoku grid, the modular knight can, in principle, attempt the full traversal while respecting both the uniqueness condition and the box constraint. (5) and (6) illustrate two journeys, from a corner and the center position (start position marked blue, end position marked green).

(5)

50	13	43	6	18	38	20	36	68
2	63	4	22	44	24	46	51	48
73	30	7	76	9	52	39	56	26
57	3	74	31	23	77	25	47	40
27	72	29	8	75	10	53	78	55
41	60	32	71	16	81	66	34	58
33	28	15	61	11	70	79	54	58
59	42	12	17	80	19	37	67	35
64	5	62	14	21	45	69	49	1

(6)

15	66	57	28	9	26	41	22	68
60	46	30	39	51	4	13	77	44
72	53	48	58	31	42	50	61	16
47	59	38	52	49	12	5	43	76
17	71	54	34	1	32	62	19	73
75	35	80	37	63	6	11	24	70
79	55	64	7	33	2	20	74	18
67	81	36	56	27	10	25	69	23
45	29	8	65	40	21	3	14	78

The contrast is striking: what is literally impossible in the classical setting suddenly becomes feasible once the grid is reimagined as a toroidal world. The modular knight is no longer trapped by corners or edges—his travels become truly unbounded. Moreover, this tour is possible from all 81 positions; for illustration, run `PsQ_ModuloKnight.py`, which displays an iteration over all coordinates and the respective tour across the grid:

```
Start coordinate: ((1, 1))
Tour found!
[[ 1 54 39 32 14 47 68  8 56]
 [ 9 63 11 16 69 18 78 48 65]
 [57 23 33 40 28 67 35 61  2]
 [62 10 42 24 34 70 19 66 36]
 [ 3 51 22 29 41 27 60 71 58]
 [37 30 25 43 81 20 73  6 75]
 [50  4 52 21 26 44 76 59 72]
 [55 38 31 13 46 80  7 74  5]
 [64 12 15 53 17 77 45 79 49]]

Press <Enter> to continue

. . .

. . .

Start coordinate: ((5, 9))
Tour found!
[[66 17 54  8 24 34 39 64 22]
```

```
 [ 4 68  6 46 28 56 30 37 32]
 [21 47  9 69 11 38 35 57 80]
 [58  5 45 27 49 70 79 31 36]
 [61 20 48 10 78 12 50 81  1]
 [52 15 26 44 41 76 71 74 59]
 [ 2 60 19 77 13 73 42 51 62]
 [16 53 14 25 43 40 75 72 65]
 [63  7 67 18 55 29 23 33  3]]

Press <Enter> to continue

. . .
```

As an aside, it is possible to test the variant without the *box-jump constraint*:

```
>>> runner = TourRunner(size=9, box_jump_required=False)
>>> runner.run_tour(start_pos=(2, 2)))
    True
>>> runner.show_visualization()

    [[54  9 45 81 36 72 27 63 18]
     [32 68 23 59 14 50  5 41 77]
     [10 46  1 37 73 28 64 19 55]
     [78 33 69 24 60 15 51  6 42]
     [56 11 47  2 38 74 29 65 20]
     [43 79 34 70 25 61 16 52  7]
     [21 57 12 48  3 39 75 30 66]
     [ 8 44 80 35 71 26 62 17 53]
     [67 22 58 13 49  4 40 76 31]]
```

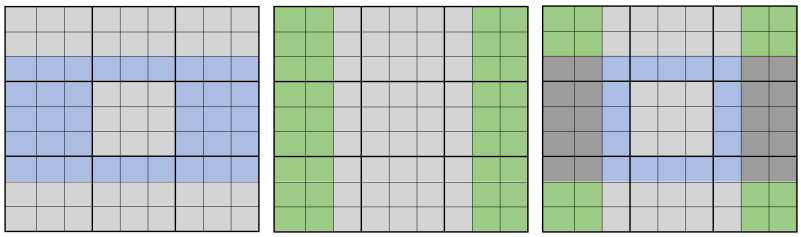
NB: the coordinate argument `start_pos` is zero-index-based, thus (2, 2) actually means row 3, column 3.

4.3 Moving the Phistomefel Ring

In this section, we take a closer look at a generalization of the so-called *Phistomefel Ring*.[5, 6] We begin with a valid Sudoku solution grid. Now, color rows 3 and 7 together with boxes 4 and 6 in blue, as in (7, left). The specific arrangement of digits does not matter; what is important is that the blue region contains exactly four copies of the digits 1–9, which we denote by the multiset B. Similarly, if we color columns 1, 2, 8, and 9 in green, we again obtain a region containing exactly four copies of the digits 1–9, which we call G, see (7, middle).

If we now overlay these two regions on the same grid, they intersect in a set of cells colored dark gray, denoted S, see (7, right).

(7)



Even though we do not know the *exact* digits in the gray intersection S, we can reason about the relationship between the sets. Every digit x in S belongs simultaneously to B and to G, so removing S from both regions leaves

$$B \setminus S \;=\; G \setminus S.$$

Thus, the digits in the remaining blue cells (the square "ring" surrounding the central box) are identical to the digits in the remaining green cells (the four 2×2 corner regions).

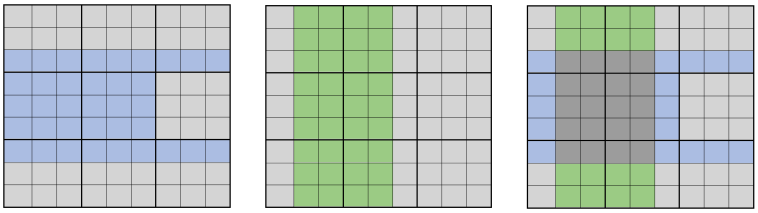
The blue "ring" around the central box has become known as the *Phistomefel Ring*, and the equivalence described above is often referred to as the *Phistomefel Theorem*.

Once this core correspondence is understood, it can be generalized. Phistomefel himself suggested several further equivalence relations, some of which we illustrate below (with slight adaptations). In each case, we follow the same procedure: 1. Color two boxes and two rows in blue (left), 2. Color four rows in green (middle), 3. Subtract the intersection (dark gray), 4. Conclude that the digits in the remaining blue cells are identical to those in the remaining green cells (right).

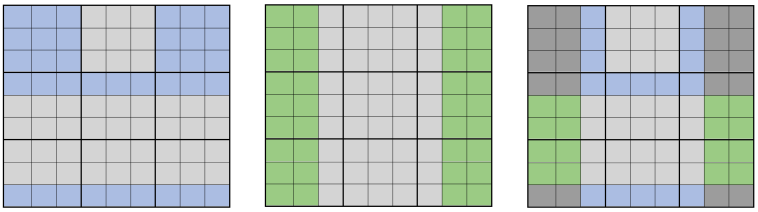
[5] To my knowledge, this was first discussed by the puzzle setter Phistomefel, after whom the phenomenon is named; see: `https://forum.logic-masters.de/showthread.php?tid=1811`.

[6] Our presentation here follows the exposition given in `https://www.youtube.com/watch?v=pezlnN4X52g`

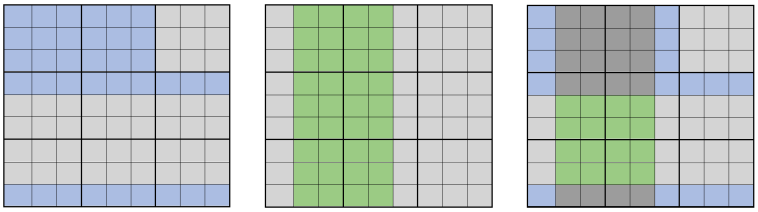
(8)



(9)



(10)



Moreover, under modular wrap-around, we see that each blue region in the right-hand panels of (8)–(10) is, in effect, *the Phistomefel Ring shifted across the grid by whole boxes*. The most striking case occurs in (10), where the "moved" ring is broken into four separate segments, yet together they are equivalent to the original ring.

4.4 CNN: modular padding

. . . .

References

Felgenhauer, Bertram, and Frazer Jarvis. 2005. Enumerating possible sudoku grids.

McGuire, Gary, Bastian Tugemann, and Gilles Civario. 2014. There is no 16-clue Sudoku: Solving the Sudoku minimum number of clues problem via hitting set enumeration. *Experimental Mathematics* 23 (2): 190–217.