Covering arrays: tests, tables, and room for one more column

READING LEVEL EVERYONE · EXPERTS

Checked by two separately written checkers: all 29 strength-four arrays pass the full strength-four check; at each of the 29 widths the array supplied is below the upper bound in the pinned comparison table and below the smallest size in Smolen's living table at commit 0fc74c0. Further material with its own statuses in the register of results: a theory of adding one column (Lean-proved, written and computed results), 48 Lean theorems built and re-checked by a second kernel checker, further arrays of strength 2 and small exact cases that pass the definition-based check, and a ledger of failed items

proved in writing 68 (61 written proofs on request) (of these, by their own register label: unconditional bound 2, special cases 1) · Lean-checked 60 (1 with files on request) · computed 26 (20 with files on request) · open 4 · conditional 2 (2 with files on request)

CHECK IT YOURSELF

The check guide for this book · How to check a result yourself

---

What you can rerun: The 29 strength-four arrays, the further arrays, the DRAT case, the hub-row construction and the 48-theorem Lean package are from-files. The comparison table, the hardness and rules Lean folders, the written proofs for chapters 6 to 13, and the solver outputs and search logs are files-not-online (available on request). Each result page gives its own rerun value and 5-minute check.

What a match shows: a match tells you that the published files really produce the published numbers. It doesn't tell you that the question was the right one to ask, or that a written argument around the numbers is complete. Each recipe says plainly what its steps show and what they don't. For a result whose evidence is logs (logs-only), a match checks the bookkeeping of the logs, not the search that wrote them.

Cite: Sonny (2026). Covering arrays: tests, tables, and room for one more column (preliminary). Go check yourself. https://gocheckyourself.org/read/covering-arrays/everyone/

1. Start here
2. Covering arrays, the short version
3. Covering arrays in plain words
4. The comparison table
5. How the check works
6. Room for one more column?
7. Strength two: one-column extension and a famous hitting-set puzzle
8. Strength three: pair classes, and when a table has room
9. The eighteen walls
10. Strength three is NP-complete, and so is every higher fixed strength
11. Strength four: rules, growth and what limits them
12. The 346th column
13. The Mirror Route and the Ladder
14. Other kinds of tables: three symbols
15. More arrays: small exact cases
16. The strength-2 collection: a summary
17. The ledger: what did not hold, and what is still open
18. The 48 Lean theorems: plain statements
19. Register of results
20. Theorems in this book
21. Check it yourself: covering arrays, the one-column extension theory, and the Lean files
22. Files and fingerprints
23. Trust and verification
24. Lean sources: how to check them
25. People and works
26. Files and rights

1 Start here

Sonny

What this book shows

- My result. I found a binary (1-bar,2)-locating array with 13 rows and 13 columns. The tables we found in the literature we checked have 14 rows. A locating array is a test table in which the failing tests tell you which pair of switch settings is faulty. Two separately written checkers of ours

confirm the table, and recipe 11 of the check guide lets you run the check yourself.

- Why it matters. Each row is one test run, so fewer rows mean a cheaper test plan that still pins down a faulty pair. The literature check is incomplete: one paper (Shi, Fu, Wang and Yan, 2021) is unread, so a table that small may already exist there. A search for a 12-row table with 13 columns found none: one solver answered UNSAT (a SAT solver tries to settle a yes-or-no logic puzzle; UNSAT means the puzzle has no solution) on 601,831 distinct cubes (slices of the search) and SAT on none; 552,691 of them are the terminal cubes that close the search tree, and the other 49,140 lie below a cube already answered UNSAT. That 13 rows is the minimum for 13 columns is CONJECTURED: one solver found no 12-row array in a SAT encoding of the problem; a second solver has re-checked a sample of the search, not the whole search, and the search is not part of this book.

- Also in this book. 29 binary covering arrays of strength four (test tables in which every choice of four switches shows all 16 on/off patterns), built from Paley's 1933 construction (a way of building tables from quadratic residues, the squares left when you divide by a fixed number) and free to share under CC BY 4.0 (a licence that lets anyone share and build on the work with credit), each below the upper bound in one pinned snapshot of Colbourn's tables and below the smallest size in Smolen's living table at its width (the smallest possible size is not known), and a theory of adding one more column to a finished table: NP-complete (among the hardest problems whose answers can be checked quickly; no quick method is known for them) from strength 3 up.

- Formalized and certified. 48 Lean theorems (Lean is a program that checks every step of a proof against fixed rules), re-checked by a second kernel checker (a separately written judge that checks the finished proof again); all 29 arrays checked on every choice of four columns by two separate programs.

- Every theorem in this book is listed in the theorem library.

The question

Suppose a machine has 200 switches, each on or off, and you want to test it. Trying every setting is impossible, so you aim for less: whichever four switches you pick, every one of their 16 on/off patterns should show up in at least one test. A table of 0s and 1s that does this is called a binary covering array of strength four. Each row is one test, each column is one switch, and the number of rows is the cost.

How few rows are enough? For the widths in this book (200 to 206, 223 and 325 to 345 columns), nobody knows the smallest possible number of rows. What exists are tables of upper bounds, kept for many years by Charles J. Colbourn, and counting arguments for lower bounds that go back to Rao (1947) and to Kleitman and Spencer (1973).

Can a finished table take one more column? Once a table is finished, someone asks for a 201st switch. Can you add it without adding a single new test? We call this one-column extension, and Chapter 5 explains it. The question was posed in the literature as decisional parameter extension (Garn and Simos, 2017, Problem 6; Garn's 2019 thesis, Problem 5.1.6) with its complexity left open there. Kokkala (2017) and Hiess (2024) gave the criterion that decides it.

A plain answer to both, in one line each: for the question about rows, we supply 29 tables that are smaller than the upper bounds in one pinned snapshot of Colbourn's tables and in Smolen's living table, and we do not know whether they are the smallest; for the question about columns, deciding it is easy to state and, from strength 3 up, NP-complete (written proofs, with Lean checking the construction).

Where to go next: can a finished table really take one more column, and how would you tell? See Chapter 5: room for one more column. Could the tables be smaller still? See Chapter 10: strength four rules and bounds.

Why it matters

Each row of a covering array is one test run, so a table with fewer rows is a cheaper test plan for the same coverage. Covering arrays are used to plan tests of systems with many on/off settings, and the question of how small they can be has been studied for about fifty years without being settled. The second question matters because test plans change: if a new setting must be added, it helps to know whether the old tests can be reused as they are.

Where to go next: what is a covering array, in plain words? See Chapter 1: covering arrays in plain words. Where do the 29 tables come from? Chapter 2 (seeds and changes) is held from this release; chapter 5 says how they were made.

What answering it would help with

Smaller covering arrays cut the cost of testing a configurable system without giving up the coverage. A reliable answer to the extension question lets a planner decide whether a new setting can be added to an existing test plan without new tests. The result that the question is NP-complete from strength 3 up (a written proof) guides the choice of methods for a quick exact check that works for a given table; it does not rule out good methods for particular tables. The link between strength 2 and a well-known hitting-set puzzle (monotone duality; given a list of groups and a proposed list of the smallest-possible sets of items that each touch every group, deciding whether the proposed list is complete), whose complexity is still open, connects that puzzle to test design. The arrays are plain text with fingerprints, so others may reuse them or compare them with other tables.

Where to go next: Chapter 5: room for one more column explains the second question, and Chapter 9: strength three is NP-complete explains why it is hard.

What our research shows

> Status

A 13 BY 13 LOCATING ARRAY AND 29 COVERING ARRAYS CHECKED EXHAUSTIVELY, WITH WRITTEN PROOFS AND LEAN THEOREMS FOR THE THEORY OF ADDING ONE COLUMN; CHECKED WITHIN THIS PROJECT; NOT YET REFEREED.

What this is

A BINARY (1-BAR,2)-LOCATING ARRAY WITH 13 ROWS AND 13 COLUMNS, AND 29 BINARY COVERING ARRAYS OF STRENGTH FOUR, WITH 200 TO 206, 223 AND 325 TO 345 COLUMNS, EACH CHECKED EXHAUSTIVELY; AT EACH OF THOSE 29 WIDTHS THE ARRAY SUPPLIED HAS FEWER ROWS THAN THE UPPER BOUND IN ONE PINNED SNAPSHOT OF CHARLES COLBOURN'S COVERING ARRAY TABLES AND THAN THE SMALLEST SIZE IN SMOLEN'S LIVING TABLE AT COMMIT 0FC74C0. ALSO IN THIS BOOK: A THEORY OF ADDING ONE MORE COLUMN TO A FINISHED TABLE (STRENGTHS 2, 3 AND HIGHER), 48 LEAN THEOREMS WITH THEIR SOURCES, FURTHER ARRAYS OF STRENGTH 2 AND SMALL EXACT CASES, AND A LEDGER OF WHAT FAILED.

Still open

THE SMALLEST POSSIBLE SIZE OF EACH COVERING ARRAY; WHETHER 13 ROWS IS THE MINIMUM FOR THE LOCATING ARRAY (CONJECTURED: ONE SOLVER FOUND NO 12-ROW ARRAY IN A SAT ENCODING OF THE PROBLEM; A SECOND SOLVER HAS RE-CHECKED A SAMPLE OF THE SEARCH, NOT THE WHOLE SEARCH, AND THE SEARCH IS NOT PART OF THIS BOOK); AND THE LITERATURE BEYOND THE SNAPSHOTS AND SOURCES NAMED IN THE BOOK (THE LITERATURE CHECK FOR THE LOCATING ARRAY IS INCOMPLETE).

Checked by

FOR THE COVERING ARRAYS: TWO SEPARATELY WRITTEN PROGRAMS OF OURS (PYTHON AND C#), WHICH ARE NOT PART OF THIS BOOK, TESTED EVERY CHOICE OF FOUR COLUMNS, AND A GENERAL CHECK OF THE DEFINITION COVERS OTHER STRENGTHS; THE RECIPES IN THE CHECK GUIDE LET YOU DO THE SAME WITH YOUR OWN CODE. FOR THE 13 BY 13 LOCATING ARRAY: TWO SEPARATELY WRITTEN CHECKERS OF

> OURS, AND RECIPE 11 OF THE CHECK GUIDE, WHICH YOU CAN FOLLOW WITH A FEW LINES OF YOUR OWN CODE. FOR THE LEAN FILES: THE LEAN KERNEL, LEAN'S OWN REPLAY CHECKER (LEANCHECKER) AND A SECOND KERNEL CHECKER (NANODA), RUN ON THE 48 THEOREMS AND ON THE TWO FURTHER LEAN FOLDERS ALIKE. FOR EACH RESULT, THE REGISTER OF RESULTS SAYS WHAT KIND OF CHECK IT HAD.
> SUPPORTING FILES AVAILABLE ON REQUEST.
> Why it's public
> IF YOU CAN CHECK FURTHER, OR FIND A MISTAKE, PLEASE TELL US.

In plain words. Each strength-four array here grows from R. E. A. C. Paley's 1933 construction from quadratic residues. The arrays for widths 200 to 206 start from Christopher Smolen's 141 × 199 array (ca-tools, CC BY 4.0), built from that construction, to which I added columns and rows by the rule of chapter 5, and for widths 202 to 206 removed and repaired rows. This row-removal-and-repair step follows the post-optimization approach of Nayeri, Colbourn and Konjevod (2009; 2013): adjust a covering array to reduce its rows while keeping full coverage (Peyman Nayeri, Charles Colbourn and Goran Konjevod, Randomized postoptimization of covering arrays, IWOCA 2009, Lecture Notes in Computer Science, doi:10.1007/978-3-642-10217-2_40; journal version, Randomized post-optimization of covering arrays, European Journal of Combinatorics 34 (2013) 91-103, doi:10.1016/j.ejc.2012.07.017). The comparison table this book links lists post-optimization constructions for many of its rows (postop NCK, postop TJ-SA). No priority is claimed for this step. The arrays for widths 325 to 328 start in the same way from his 165 × 324 array (ca-tools, CC BY 4.0), with columns and rows added by the same rule. The arrays for width 223 and widths 329 to 345 start from the half-Paley array of order 359, which I computed from Paley's formula, cut to width and then trimmed by removing and repairing rows. This row-removal-and-repair step follows the post-optimization approach of Nayeri, Colbourn and Konjevod (2009; 2013): adjust a covering array to reduce its rows while keeping full coverage (Peyman Nayeri, Charles Colbourn and Goran Konjevod, Randomized postoptimization of covering arrays, IWOCA 2009, Lecture Notes in Computer Science, doi:10.1007/978-3-642-10217-2_40;

journal version, Randomized post-optimization of covering arrays, European Journal of Combinatorics 34 (2013) 91-103, doi:10.1016/j.ejc.2012.07.017). The comparison table this book links lists post-optimization constructions for many of its rows (postop NCK, postop TJ-SA). No priority is claimed for this step. Chapter 5 gives the details, and each array's data file gives its exact construction. We stand on the shoulders of giants: the construction, the starting table, the column criterion (Kokkala and Hiess), the comparison tables and the bounds are other people's work, credited in the credits page. There is one array at each width, and all 29 are free to share, redistribute and build on under CC BY 4.0, with credit to Sonny (gocheckyourself.org), and to Christopher Smolen as well for widths 200 to 206 and 325 to 328 (his CC BY 4.0 licence allows building on his tables with credit).

The second half of the book asks about a finished table: can it take one more column, keeping every row? At strength 2 it is the same question as a famous hitting-set puzzle; at strength 3 and above it is NP-complete (written proofs, with Lean checking the construction); at strength four we used the exact rule to build the arrays above, and whether the widest one can take a 346th column is open.

What's in this book

All counts are computed from the files of the book (the results data file for the register, the folders for the rest).

	Count
Results in the register of results (its nine status counts add up to 117)	117
PROVED-Lean (a Lean proof is shipped)	12
PROVED-written (a proof on paper, in the proofs folder)	65
COMPUTED (a finite computation, with sizes)	27
CONJECTURED (believed, with the basis stated, and not yet confirmed by a check anyone can run)	1
CONDITIONAL (valid only under a stated condition)	2
OPEN (stated as open)	4
REFUTED (tested and found false; in the ledger, chapter 16)	3
SPECIAL CASES (covered by a more general result; in the ledger)	1
UNCONDITIONAL BOUND (a bound that stands without a stronger conditional one; in the ledger)	2
Lean theorems in the verified package	48
Further Lean declarations with an axiom check: rules folder, hardness folder, bridge file	29, 12 and 4
Lean source files	27
Arrays (strength four: 29; strength two: 20 plateau tips and the alphabet-7 pair; small exact cases: 2; locating array: 1)	54
Fixture tables, DRAT proofs	5, 1
Written check recipes	11
Chapters	15
Written proofs (the proofs folder and the hub-row folder)	7 and 2 files
Files in the book (147 of them online; the fingerprint list certificates/SHA256SUMS.txt covers the 113 files in the certificates folder, and the check recipe prints the fingerprint of each other file it uses)	206

The Lean-proved results in the register depend on the theorems named in the Lean theorem table; a PROVED-Lean row states only what its Lean theorems cover; a written or computed step beside it (for example a running time, or a comparison over many arrays) has its own row with its own status. The strength-2 collection of chapter 15 is a summary: 22 of its arrays are shipped and the rest are listed.

Yesterday, today, tomorrow

Yesterday. R. E. A. C. Paley (1933) built orthogonal matrices from quadratic residues, and tables cut from them serve as starting tables for binary covering arrays of strength four; Christopher Smolen built a 141 × 199 array and a 165 × 324 array from that construction and placed them in his ca-tools table under CC BY 4.0. Charles Colbourn kept public tables of upper bounds on the smallest covering arrays, Ulrike Groemping preserved the November 2024 version, and Christopher Smolen turned it into a CSV file and grew it into a living table. Locating arrays are studied by Colbourn and McClary (2008) and by Konishi, Kojima, Nakagawa and Tsuchiya (2019, 2020), who report 14 rows for 13 columns (chapter 13).

Today. I started from those two tables (chapter 5; chapter 2, seeds and changes, is held from this release), tested all 29 arrays on every choice of four columns with two separately written programs (chapter 4), ran all 29 through the living table's own check script, check_submission.py, and compared their sizes with the pinned snapshot and the living table (chapter 3). Still open: the literature beyond these two versions of the table, and the smallest possible size. What was shown about adding one more column is in chapters 5 to 12, with statuses, and the 13 by 13 locating array is in chapter 13.

Tomorrow. The arrays are plain text with checksums, so others may reuse them, compare them with other tables, or try the same route from the half-Paley arrays of other primes. The smallest possible numbers of rows stay unknown, width 346 and above is not covered, and the 13-row minimum for 13 locating-array columns is CONJECTURED, on the result of one solver in one SAT encoding. Recommended further research: reading the one unread locating-array paper, a full replay of the 12-row search with a second solver, Lean proofs of the running-time parts, outside review of the proofs that are only written, a search for a 346th column whose power has been tested, and a full second check of the older strength-2 collection.

Where to go next: can the widest array take a 346th column? See Chapter 11: the 346th column. What did not work along the way? See Chapter 16: the ledger.

Check it yourself

The check guide says what is claimed, what is not, and gives a written recipe for each check: the exact file format, the exact condition to test, the expected answer, and a standard public tool or a few lines you can write yourself. In short:

- the fingerprint list for every file is checked with any SHA-256 program (SHA-256 gives each file a 64-character code; change one bit and the code changes);
- each strength-four array is checked by testing every choice of four columns for all 16 patterns, and the guide lists the fingerprint of all 29;
- the Lean proofs are checked with the Lean build and a printout of the axioms of every theorem, and optionally with the nanoda kernel checker;
- the one exact case with a SAT proof is checked with drat-trim (a program that checks a proof log for a puzzle with no solution);
- the 13 by 13 locating array is checked by listing all 312 pairs of switch settings and testing that each shows in some row and that no two show in the same rows (recipe 11); a few lines of Python with no extra libraries are enough.

How I direct this work, and my own tools, are not part of this book. What ships: the 29 strength-four arrays and the further arrays, the 13 by 13 locating array, the Lean package of the 48 theorems, the written proof and data of the hub-row construction, a SHA-256 fingerprint for each file and a written recipe for each check. What is not online, and is available on request: the pinned comparison table, the two further Lean folders (hardness and rules), the written proofs for chapters 6 to 13, the solver outputs and search logs, the proof files behind the exact 14-row minimum and the four-isolated-models theorem, and the arrays of the Ladder.

Where to go next

How is an array checked, and why does the check take minutes? See Chapter 4: how the check works. The chapters:

- Chapters 1 to 4 on covering arrays, where the arrays come from, the table and the check: covering arrays in plain words, seeds and changes (chapter 2, held from this release), the comparison table, how the check works.
- Chapters 5 to 13 on adding one more column: room for one more column, strength two and duality, strength three and pair classes, the eighteen walls, strength three is NP-complete, strength four rules and bounds, the 346th column, the Mirror Route and the Ladder, other kinds of tables.
- Chapters 14 and 15 on further arrays: more arrays and the strength-two collection; chapter 16, the ledger.

- The statement page, the register of results and the Lean theorem table are in the statement folder; the arrays are in the certificates folder; the Lean files are in the Lean folders; the pinned comparison table is not on this site (it is available on request, with its MIT licence text, and the same file is public at the pinned commit of Smolen's ca-tools repository; see chapter 3). The credits and the licence page complete the book.

Go check yourself.

Have a question about this? Ask it

2 Covering arrays, the short version

Covering arrays, the short version

The question: how few tests can check every combination of a machine's switches, and can a finished test table take one more switch? The answer in brief: I found a 13 by 13 locating array. I built 29 strength-four covering arrays, each with fewer rows than the upper bound in one pinned snapshot (fixed to one exact version) of Charles Colbourn's tables at its width. And deciding whether a table has room for one more column is NP-complete (among the hardest problems whose answers can be checked quickly; no quick method is known for them) from strength 3 up.

Tests as tables

In plain words. Suppose a machine has 200 switches, each on or off, and you want to test it. Trying every setting is impossible, so you aim for less: whichever four switches you pick, every one of their 16 on/off patterns should show up in at least one test. A table of 0s and 1s that does this is called a binary covering array of strength four. Each row is one test, each column is one switch, and the number of rows is the cost.

The 13 by 13 locating array

In plain words. A locating array is a test table in which the failing tests tell you which pair of switch settings is faulty. I found a binary (1-bar,2)-locating array with 13 rows and 13 columns. The tables we found in the literature we checked have 14 rows. Two separately written checkers of ours confirm the table. One paper (Shi, Fu, Wang and Yan, 2021) is unread, so a table that small may already exist there. That 13 rows is the minimum for 13 columns is conjectured: one solver found no 12-row table, and a second solver has re-checked a sample of that search.

29 strength-four arrays

In plain words. For the widths in this book (200 to 206, 223 and 325 to 345 columns), nobody knows the smallest possible number of rows. The 29 arrays grow from R. E. A. C. Paley's 1933 construction. Each has fewer rows than the upper bound in one pinned snapshot of Colbourn's tables and than the smallest size in Smolen's living table at its width. Two separately written programs of ours tested every choice of four columns, and the living table's own check script, check_submission.py, passes all 29. The arrays are free to share under CC BY 4.0 (a licence that lets anyone share and build on the work with credit).

Room for one more column

In plain words. Once a table is finished, someone asks for one more switch. Can you add it without adding a single test? Garn and Simos posed this in 2017, and Kokkala (2017) and Hiess (2024) gave the criterion that decides it.

Deciding it is NP-complete from strength 3 up: written proofs, with Lean (a program that checks every step of a proof) checking the construction. So no quick exact method is expected for every table, though good methods may exist for particular tables.

Status

Checked within this project.

Experts: the files and checks are here: check guide.

3 Covering arrays in plain words

Covering arrays in plain words

The question

If a machine has many switches, how few tests are enough so that every on/off pattern of any four switches shows up in at least one test?

Why it matters

Each row of a covering array is one test run, so fewer rows mean a cheaper test plan with the same coverage.

What answering it would help with

Knowing the terms lets a reader check any array in this book, or one of their own, without trusting us.

What our research shows

A binary covering array of strength four is a table of 0s and 1s in which any four columns show all 16 patterns; a computer can check this by looking at every choice of four columns.

Suppose a machine has 200 switches, each on or off, and you want to test it. Trying every setting is impossible: there are 2 to the power 200 of them. A

cheaper aim is this: whatever four switches you pick, every one of their 16 on/off patterns should show up in at least one test.

A binary covering array of strength four is a table that does exactly that.

- Each row is one test. The number of rows is the cost; we call it the number of runs.
- Each column is one switch. The number of columns is the width.
- Each entry is 0 or 1.
- Strength four means: in any four columns, all 16 patterns of 0s and 1s appear among the rows.

For a table with k columns there are k(k-1)(k-2)(k-3)/24 choices of four columns. For 325 columns that is 456,326,325 choices, each with 16 patterns to find. A computer can check this by enumeration. No one needs to trust the table; anyone can write the check.

The aim is few rows for a given width. A wider array gives a narrower one by deleting columns, so run counts never rise as the width falls. Nobody knows the fewest rows possible at these widths. Counting arguments give lower limits on the number of rows, but they do not settle the smallest possible number.

The main collection of this family holds 29 arrays of strength four, one for each width from 200 to 206, for width 223 and for each width from 325 to 345. Chapter 5 says where they come from: all grow from Paley's 1933 construction. Chapters 5 to 16 add the theory of adding one more column to a finished table, and chapters 14 and 15 describe further arrays and small exact cases.

Yesterday, today, tomorrow

Yesterday. Counting bounds for the exact-coverage relatives of covering arrays, called orthogonal arrays, go back to Rao (1947), and Kleitman and Spencer (1973) gave lower-bound inequalities for binary covering arrays, as surveyed by Lawrence, Kacker, Lei, Kuhn and Forbes (2011). These bounds do not settle the smallest possible numbers of rows, which remain unknown.

Today. The terms are explained above, and recipe 2 of the check guide runs the check by enumeration. I supply 29 arrays, for widths 200 to 206, 223 and 325 to 345, that anyone can check this way, and anyone may share them (CC BY 4.0).

Tomorrow. A reader with no special background can write the check and see the result for themselves. The question of the fewest rows possible at these widths stays open for anyone to work on.

Check it yourself

Recipe 2 of the check guide tests every choice of four columns of an array; recipe 1 checks the fingerprints.

Where to go next

- Where do the 29 arrays come from, and may I share them? Chapter 5: room for one more column.
- Can a finished table take one more column? Chapter 5: room for one more column.

4 The comparison table

The comparison table

The question

Is each of our arrays smaller than what the standard published tables list for its width?

Why it matters

A size means little without a fixed reference; a pinned snapshot makes the comparison repeatable by anyone.

What answering it would help with

It lets others repeat the comparison exactly, or bring it up to date against newer tables.

What our research shows

At every one of the 29 widths, our array has 3 to 11 fewer rows than the upper bound in one pinned snapshot of Colbourn's tables, and 1 to 7 fewer rows than the smallest size in Smolen's living table at commit 0fc74c0.

To say whether an array is small, we need something to compare it with. We use two fixed versions of one table, so the comparison does not change under us.

What it is. Charles J. Colbourn kept tables of upper bounds on covering array sizes, for many sizes, at Arizona State University. The November 2024 version was preserved by Ulrike Groemping. Christopher Smolen's ca-tools repository turned it into a CSV file, and has since grown it into a living table, to which anyone can submit arrays with evidence.

- The pinned snapshot. We pinned that CSV file at commit 5ad91a4925e55e4a40a0332ae6d1aa66777e2429, whose SHA-256 is 30e5f10e226f881a413dd0a61f98bd7dac6d4dfb4916cf36505bdc31844833c0. The table's source labels for the rows that give the bounds are "SBSTT (TJ-AG)" at widths 200 to 206 (the row of width 208), "Power CZ2-16.5" at width 223 (the row of width 256) and "Torres-Jimenez" at widths 325 to 345 (the rows of widths 338 and 354). Source not yet identified for the labels "SBSTT (TJ-AG)" and "Power CZ2-16.5".
- The living table. We also read Smolen's table at commit 0fc74c021747b6a160d421dca76bba5b6f3e1bf7. At these widths its smallest sizes include his own arrays built from Paley's construction (for example 148 rows at width 200 and 154 at width 225).

How we read it. For the pinned snapshot, take the rows of the CSV file that come from Colbourn's tables (snapshot colbourn-2024-11-ug) and, among those whose width is at least the one you want, the smallest run count. The

same file also holds rows added after Colbourn's tables (for example 153 rows at width 222); they are not part of the snapshot. For the living table, take the smallest run count among rows marked CITED and VERIFIED whose width is at least the one you want, as its own check script, check_submission.py, does. A wider array can be narrowed by deleting columns, so such a row gives a valid bound. For width 206 the pinned bound is 154, from Colbourn's row of width 208; for width 223 it is 160, from the row of width 256.

What we found.

Widths	Our rows	Pinned snapshot	Living table at commit 0fc74c0
200	143	154	148
201 to 206	145	154	149
223	153	160	154
325	166	173	173
326	167	173	173
327 and 328	169	173	173
329 to 338	170	173	173
339 to 345	170	174	174

At each of the 29 widths our array has fewer rows than both. The margin is 3 to 11 rows against the pinned snapshot and 1 to 7 rows against the living table; the smallest margin, 1 row, is at width 223.

The maintained table. The ca-tools table has grown since the commit above. At commit 8e256aa41d58d9745bfefa0f37e0afeecd483a1b, read by the same rule, it lists binary strength-four arrays by Christopher Smolen, built with the Paley construction: 137 rows at 187 columns, 149 rows at 222 columns and 154 rows at 225 columns. At width 206 our array has 145 rows against his 149, and at width 223 it has 153 against his 154. In the version we read of Smolen's ca-tools table, all 29 of our arrays are listed there, each the smallest listed at its width (credited Sonny): table/ca_table.csv at commit ce719cb. The comparison in the table above is with the pinned snapshot and with the living table at commit 0fc74c0 only.

What this shows. Our arrays are below the bound in these two versions of the table, and all 29 pass check_submission.py. Still open: the smallest possible size at each width, and arrays outside these two tables; tables change, and we did not search the whole literature. Please tell us if you know of one.

Our copy of the pinned CSV is not online on this site; it is available on request, with its MIT licence text from the same commit. The same file is public in Christopher Smolen's ca-tools repository at that commit: table/ca_table.csv at commit 5ad91a4, and the living table is in the same repository: table/ca_table.csv at commit 0fc74c0. Download either, check the pinned file's SHA-256 against the value above, and you can compare the bounds by hand under the rule above. Colbourn's original pages at Arizona State University asked for a university sign-in when we last looked.

Yesterday, today, tomorrow

Yesterday. Colbourn's tables were the standard reference for many years. The pinned file is described above; Smolen's living table grows from it.

Today. I compared each of the 29 widths with the pinned file and with the living table under the stated rule. The pinned bounds were recomputed from the CSV file; all 29 agree with the table above.

Tomorrow. Because both versions are pinned to a commit, anyone can repeat the comparison and get the same answer. Readers who know of smaller published arrays at these widths are invited to say so, so that the comparison can be brought up to date.

Check it yourself

Recipe 7 of the check guide recomputes all 29 bounds from the pinned CSV file (not on this site; download it from Smolen's repository at the pinned commit, as above).

Where to go next

- How is an array checked, and how long does it take? Chapter 4: how the check works.

- What about adding a column to a finished table? Chapter 5: room for one more column.

5 How the check works

How the check works

The question

Given a table of 0s and 1s, how do you know it is a covering array of strength four?

Why it matters

A claim about an array is only as good as the check behind it, and this check needs no trust in us.

What answering it would help with

A reader can test any binary strength-four array stored in the same plain format.

What our research shows

All 29 arrays pass the full check of every choice of four columns, by two separate programs; nothing is sampled.

The full check goes through every choice of four columns and every one of the 16 patterns, and looks for a row that shows it. If any pattern is missing, the array fails and the check can name the four columns. Nothing is sampled. For 345 columns there are 580,078,170 choices of four columns, which is 9,281,250,720 pairs of a choice and a pattern. A direct program in a compiled language checks all 29 arrays in about 5 minutes on one core of an ordinary computer; an interpreted language needs much longer.

The 29 arrays were checked by two separate programs, one in Python and one in C#. Both found all 29 arrays to be covering arrays. The check does not

depend on how the arrays were made, so a reader can write their own program from the written recipe in the Check it yourself guide: the exact file format, the exact condition, the expected answer and the fingerprint of every file.

The fingerprint check takes seconds. A SHA-256 tool compares each file with the fingerprint list. It confirms that the file you hold is the file we published; it does not test coverage.

Yesterday, today, tomorrow

Yesterday. A covering array can be checked from its definition, without any trust in how it was made. The tables kept by Colbourn and preserved by Groemping give sizes and sources, and Smolen's ca-tools repository carries many arrays themselves as data, including the 141 × 199 array that seven of ours start from.

Today. The C# route takes about 5 minutes for all 29 arrays on one core. The Python route takes several hours on one core (a check written from the recipe took about 1.5 minutes for a 201-column array and about 8 minutes for a 345-column array), or under an hour spread over 16 processes. A fingerprint check takes seconds. If a pattern is missing, the check names the four columns.

Tomorrow. The same recipe applies to any binary strength-four array in the same plain format, and a failure names the columns and pattern, which tells someone where to repair an array. A reader with a compiler can write a checker and test an array of their own.

Check it yourself

Recipes 1 and 2 of the check guide give the fingerprint check and the full four-column check.

Where to go next

- Can a finished table take one more column? Chapter 5: room for one more column.

- Which other arrays were checked? Chapter 14: more arrays.

6 Room for one more column?

Room for one more column?

The question

Take a finished covering array. Can you add one more column and keep every row as it is?

Why it matters

Test plans change; knowing whether a new setting fits into the existing tests can save new test runs.

What answering it would help with

The colouring rule decides such cases directly for small tables, and it frames the rest of the book.

What our research shows

Sometimes yes and sometimes no: a table can take one more column exactly when its rows can be coloured 0 and 1 so that no required group of rows is all one colour, and this has been proved in Lean.

Chapters 1 to 4 were about finished tables: 29 arrays, each checked. This chapter asks a different question, and the rest of the book follows from it. It is also where the book says how those 29 arrays were made, because the rule below is the tool that built those at widths 200 to 206.

Take a finished table. Can you add one more column, and keep every row as it is?

In test language: a covering array has been designed for 200 switches. Someone asks for a 201st switch. Can you add it by writing one more 0 or 1

into each existing test, so that every group of switches still shows all its patterns, and without adding a single new test? We call this one-column extension.

A small example

Start with the 16-row table that lists every pattern of four switches. It has strength 3 (indeed strength 4). Ask for a fifth column that keeps strength 3. There are 7,842 ways. Now take 8 rows: the three switches x, y, z written in all 8 ways, plus a fourth column x + y + z (counted mod 2). Every three switches show all 8 patterns, so this has strength 3. But there is no way to add a fifth column: this table is finished in a strong sense, a fixed point. A table that has room is not always the one you would guess. Anyone can recompute both counts from the colouring rule below; the Check it yourself guide gives the exact test.

The rule that decides it

Pick two switches, say columns a and b, and one of the four settings of them, say $a = 0$ and $b = 1$. Look at the rows that show that setting. Call that set of rows a pair class. A new column keeps strength 3 exactly when it contains both a 0 and a 1 inside every pair class. If a pair class were all 0 in the new column, then the three switches a, b and the new one would never show the pattern 0, 1, 1.

So adding a column is a colouring puzzle. Colour each row 0 or 1 so that no pair class is all one colour. That is the whole of it, and Lean has checked it:

- The class lemma (every strength): a new column keeps strength t exactly when it takes both values on every class defined by t-1 of the old columns. Proved in Lean.
- The support criterion (strength four): for every choice of three old columns and every pattern on them, the rows showing that pattern must contain both a 0 and a 1 of the new column. Proved in Lean.

Both are in the 48-theorem Lean package shipped in this book; the Lean theorem table explains each theorem in plain words. The criterion itself is

known in the literature (Kokkala's 2017 thesis, section 3.5, and Hiess's 2024 thesis); we claim no priority for it. Everything built on it in the next chapters is where our work is, and for each result the register of results says exactly what was proved and how it was checked.

How the strength-four arrays of this book were made

The 29 strength-four arrays of this book grow from one piece of classical mathematics: R. E. A. C. Paley's 1933 construction from quadratic residues. There are three starting tables.

- Widths 200 to 206. I started from Christopher Smolen's CA(141; 4, 199, 2), a table of 141 rows and 199 columns in his ca-tools table (which he built from Paley's construction). He offers it under CC BY 4.0. I added columns one at a time with the rule above: each added column takes both values on every class of rows, and where a column needed it, I added a row. The 143 × 200 and 145 × 201 arrays keep every cell of his table unchanged. For widths 202 to 206 I went on to 148 rows and 206 columns, then removed rows one at a time, repairing each removal by changing cells in a local search, until 145 rows were left; the arrays for widths 202 to 205 keep the leading columns of the 145 × 206 array. This row-removal-and-repair step follows the post-optimization approach of Nayeri, Colbourn and Konjevod (2009; 2013): adjust a covering array to reduce its rows while keeping full coverage (Peyman Nayeri, Charles Colbourn and Goran Konjevod, Randomized postoptimization of covering arrays, IWOCA 2009, Lecture Notes in Computer Science, doi:10.1007/978-3-642-10217-2_40; journal version, Randomized post-optimization of covering arrays, European Journal of Combinatorics 34 (2013) 91-103, doi:10.1016/j.ejc.2012.07.017). The comparison table this book links lists post-optimization constructions for many of its rows (postop NCK, postop TJ-SA). No priority is claimed for this step.

- Widths 325 to 328. I started from Christopher Smolen's CA(165; 4, 324, 2), a table of 165 rows and 324 columns in his ca-tools table (which he built from Paley's construction). He offers it under CC BY 4.0. I added columns one at a time with the rule above: each added column takes

both values on every class of rows, and where a column needed it, I added a row. The arrays keep every cell of his table unchanged, and no rows were removed. The added rows and columns are 1 and 1 at width 325, 2 and 2 at width 326, 4 and 3 at width 327, and 4 and 4 at width 328.

- Width 223 and widths 329 to 345. I computed the half-Paley array of order 359 from Paley's formula, so these arrays use no one else's file. Its 179 rows are the nonzero squares x modulo 359, its 358 columns are y = 1 to 358, and an entry is 1 when x + y is a nonzero square modulo 359. I cut it to the columns needed and removed rows one at a time in the same way: 153 rows at width 223, and 170 rows at each width from 329 to 345.

Keeping the leading columns of a wider array keeps every choice of four columns covered, so one search can serve several widths. The data file beside each array gives its exact construction: the starting table, how many rows and columns were added or removed, and the width it was cut from. Every array passes the check of every choice of four columns by two separately written programs (chapter 4). At each of the 29 widths the array has fewer rows than the smallest size listed at that width or wider in Smolen's living table at commit 0fc74c0 (chapter 3).

The arrays are free to share. They are offered under CC BY 4.0: anyone may share, redistribute and build on them, with credit to Sonny (gocheckyourself.org), the site that publishes this book. The arrays for widths 200 to 206 and 325 to 328 also carry Christopher Smolen's credit for the 141 × 199 and 165 × 324 tables they start from; he offers those tables under CC BY 4.0, which allows building on them with credit. A suggested credit line: "Covering array from Sonny, gocheckyourself.org, CC BY 4.0; built from Paley's construction (1933)", adding "starting from C. Smolen's CA(141; 4, 199, 2), ca-tools, CC BY 4.0" for widths 200 to 206, or "starting from C. Smolen's CA(165; 4, 324, 2), ca-tools, CC BY 4.0" for widths 325 to 328. The licence is summarised in LICENSE-CC-BY-4.0.txt; as CC BY 4.0 asks, say so if you changed an array.

What the rest of the book does with it

Chapter	Question	What we found
6	Strength 2: can one more column be added?	The question is exactly the same as a famous hitting-set puzzle, monotone duality. It can be decided in quasi-polynomial time, and (assuming the Exponential Time Hypothesis) it is not NP-hard.
7	Strength 3: what makes a table easy?	Pair classes, a counting test, a local-lemma test, small cases.
8	Why do the obvious ways of proving hardness fail?	Eighteen walls: a written map of failed routes.
9	Is strength 3 hard after all?	Yes: NP-complete, and so is every fixed strength from 3 up. A Lean proof covers the construction's correctness.
10	Strength four: rules we use to build and extend arrays.	An exact rule for finishing a table with a few extra rows, growth that works but costs too much, lower bounds.
11	A 346th column for the 345-column array	Open for the 170-row array supplied here; the chapter gives the rule such a column must meet.
12	The Mirror Route and the Ladder	An attempt to turn strength 3 into a duality question, with a scoreboard.
13	Other kinds of tables	Three symbols instead of two.
14	More arrays	Small exact cases: a cyclic array, a non-extendible array, a proof about constant rows.
15	The collection of strength-2 arrays	A summary, with which arrays were fully re-checked.
16	The ledger	What did not hold, and what is still open.

What a reader should keep in mind

Every claim in these chapters is one of: a Lean theorem (status PROVED-Lean), a proof written on paper (PROVED-written), a finite computation (COMPUTED), a statement that holds only under a stated condition (CONDITIONAL), or an open question. The register gives the status of each, and the chapters never state a result more strongly than its status.

Yesterday, today, tomorrow

Yesterday. Garn and Simos (2017, Problem 6; Garn's 2019 thesis, Problem 5.1.6) posed the question as parameter extension, with its complexity left

open there. Kokkala (2017) and Hiess (2024) gave the colouring criterion and used SAT solvers to extend arrays.

Today. Sonny wrote the strength-2 and strength-3 results in the next chapters, other than those credited there to others (the class criterion is Kokkala's and Hiess's; the reverse reduction from duality to a table is Eiter and Gottlob's, who credit Seymour; the link between duality and colouring intersecting hypergraphs is Elbassioni's and Kokkala's), with checks and a Lean package for the finite parts, and used the criterion to build the strength-four arrays at widths 200 to 206 (above).

Tomorrow. The recipes work on any small binary array in plain text. A reader can try the colouring puzzle on their own tables, check the Lean files, or look for other tables that are finished in the sense above. Recommended further research: a Lean proof of the polynomial-time parts, and an outside review of the proofs that are only written.

Check it yourself

Recipe 4 of the check guide counts the valid new columns of five small tables; recipe 9 builds the Lean package that proves the rule.

Where to go next

- Is the question easier at strength 2? Chapter 6: strength two and duality.
- Is it hard at strength 3? Chapter 9: strength three is NP-complete.

7 Strength two: one-column extension and a famous hitting-set puzzle

Strength two: one-column extension and a famous hitting-set puzzle

The question

At strength 2 every pair of switches must show all four patterns.

The one from chapter 5: can a finished strength-2 table take one more column, and how hard is it to decide?

Why it matters

At strength 2 the question turns out to be monotone duality, a hitting-set problem whose exact complexity is still open.

What answering it would help with

Progress on either question may carry over to the other.

What our research shows

At strength 2 the question is the same as a well-known hitting-set puzzle, so it can be decided in quasi-polynomial time (a written proof) and, assuming the Exponential Time Hypothesis, is not NP-hard (a conditional statement).

The idea in plain words

Pick one row of the table and call it the reference row. Flip columns (swap 0 and 1 in a column; this never changes whether a table is a covering array) until the reference row is all zeros. Now each column is described by the set of other rows that hold a 1. Call these sets T_1, T_2, T_3 and so on.

For the table to have strength 2, two things must hold, because the pair (0,0) is guaranteed by the reference row and the other three patterns need witnesses:

- any two of the sets must meet (some row has a 1 in both columns), and
- no set may contain another (otherwise a pattern is missing).

A list of sets with these two properties is an intersecting antichain. The table can take one more column exactly when this list is not maximal: when some further set could be added without breaking the two properties. Lean checks the anchored form of this.

Hitting sets and duality

Take a list of sets. A hitting set (also called a transversal) picks at least one element from every set in the list. The minimal hitting sets are the hitting sets with nothing to spare. Given a list and a proposed list of its minimal hitting sets, monotone duality asks: is the proposed list complete? (If the proposed list is the full list of minimal hitting sets, the two lists are dual.) A well-known algorithm for this question, by Fredman and Khachiyan (1996), takes quasi-polynomial time: not polynomial, but far below exponential. Whether a polynomial algorithm exists is a well-known open problem.

What was shown

- Reduction to duality questions. For every binary strength-2 table with N rows, the question 'can it take one more column?' reduces to N-1 duality questions, each of polynomial size. Hence strength-2 extension can be decided in quasi-polynomial time. Status: PROVED-written, accepted in a separate reading, and compared with brute-force search on thousands of random tables with no disagreement.
- No likely NP-hardness. If someone found a polynomial reduction from not-all-equal 3-satisfiability (an NP-complete problem) to strength-2 extension, then that problem could be decided in time $2^{o(\log^2 n)}$, contradicting the Exponential Time Hypothesis. Status: CONDITIONAL, since it depends on that hypothesis. It indicates that strength-2 extension is unlikely to be NP-hard; further research is recommended before drawing a wider conclusion.

- The equivalence with duality. Strength-2 extension and monotone non-duality can each be turned into the other, with the answer preserved. Lean proves both reductions correct, with their size bounds (see the Lean theorem table). That each can be carried out in polynomial time is a separate, written result (PROVED-written). The reverse direction (from duality to a table) is, word for word, Theorem 4.13 of Eiter and Gottlob (1995), who credit Seymour. The forward direction, from any covering array to a duality question, is the part we did not find stated in the sources checked for it (Katona 1973, Kleitman and Spencer 1973, the 2011 survey of Lawrence and co-authors, Eiter and Gottlob 1995,

the literature on non-dominated coteries, Elbassioni 2016, Colbourn and McClary 2008, and Shi, Tang and Yin); a specialist should be asked before anyone calls it new. The running time of carrying out the construction is proved on paper only, not in Lean.

Easy cases and a counter-example

- Easy cases. In the family form, the table extends exactly when some hitting set exists that neither contains nor is contained in any set of the list (an incomparable hitting set). That the list is maximal is the same as saying that a related list of minimal hitting sets is complete. And a counting test gives a polynomial algorithm: if the number $\Lambda = \Sigma_j [2\^(1-|S_j|) + 2\^(1-|T_j|)]$ is below 1, an extension exists and can be built deterministically.
- A local test that is not enough. A list in which every set passes the local test (it really is a minimal hitting set) can still be incomplete: in a valid 8-row, 10-column table with exactly four extensions, twelve listed sets pass every local check, yet the full list has fourteen. Checking one set at a time is easy; checking that the list is complete is the hard part. Lean proves the local test.

- A shortcut that fails. Maximal does not mean self-dual. All four triples of {a,b,c,d} form a maximal intersecting antichain, yet their hitting-set list is the six pairs, and the 5×4 table they define does not extend.
- Three further results. For a fixed 'rank spread' the decision is polynomial; the exact number of extensions is $2W(F)$; and a rank count gives a sufficient certificate.
- Stuck arrays. Seven tables for which a SAT solver answered 'no more columns fit' each have a small unsatisfiable core. At strength 2 the core uses every pair of columns; at strength 3 only 5 to 18 pairs. The answer concerns that table only and is never a lower bound for the smallest possible array.
- The hypothesis that matters. One lemma of the equivalence needs the array to be a strength-2 covering array; without that hypothesis it fails

for 310 families on four points. About 1.5 million cases agree with brute force.

How to check

The Lean package shipped in this book (build steps in the Check it yourself guide) covers the equivalence with duality. For the rest, small tables can be decided by exhaustive search for tables of up to a couple of dozen rows, using the colouring rule of chapter 5. The larger comparisons (2,901 random tables, 163 antichains and 27,524 pairs) were run in the working notes, which are not shipped; the register says how each statement was checked.

What this suggests

The strength-2 question lies between two well-studied things: it is not known to be easy, and (under a stated assumption) it is not NP-hard. That indicates strength 2 is a special case; whether the same is true for strength 3 is the subject of the next chapters, and further research is recommended.

Check it yourself

Recipe 9 of the check guide builds the Lean package, which proves the two reductions correct; small tables can be decided with the colouring rule of recipe 4. The register of results says how each written result was checked.

Where to go next

- Does strength 3 behave the same way? Chapter 7: strength three and pair classes.
- Why is strength 3 different? Chapter 9: strength three is NP-complete.

8 Strength three: pair classes, and when a table has room

Strength three: pair classes, and when a table has room

The question

At strength 3, every group of three switches must show all eight patterns. One-column extension asks again: can the finished table take one more column?

Why it matters

Strength 3 is where the question becomes hard (chapter 9), so its easy cases matter in practice.

What answering it would help with

The easy-case tests decide many practical tables quickly, without a full search.

What our research shows

At strength 3 the question is a colouring puzzle on pair classes; some tables are easy (a counting test, a local-lemma test), and a table can have room and still lose it when one row is deleted.

Pair classes

Chapter 5 gave the rule. Choose two columns a and b and one of their four settings. The rows that show that setting form a pair class. A new column keeps strength 3 exactly when it is both 0 and 1 somewhere in every pair class (the $t = 3$ case of the Lean-proved class lemma).

A few facts about pair classes:

- A table with k columns has exactly $4 \times C(k,2)$ pair classes, and no pair class contains another.
- Every pair class has at least 2 rows, and at most $N - 6$ rows.

- Checking a proposed new column takes O(N k^2) time, so the problem is in NP.

Easy cases

- The counting test. Give each pair class E the weight 2^(1−|E|), the chance that a random column is constant on it. If the total weight W is less than 1, a good column exists, and a deterministic polynomial algorithm finds it. Example: a 64×32 table has $W = 31/512$.
- The local-lemma test. A finer test with weights x_E: if 2^(1−|E|) ≤ x_E · Π(1 − x_F) over the classes F that overlap E, a column exists, and the resampling method of Moser and Tardos finds it. A 28×3 table has total weight $1153/1024$, which is more than 1, so the counting test is silent, but the local-lemma test passes. This uses the published Moser-Tardos theorem as given.
- Small intersections. If two pair classes never share more than b rows, extension is decidable in time O(2^(8b) N k^2). With $b = 1$ it is the question of whether a graph is bipartite.

- Dropping a column always leaves room. Every table obtained from a strength-3 table by deleting a column can be extended (put the deleted column back). So every table with no room is column-minimal.
- The smallest table with no room has 8 rows. Take x, y, z in all eight ways and the fourth column x + y + z: that is the parity table of chapter 5.
- Unit rows. For tables made of unit rows, the question is polynomial: it extends exactly when both constant columns are present, and then there are exactly two extensions.

Rooms and worked examples

Some small tables are good for trying the colouring rule by hand or by a short program of your own. They are shipped in the fixtures folder as plain text.

Table	Strength	Number of ways to add a column
the 16-row table of all patterns of four switches	3	7,842
the parity table with 8 rows and 4 columns	3	0
the extended Hamming code (16 words of 8 bits)	3	56
every table obtained from the Hamming-code table by deleting one row	3	0 (all 16)
the 4-cube minus the rows 0000, 0011, 1101 and 1110 (12 rows)	3	28

The 16 × 8 design is a fixed point of a different kind: it has 56 extensions, yet the loss of any single row leaves it with none. The same pattern (a table with room whose every subtable has none) is what the shared-design constructions tried to use; the lift of this design to larger members of the family fails from r = 4 on (chapter 8).

How strength-3 extension relates to duality

- Duality and extension. The complement of monotone duality reduces, in polynomial time, to strength-3 extension: from two families (J, G) one builds a strength-3 table that extends exactly when G differs from the hitting-set family of J. Status: PROVED-written, accepted in a separate reading, and compared with brute force on 60 instances. We claim no priority: the link between duality and the colouring of intersecting hypergraphs is in Elbassioni (2016) and Kokkala (2017).
- Padding and orientation. For an intersecting Sperner hypergraph J, the padded table is extendible exactly when J is properly two-colourable; and J is not two-colourable exactly when J equals its own hitting-set family.
- Bad hitting sets. Strength-3 extension is possible exactly when a certain list B(H) of bad hitting sets differs from the hitting-set family of H.

- How long the list can be. That list can be enormous: for the incidence tables H_m it has at least 2^binom(m−1,2) members, which grows like 2^Θ(√L) in the table size L. So the direct plan 'compute the list, then ask duality' fails. This does not rule out other plans (chapter 12).

Together with the statement of chapter 9 (strength 3 is NP-complete), these results indicate that strength 3 differs from strength 2 in an essential way. Further research is recommended: in the sources we checked, no one has shown, either way, whether strength-3 extension can be reduced to duality.

What was checked

Most of this chapter is PROVED-written, read in a separate reading and compared with brute force on small tables; the register says how each result was checked. The Lean package covers the class criterion itself and the finite facts about classes. The union-bound and local-lemma examples are exact computations on the stated tables.

Check it yourself

Recipe 4 of the check guide reproduces the counts of the worked tables in the fixtures folder; the register of results says how each result was checked.

Where to go next

- Why did the obvious ways of proving hardness fail? Chapter 8: the eighteen walls.
- Which route did prove hardness? Chapter 9: strength three is NP-complete.

9 The eighteen walls

The eighteen walls

The question

Why does the obvious way of proving that strength-3 extension is hard keep failing, and what does each failure teach?

Why it matters

A map of failed routes can save others from repeating them, and explains the shape of the route that worked.

What answering it would help with

It shows anyone designing another reduction which designs fail, and why.

What our research shows

Eighteen natural ways of building a hard-looking table failed, and each failure explains why the obvious route is blocked.

This chapter is a map of routes that did not work. We include it because a failed route teaches a reader where not to look, and because the walls explain why the route that finally worked (chapter 9) looks the way it does.

The goal and the attempts

The aim was to show that one-column extension at strength 3 is hard: to disguise any hard puzzle (say, whether a set system can be two-coloured) as a test table, so that the table can take a new column exactly when the puzzle is solvable. The attempts, in about twenty tries, each used a different way of building the table. Each hit an obstacle. The obstacles were grouped into eighteen walls in one document. A later reading of it found that the implications hold at their stated scope ("No false mathematical implication was found") and also found that the document does not establish a universal obstruction to a hardness reduction. Some walls are universal, some apply only to one kind of construction. That reading counted 110 results: 22 universal walls, 41 template-specific failures and 47 finite observations (aliases and assumptions included, so not 110 independent theorems). Among its 94 evidence-labelled rows, 59 were proved, 28 were computed and 7 rest on an assumption (the NP-completeness of the source problem, the strength-2 equivalence of chapter 6, the Moser-Tardos theorem, consequences of the Exponential Time Hypothesis, and the claim that no second recipe was tried).

That reading was done by reading, not by a proof assistant, and the document has not yet been read by anyone outside this project.

The walls, in plain words

Wall	What went wrong (plain words)	Kind
1	Strength 3 forces variation, not pairwise intersection; at least 8 rows are needed, and a table with no room needs at least 4 columns (the parity 8×4 table reaches both bounds).	universal
2	Every old column outside a class's defining columns already splits that class. So a family of classes that cannot be split uses every old column, and has at least half as many classes as columns (the transfer lemma, partly Lean-proved). The bound is sharp.	universal
3	If each source clause is enforced on its own private columns, small sources such as the Fano plane plus a disjoint triple cannot be represented.	template
4	Exact 'incidence labels' fail before a table exists: some satisfiable sources have no private-pair labelling.	template
5	If a frame of old columns can already take a column, adding any rows keeps it so, with the new rows' values free: padding cannot force anything.	universal
6	Washing: duplicated blocks wash out the constraints they were meant to carry.	universal
7	The shared affine frame loses its rigidity: the 16×8 design ($r = 3$) has 56 extensions but every single-row deletion leaves none; for $r \geq 4$ the larger frames survive single deletions, so what is seen at $r = 3$ does not persist.	universal / template
8	One common reference cannot support two separate equality stars.	universal
9	Complementary hubs enforce the wrong clause: all three unsatisfiable test sources become satisfiable.	template
10	Clean-star and signed-guard constructions each hit a narrow impossibility certificate.	template
11	Foreign auxiliary variables can defeat opposite guards at the same time.	universal
12	A semantic 'washing' criterion, and a counter-example to the claim that washing is inevitable.	universal
13	Support coverage is not enough: protecting all admissible classes accepts exactly the middle weights, $2^g - 2g - 2$ assignments, so no NO instance exists for $g \geq 4$.	template
14	The compact feasibility form is not ordinary duality.	universal
15	Changing both duality families does not help: a private-intersection family always has a false source gap, and repairing one family needs at least 3^M terms.	template
16	Positive auxiliaries cannot compress the forbidden formula below the transversal count.	template
17	Why the full bad-transversal list can be superpolynomial.	template
18	Testing each star separately does not characterise extension, and probabilistic sufficient tests are not characterisations either. The 21×6 triangle example is NO although all six stars are YES.	universal (assumes some external theorems)

The wall document also lists four minimum counterexamples to its own working rules (at 4 and 5 variables) and a table of twelve counterexamples; they are part of the working notes.

The shared-column gadgets and how far they got

A few of the attempts produced true, if limited, theorems (all PROVED-written):

- Equality stars. Polynomial-size binary strength-3 tables can force arbitrarily many occurrence copies of a variable to carry equal bits in any extension (a 13×3 star has 534 extensions). A single reference shared by two disjoint stars fails strength 3.
- Shared flags. Uniform shared flags enforce not-all-equal on all variables of a clause set (exactly the $2^n - 2$ non-constant assignments): correct, but the wrong relation for encoding clauses one at a time.
- Supplied gadget data. If gadget data satisfying a list of equations existed, the construction would be exact. The equations could not be satisfied in the cases tried (finite refutations, 607,275 search nodes).
- Fitted and uniform recipes. A fitted compiler gave exact projections on eight sources but is not a polynomial-time reduction; uniform clause recipes passed 1 of 28 cases; one more orientation recipe repaired three clause sets but was exact on only 6 of 28. No second recipe was tried.

What came next

The route that worked (chapter 9) put many classes on each column (a 'star' of hub links) rather than a private pair of columns per clause. Wall 3 and the transfer lemma explain why private pairs fail: each such column would have to be a near-solution of the whole source problem, so the reduction would have to solve the puzzle before it began.

The eighteen walls are a map of failures at the stated scope. They are not a proof that no reduction exists, and they do not contradict chapter 9; they indicate which designs to avoid, and further research on other designs is recommended.

Check it yourself

The walls are written results; the register of results lists each with how it was checked. The worked tables in the fixtures folder can be tried with recipe 4 of the check guide.

Where to go next

- Which route worked? Chapter 9: strength three is NP-complete.
- Can strength 3 be turned into duality after all? Chapter 12: the Mirror Route and the Ladder.

10 Strength three is NP-complete, and so is every higher fixed strength

Strength three is NP-complete, and so is every higher fixed strength

The question

Is there a fast way to decide whether a finished strength-3 table can take one more column, or is the problem NP-complete?

Why it matters

NP-completeness means that no fast exact method is expected to work for every table.

What answering it would help with

It tells planners to choose search or approximate methods instead of hunting for an exact fast rule.

What our research shows

Yes: deciding whether a strength-3 table can take one more column is NP-complete (a written proof; Lean checks that the construction is correct), and so is every higher fixed strength (the raising lemma that lifts the problem

from one strength to the next is proved in Lean; putting it together with the hub-row reduction to get NP-completeness is a written proof).

This chapter states the covering-array results of the hardness family. They are results about one decision problem on tables. They are not about any other question.

The problem

One-column extension at strength t. Input: a binary covering array of strength t. Question: can one more column be appended, keeping every row and adding none, so that the result still has strength t?

A problem is NP-complete if (a) a proposed answer can be checked quickly (here: a proposed column is checked in time $O(N\,k^2)$), and (b) every problem of that checkable kind can be turned into it by a polynomial-time translation. No polynomial-time algorithm is known for any NP-complete problem. Chapter 6 described how the strength-2 case is quasi-polynomial and, assuming the Exponential Time Hypothesis, not NP-hard. The results below say that at strength 3 the situation changes.

The results

Result	**Statement**	**Status**
The hub-row reduction	One-column extension of binary strength-3 covering arrays is NP-complete.	PROVED-written; the correctness of the construction is Lean-checked (PROVED-Lean for that part); polynomial time is written, not formalised
The raising lemma	Raising lemma: from a table M of strength t one builds a table M' of strength t+1 that extends if and only if M extends. Hence one-column extension is NP-complete for every fixed strength $t \geq 3$.	PROVED-Lean (raising lemma and size bounds). That the problem is NP-complete at every fixed strength, by composing the raising lemma with the hub-row reduction, is PROVED-written
Distinct rows	The problem stays NP-complete when all rows of the table are different.	PROVED-written
Thin tables	It stays NP-complete when the table is nearly square (about as many rows as columns), and with all rows different, at every fixed strength.	PROVED-written

The construction in plain words

The source problem is positive 3-uniform not-all-equal satisfiability: given m clauses of three variables each, can every variable be set to 0 or 1 so that no clause is all 0 or all 1? (Equivalently: can a set system whose members have three elements be two-coloured so that no member is a single colour? This problem is NP-complete by a theorem of Schaefer (1978), which we use as given.)

From the clauses, the construction writes a table with $k = 4m + 2$ columns and about m^3 rows. Each clause owns four columns. Some pairs of columns are designated; one row for every column (a hub row) marks which designated pairs it carries; one row for every variable marks the clauses it appears in; and, for every set Y of at most three columns containing no designated pair, a padding row that is 0 exactly on Y is written twice. The point of the padding is to make every group of three columns show all eight patterns, whatever the clauses are. Two things are then true for every input:

1. the table has strength 3 (proved in Lean), and
2. the table can take a new column if and only if the clauses can be satisfied (proved in Lean).

If the input is malformed, the translation outputs a fixed table with no room (the odd-parity 8×4 table: the eight words of length 4 with an odd number of ones, the complement of the parity table of chapter 5). If there are no clauses, it outputs the 3-cube, which has room.

The main idea, which the failed attempts of chapter 8 led to, is that a new column is a colouring of the rows. A satisfying assignment gives a colouring that splits every pair class, and the other way round. Each column carries a star of hub links, so that it need only flip one hub row; it does not have to be a near-solution of the whole source problem.

The raising lemma

Let M have strength t. Add a column w. Keep the old rows with $w = 0$. Add every row with at most $t + 1$ zeros with $w = 1$, each written twice. Then M' has strength $t + 1$, and M' can take a new column at strength $t + 1$ exactly when M can at strength t. Applying it $t - 3$ times to the output of the hub-

row reduction gives hardness at strength t. The padding grows like k^(t+1): polynomial for a fixed strength, which is all that is claimed. Nothing is claimed for strength growing with the input.

Distinct rows and thin tables

The padding rows of the hub-row reduction are written twice, so rows repeat. The distinct-row version writes each padding row once and colours it by the parity of its number of zeros (a parity lemma), removes twin variables (variables that appear in exactly the same clauses) before building, and outputs a fixed small table when at most one clause survives. The thin-table version shrinks the padding from about m^3 rows to O(log m), using a small three-symbol covering array, so the hard tables have about as many rows as columns.

How it was checked

- Lean proves, for every input, that the output of the hub-row reduction has strength 3 and extends exactly when the source is satisfiable, and proves the raising lemma and its size bounds. The files for the raising lemma are part of the 48-theorem package shipped in this book. The files for the hub-row reduction are in a separate hardness folder. What Lean does not cover: the running time of the translation (written in the notes, not formalised); the NP-completeness of the source problem (taken from Schaefer); and whether the Lean definitions match the plain statements above (read them to judge).

- Computation. For the hub-row reduction, 1,359 small source puzzles were run through the construction and compared with brute-force satisfiability in the working notes; 36 are unsatisfiable; there is no disagreement. The hub-row folder ships the written proof and the written polynomial-time argument. The check is a recipe anyone can follow: rebuild the table from a source by the construction in the proof, test strength 3 directly, search for a new column by brute force, and compare with brute-force satisfiability. It was rerun on all 1,024 clause families on five variables and 11 named sources (the Fano plane among them), with agreement throughout. A proof rebuilt from the theorem

statement alone found no error; that is a check of the argument, not an outside referee's report. For the raising lemma and the distinct-row version, both were rebuilt from the written proofs and checked: 202 small raising cases and 19 full strength-four reductions; and all 1,359 sources for the distinct-row version.

- The thin-table version was read line by line in a separate reading; computer checks agree with brute force on more than 2,000 small puzzles. It has not yet been rechecked a second time.

Prior art and credit

The decision problem is posed in the literature as the decisional parameter extension problem (Garn and Simos 2017, Problem 6; Garn's 2019 thesis, Problem 5.1.6; Kampel's thesis and the Kampel-Simos survey repeat the search version), with the complexity left open there. In the sources we checked (the Kampel-Simos survey, Kampel's thesis and Garn's thesis, read in the working notes) we found no statement that it is NP-complete. Seroussi and Bshouty (1988) remains unread; priority depends on completing that literature check (frontier not yet checked). The pair-class criterion that the proofs use is in Kokkala (2017) and Hiess (2024). The equivalence between detecting arrays and covering arrays of one more strength is in Colbourn and McClary (2008) and Shi, Tang and Yin; our part concerns the extension problem only.

What this suggests

These theorems indicate that, for tables of strength 3 or more, deciding whether a finished table can take one more column is a hard problem in the usual sense, in contrast with strength 2. Further research is recommended on the cases not covered: alphabets with four or more symbols, strength growing with the input, and approximation versions. Nothing here is a claim about any other question.

Lean sources and status

The Lean source files for the hub-row reduction are in the hardness folder; those for the other results of this book that are not in the 48-theorem package

are in the rules folder. They were rebuilt for this edition with Lean 4.34.1 against the pinned Mathlib revision: every file compiled without errors and without admitted steps, and every listed theorem depends only on the standard axioms (some on none). That is a check by the Lean compiler on one machine. Lean's own replay checker (leanchecker) also passed on both folders, and the second kernel checker nanoda 0.4.26 accepted exports of the listed declarations (hardness folder: 12 declarations, 5,195 declarations checked; rules folder: 29 declarations, 7,503 declarations checked). Unlike the 48-theorem package, these files have not had a fresh-directory build with a pinned dependency cache, and no outside reader has looked at them. The Check it yourself guide gives the commands. Two comment lines in the header of the main hardness file, and one comment line of the rules file on the repair clique, were edited; no other byte differs from the project files. After the last edit both folders were compiled again with the same results, and leanchecker passed again.

Check it yourself

Recipe 8 of the check guide compares the construction with brute force on small sources; recipe 10 builds the hardness Lean folder and recipe 9 the package with the raising lemma.

Where to go next

- What exact rules hold for strength four? Chapter 10: strength four rules and bounds.
- What happens with three settings per switch? Chapter 13: other kinds of tables.

11 Strength four: rules, growth and what limits them

Strength four: rules, growth and what limits them

The question

What exact rules govern binary arrays of strength four, how far can they be pushed by adding a column or removing rows, and what limits them?

Why it matters

These rules are how strength-four arrays are checked, extended and trimmed, and they show where the limits are.

What answering it would help with

Others can use them to build or trim arrays at other widths, and they mark the gap between the arrays and the lower bounds.

What our research shows

Several exact rules hold for strength four, growth by cloning works but costs many rows per column, and published inequalities leave a gap of 98 to 120 rows to the arrays we supply.

Chapters 1 to 4 described 29 arrays of strength four, and chapter 5 says how they were made. This chapter gives the exact rules that apply to such arrays, ways of growing an array that cost too much, and the lower bounds.

Exact rules

Each rule below is a statement about all binary arrays of strength four.

The support rule for one more column (proved in Lean). A new column keeps strength four exactly when, for every choice of three old columns and every one of the eight patterns on them, the rows showing that pattern include a row where the new column is 0 and a row where it is 1. This is the test used to add each column to the 141×199 starting table, and the one a 346th column would have to pass (chapter 11).

The completion rule for extra rows (proved in Lean, for the set of missing patterns). Suppose you keep a block of rows and want to add r more rows to make a strength-four array. List the four-column patterns that no kept row shows. Join two missing patterns when they ask for opposite bits in some column (a single row cannot show both). Then r new rows suffice exactly when this conflict graph can be coloured with r colours so that joined patterns get different colours. Each colour class becomes one row. The rule is exact in both directions. The same rule, in graph form, is the colouring form of fixed-row replacement. (Rows of the new block need not be filled in outside the patterns they carry.) Lean proves the rule for the set of missing patterns; the step from an array file to that set is written, not Lean, and was tested on 960 exhaustive comparisons over 240 seeded systems, all agreeing.

The block rule. For a block of new columns, the conditions to check include every choice of j new and $4 - j$ old columns for $j = 1, 2, 3, 4$. Checking $j = 1$ alone is not enough. PROVED-written.

Projection (proved in Lean). Keeping any set of columns of a strength-four array keeps strength four; permuting columns and flipping a column's 0s and 1s also keep it. Lean proves the restriction and the permutation (the projection theorem) and, separately, the flipping (a theorem of the 48-theorem package). This is why an array with more columns gives arrays with fewer: every comparison with the pinned table uses this rule.

Width for a given number of rows (proved in Lean). A binary array of strength four with N rows and at least four columns has at most $2^{(N-1)}$ columns. This is far from the sizes in this book, and no sharper bound is claimed.

A tempting invariant that fails. One might hope that if each new column splits every triple of the original array, adding such columns one at a time stays valid. It does not: on the 16-row table of all patterns of four switches, adding the parity column and then the same parity column again gives two columns that each pass the test against the original columns but never differ. The exhaustive check of all 65,536 candidate columns of that table finds exactly two that extend it: parity and its complement.

Growth works but costs too much

Can a strength-four array always be grown by one column? The answer in the cases computed is yes, at a price.

- Cloning a column (proved in Lean). Copy column 1, then add rows that repair the patterns the copy cannot show. With $m = \lceil \log_2(k-1) \rceil$, a k-column array with N rows becomes a (k+1)-column array with $N + 4m + 4$ rows. Lean theorems prove the construction and these counts. At width 345, $m = 9$, so cloning a 170×344 array gives 210 rows, 40 more than the 170×345 array supplied here.
- Repair rows. Copying a column needs at least eight repair rows (proved in Lean). Complementing a column needs at least eight as well; that follows from the copy case by flipping the column, and is a written argument (PROVED-written), not Lean.
- No universal one-row extension (proved in Lean). There is no universal one-column, one-row extension: the 16 even-parity rows on five bits cannot take a column plus one row.

- A full linear block (proved in Lean). A binary linear block in which every four columns are independent needs at least 65,536 rows at width 345; the pinned bound there is 174.
- Affine blocks. Twelve affine blocks give a 170-row block at width 345 that needs at least 4 repair rows (one four-column set misses four words that no single row can repair together); the construction tried used 58, for 228 rows. Neither count reaches the 170 rows of the array supplied at width 345.

These results indicate that growth by cloning, one column at a time, cannot keep up with the rows of the arrays supplied here; whether another growth rule could is not decided, and further research is recommended.

Removing rows

For 23 of the 29 arrays, rows were removed one at a time and each removal was repaired by changing cells (chapter 5). This row-removal-and-repair step follows the post-optimization approach of Nayeri, Colbourn and Konjevod

(2009; 2013): adjust a covering array to reduce its rows while keeping full coverage (Peyman Nayeri, Charles Colbourn and Goran Konjevod, Randomized postoptimization of covering arrays, IWOCA 2009, Lecture Notes in Computer Science, doi:10.1007/978-3-642-10217-2_40; journal version, Randomized post-optimization of covering arrays, European Journal of Combinatorics 34 (2013) 91-103, doi:10.1016/j.ejc.2012.07.017). The comparison table this book links lists post-optimization constructions for many of its rows (postop NCK, postop TJ-SA). No priority is claimed for this step. The 145 × 206 array, and the arrays at widths 202 to 205 cut from it, have 3 rows fewer than the 148 × 206 table grown from Smolen's array; the arrays at width 223 and at widths 329 to 345 have 26 and 9 rows fewer than the 179-row half-Paley array. The arrays at widths 200, 201 and 325 to 328 had no rows removed. Whether more rows can be removed at any width is not known.

For the narrowest case, replacing two removed rows by one row, the one-row replacement criterion is exact: it succeeds exactly when the missing patterns, taken from the kept rows, never ask for opposite bits in the same column. The Check it yourself guide says how to test any pair of rows of any array in this book from the array bytes; the test takes about as long as the full check of one array. A failure is narrow: these two rows cannot become one. It does not say that a smaller array is impossible. A further result adds that the simple shortcut of looking for a clique of mutually conflicting patterns has at most 15 vertices when at least one row is kept, so a clique alone cannot rule out large replacements.

Lower bounds

Published inequalities. From the published inequalities (Kleitman and Spencer 1973; Roux 1987; Choi, Kim and Oh 2011 and 2012; Rényi; Katona), exact computations give: at least 47 rows at widths 201 to 203 and at least 50 rows at widths 325 to 342. (The ordinary fractional bound alone gives just 16 at width 201.) The arrays supplied at those widths have 145 rows (201 to 203) and 166 to 170 rows (325 to 342), so the gap is 98 to 120 rows. No lower bound was computed at widths 200, 204 to 206, 223 or 343 to 345. The

smallest possible numbers of rows are not known. These are calculations from known inequalities, not a new method.

Other facts

- Support antichain. Distinct triple-pattern supports of a valid array form an antichain, so the clause set for a one-column extension has no repeated or redundant clauses.
- Strength five. Five attempts at a binary strength-five array with 16 columns and 97 rows ended with 3 to 7 patterns still missing.

Credit

The starting tables, the table and the bounds are other people's work: R. E. A. C. Paley (the 1933 construction), Christopher Smolen (the 141 × 199 and 165 × 324 starting tables, CC BY 4.0), Kokkala and Hiess (the column criterion), Colbourn, Groemping, Sloane, Sherwood, Martirosyan and Colbourn, Walker and Colbourn, Colbourn and Lanus, and Hyun, Mondal and Lee (for the code constructions compared above). See the credits page.

Check it yourself

Recipe 5 of the check guide tests the one-row replacement for any pair of rows; recipes 9 and 10 build the Lean files behind the Lean-proved rules.

Where to go next

- Can the widest array take one more column? Chapter 11: the 346th column.
- What did not work? Chapter 16: the ledger.

12 The 346th column

The 346th column

The question

Can the widest array of the main collection take a 346th column?

Why it matters

Whether the widest array can grow by one more column tests how close it is to a limit.

What answering it would help with

A search whose power has been tested may settle it; the rule here says exactly what such a search must find.

What our research shows

It is open: no search for a 346th column has been run on the 170 × 345 array supplied here, and this chapter gives the exact rule such a column must meet.

The widest array of strength four in the main collection has 345 columns and 170 rows (the pinned table's bound is 174 rows there, and the smallest size in Smolen's living table at that width or wider is also 174). The question is whether one more column can be added while keeping the 170 rows.

What would be asked

By the support rule of chapter 10, a 346th column exists exactly when the 170 rows can be coloured 0/1 so that, for each of the 6,784,540 choices of three old columns and each of the 16 patterns on the three old columns and the new one, a row shows that pattern. That is 108,552,640 conditions, which together form a colouring puzzle of the kind in chapter 5.

Because the 170 × 345 array is columns 1 to 345 of the 179-row half-Paley array with 9 rows removed and repaired, one natural place to start is column 346 of the half-Paley array, restricted to the 170 rows that remain; whether it, or any column, passes the rule has not been checked. This row-removal-and-repair step follows the post-optimization approach of Nayeri, Colbourn and Konjevod (2009; 2013): adjust a covering array to reduce its rows while keeping full coverage (Peyman Nayeri, Charles Colbourn and Goran Konjevod, Randomized postoptimization of covering arrays, IWOCA 2009, Lecture Notes in Computer Science, doi:10.1007/978-3-642-10217-2_40; journal version, Randomized post-optimization of covering arrays, European

Journal of Combinatorics 34 (2013) 91-103, doi:10.1016/j.ejc.2012.07.017). The comparison table this book links lists post-optimization constructions for many of its rows (postop NCK, postop TJ-SA). No priority is claimed for this step.

A way to split the puzzle

We studied splitting the colouring puzzle into blocks of rows that can be checked separately. Four statements hold for any array: a sufficient condition, an exact one (a column extends exactly when every block passes), the fact that any forced edge blocks the extension, and monotonicity (enlarging the set of fixed rows keeps residual edges residual). These are proved on paper. Whether such splits are small enough to help on this array is not known.

What it indicates

Nothing yet about whether a 346th column exists. We recommend a search whose power has been tested beforehand on smaller arrays where the answer is known.

Checks

The 345-column, 170-row array passes the full four-column check by both programs (chapter 4). The rule above is the support rule, proved in Lean (chapter 10).

Check it yourself

Recipe 2 of the check guide checks the 345-column array, and a proposed 346th column can be checked the same way by appending it to the array file; the register of results says how each result was checked.

Where to go next

- What rules govern strength four? Chapter 10: strength four rules and bounds.
- Which questions are still open? Chapter 16: the ledger.

13 The Mirror Route and the Ladder

The Mirror Route and the Ladder

The question

Can the strength-3 question be turned into a duality question by a short, honest translation, and how small could that translation be?

Why it matters

If strength-3 extension had a short translation into duality, it would sit closer to strength 2 than chapter 9 suggests.

What answering it would help with

The Ladder gives fixed test tables for anyone who tries another translation.

What our research shows

No general translation from strength 3 to duality was found; the exact smallest sizes are known for 8, 10 and 14 rows and are open for 12 and 16.

Chapter 6 described how, at strength 2, one-column extension is the same question as a hitting-set puzzle (monotone duality). Chapter 9 states that at strength 3 it is NP-complete. The Mirror Route is an attempt in between: to turn the strength-3 question into a duality question by a short, honest translation. It did not succeed in general. It produced some theorems, a scoreboard called the Ladder, and several failed rules, all set out here.

The idea

A new column of a strength-3 table is a colouring of its rows. If the table is small, one can write each colouring as a low-degree formula in the old columns. A formula of degree at most 2 has a list of coefficient bits, and a column is valid exactly when its coefficient bits satisfy a set of conditions. Call the set of valid coefficient settings the gaps. A mirror pair is two families of constraints, F (clauses that must contain a 1) and G (clauses that must

contain a 0), cross-intersecting, such that the gaps are exactly the valid columns and nothing else. The question is how small F and G together can be: $|F| + |G|$. If that size could be bounded by a polynomial in the table's size, strength-3 extension would reduce to duality. The Ladder measures this on a fixed series of tables.

The Ladder

For tables with 8, 10, 12, 14 and 16 rows (3, 4, 5, 6, 6 old columns; 7, 11, 16, 22, 22 coefficient bits; 2, 2, 22, 6, 102 valid colourings):

Rows	Smallest found $\|F\| + \|G\|$	Lower bound not depending on the four-isolated-models theorem	Best pair found (F / G)
8	6 (exact)	6	3 / 3
10	10 (exact)	10	7 / 3
12	70	16 (17 if that theorem holds)	27 / 43
14	24 (exact)	24	15 / 9
16	654	22 (23 if that theorem holds)	304 / 350

Status: COMPUTED. The displayed pairs were checked over the whole coefficient cubes (4,194,304 settings on the larger rungs). The exact minima for 8 and 10 rows and the displayed bounds for the others were rederived a second time. Only 369 of the 10,635 listed file fingerprints were rechecked. The note states that no growth law is claimed; the table is a scoreboard. The 12-row and 16-row minima are known only to lie between 16 and 70, and between 22 and 654 (the lower ends are 17 and 23 if the four-isolated-models theorem holds); the 14-row minimum was at one stage known only to lie in 23 to 24, and was then settled at 24.

Theorems on the way

- Critical clauses. If a valid colouring has p rigid one-bits and q rigid zero-bits (flipping one coefficient breaks it), then $|F|$ is at least p and $|G|$ at least q. This is the 'critical clause' idea of Paturi, Pudlák and Zane (1997). If the pair has exactly $d - 1$ edges and all non-constant directions are rigid, the constant coefficient plays no part.

- The four-isolated-models theorem (CONDITIONAL). A positive/negative monotone CNF on n variables with at most $n + 1$ clauses and cross-intersecting positive and negative supports has at most four isolated models (models whose every one-bit change breaks them); if it has four, their bitwise XOR is zero, so they form a translate of a 2-dimensional space. The bound is sharp (a six-variable, seven-clause example). If it holds, it gives Ladder lower bounds 17 and 23 for 12 and 16 rows (the 14-row value is settled by the exact 14-row minimum). The bounds 16 and 22 that do not depend on it come from an argument about the number of valid colourings and a weaker cross bound, which are in the ledger of chapter 16. Why conditional: the proof is computer-assisted. The theorem was confirmed by exhaustive enumeration for n up to 6 (for $n = 6$, 26,324,608,236 clause sets) and by fresh encodings up to $n = 12$. But eight of the twelve original ten-variable proof files delete unit clauses in the middle of the proof, which a strict proof checker rejects. They need to be regenerated without deletions before the theorem is cited as certified, and we do not have them. We therefore label it CONDITIONAL, not PROVED-written. The maximum number of isolated models $f(n,1) = 4$ for n up to 5 is COMPUTED.

- Fixed coordinates. After fixing the coordinates every valid column agrees on, the minimum is $p + q$ plus the minimum on the remaining coordinates.
- Critical cover. Replacing each positive clause by a contained minimal transversal gives lower bounds from an optimal cover of the critical demands.
- The 14-row minimum (COMPUTED, by solver certificates). For the 14-row table with its six old columns and its 22-bit degree-2 decoder, all six valid colourings kept, the smallest total number of constraints is 24: 15 in F and 9 in G. A restricted form (one pattern per complementary pair) was settled earlier by an exhaustive search over 256^3 triples; the general case was then settled by 256 cases, each split three ways, all unsatisfiable, with 256 strict refutation proofs covering the 768 sub-

cases, and a 24-edge witness exhausted over its 4,194,304 settings. Of the 768 sub-proofs, 553 were accepted by two checkers and 215 by one checker only; 7 checks were redone.

- Partial isolation. Models isolated in only n designated directions, with at most n + h clauses, differ in at most 3h + 2 coordinates. A reading found no mathematical gap (4 wording gaps), and the 24 does not depend on it.
- From a cross-intersecting pair to an intersecting family. A cross-intersecting pair becomes an intersecting family whose proper colourings match its gaps; a list of r patterns costs 1 + rd + C(r,2) edges. The cost is polynomial in the supplied list, not uniform in the table.
- Decoder failure. On the 16 distinct rows of the 4-cube, a valid new column can lie outside the fixed quadratic decoder's image: 7,842 valid columns, only 370 in the image. So 'keep every valid colouring with this decoder' is impossible in general. (You can reproduce 7,842 by the colouring rule of chapter 5 on the 16-row table of all patterns of four switches.)

- Witness selection. Witness selection results; the honest summary: a general polynomial reduction from strength-3 extension to non-duality was not established. What is known is the unit-row subclass (chapter 7) and the lower bound |B(H_m)| ≥ 2^C(m−1,2). A self-dual selector construction and an auxiliary-slice lemma were found, but the attempt to complete the reduction did not close.

What it indicates

The Ladder and its theorems indicate that the Mirror Route has no cheap general form for these tables: the smallest pair found grows unevenly (6, 10, 70, 24, 654), and the exact values at 12 and 16 rows are open. The scope is the tested decoder. Still open: a short translation using a different decoder. Further research is recommended: regenerate the proofs behind the four-isolated-models theorem without unit deletions, close the 12- and 16-row rungs, and test other decoders.

What is checked and what is not

The exact 14-row minimum and the Ladder table were rechecked a second time; the four-isolated-models theorem is CONDITIONAL until its proofs are regenerated; the other theorems of this chapter are PROVED-written (read, and in part computed). Nothing in this chapter has Lean support. The files behind the exhaustive checks are not shipped in this edition; the Check it yourself guide says so.

Check it yourself

The files behind the exhaustive checks are not shipped; the register of results says how each result was checked. The count of 7,842 can be reproduced with recipe 4 of the check guide.

Where to go next

- What did not hold, or is still open, on the Ladder? Chapter 16: the ledger.
- Why is a general translation unlikely? Chapter 9: strength three is NP-complete.

14 Other kinds of tables: three symbols

Other kinds of tables: three symbols

The question

Does the picture change when each switch has three settings instead of two? And how many rows does a table need when it must also locate a faulty pair of settings?

Why it matters

Many test plans use switches with more than two settings.

What answering it would help with

It sorts the kinds of tables by whether they admit fast rules for adding a column.

What our research shows

Yes: with three settings per switch, strength-2 extension is NP-complete (a written proof, rebuilt and checked on small cases; no Lean proof). For the locating arrays below, the 13 by 13 table is printed so that you can check it (COMPUTED), and that 13 rows is the minimum for 13 columns is CONJECTURED: one solver found no 12-row array in a SAT encoding of the problem; a second solver has re-checked a sample of the search, not the whole search, and the search is not part of this book.

So far every switch had two settings, 0 and 1. A covering array can have more. With three settings per switch ('ternary'), strength 2 means: for any two switches, all nine pairs of settings occur.

The result

Ternary strength 2. Deciding whether a ternary strength-2 covering array can take one more column, keeping all rows and adding none, is NP-complete. (Status: PROVED-written.)

Compare this with chapter 6: for two settings per switch, strength 2 is the monotone-duality question and is not expected to be NP-hard (CONDITIONAL). With three settings the same strength becomes hard. That is a statement about one decision problem, computed for one alphabet size.

How it was checked

The result was rebuilt from the written proof alone and checked: 7,404 small source puzzles were run against two complete solvers, 71 of them with the answer no, with no disagreement. Rows can repeat among certain edge classes when twin vertices occur, and the argument does not use distinct rows. Nothing here has a Lean proof.

Prior art

The problem is the ternary case of the parameter extension problem of chapter 9 (Garn and Simos 2017, Problem 6). In the sources we checked we found no statement of this result; as for chapter 9, we did not read all the relevant sources, and we claim no priority.

Locating arrays: how many rows do 13 switches need?

Picture a machine with 13 switches, each on or off, and one pair of switch settings that is faulty, say switch 3 on together with switch 9 off. A test run is a row of the table, and a run fails when the faulty pair shows in it. The table is a (1-bar,2)-locating array when the failing runs tell you which pair is at fault. In symbols: each of the $4 \times 78 = 312$ pairs of settings on two switches shows in at least one row, and different pairs show in different sets of rows.

What we found. A computer search found a table with 13 rows and 13 columns that is a (1-bar,2)-locating array (COMPUTED: found by search, then checked against the definition by two separately written programs). A second search, for a table with 12 rows and 13 columns, found none: the search used a SAT encoding of the definition, with row 0 fixed to zero by complementing columns and rows and columns each sorted in lexicographic order (double-lex symmetry breaking). The search tree was cut into cubes. In the receipt of the run, the solver CaDiCaL 1.9.5 answered UNSAT on 601,831 distinct cubes and SAT on none. Of these, 552,691 are the terminal cubes that close the tree, and the other 49,140 lie below a cube already answered UNSAT. A completeness check over the recorded cubes found every branch of the tree covered by the terminal cubes, with no gap. That 13 rows is the minimum for 13 columns is therefore CONJECTURED: one solver found no 12-row array in a SAT encoding of the problem; a second solver has re-checked a sample of the search, not the whole search, and the search is not part of this book. If the search is complete and the encoding and sorting rules are right, the minimum is 13. The sorting rules lose no tables by a written argument that the solver does not check.

Checks. A second solver, glucose 4, replayed a seeded random sample of 300 cubes and the 25 hardest: 307 returned UNSAT, 18 timed out, and none

disagreed. As an encoding control, the encoding accepts the 13 by 13 table below once it is put in canonical form, and rejects a copy with one bit flipped. Neither check replays the whole search with a second solver.

Smaller numbers of columns. With the same encoding, the smallest number of rows we found for k columns is 6, 7, 8, 9, 10, 11, 11, 11 and 11 for k = 3 to 11. COMPUTED. These agree with the minima that Konishi, Kojima, Nakagawa and Tsuchiya proved for 3 to 11 columns.

Prior art. Konishi, Kojima, Nakagawa and Tsuchiya (2019, journal version 2020) searched for minimum locating arrays with a CSP solver. For 13 columns their solver timed out at 12 and 13 rows and found 14, and for 12 columns it proved 12 the minimum. Their simulated-annealing paper (2019) also reports 14 for 13 columns, with an earlier SAT-based paper (Konishi and others, 2017) as the source of that value; we did not read the 2017 paper directly. We did not read Shi, Fu, Wang and Yan (2021), which gives upper bounds for small numbers of factors and might cover this case, so our literature check is incomplete. The tables we found in the literature we checked have 14 rows, and we claim no priority.

The 13 by 13 table, so that you can check it. Each line is a row; each column is a switch. The same table is in the file array-13x13.txt.

```
0000000000000
0000011111111
0001100001111
0010100110011
0110101010101
0110110101010
1001010011010
1100111010010
1101000111001
1101001100110
1110001001111
1110010110100
1111110000001
```

To check it, list the 312 pairs (switch i, switch j, value a, value b) with i < j. For each, collect the set of rows that show value a in switch i and value b in switch j. All 312 sets are non-empty and all 312 are different. Flip any single bit of the table and the check fails, for each of the 169 positions. Recipe 11 of the check guide gives the condition and the expected answer; a few lines of your own code are enough. The search itself is not shipped.

What is not in this edition

Other table problems, apart from the locating array above, are not part of this edition of the book. This chapter is deliberately short.

What it suggests

The change from 'not expected to be NP-hard' at two settings to NP-complete at three settings indicates that the number of settings, not just the strength, changes how hard the extension problem is. Further research is recommended for four or more settings, which are not covered here.

Check it yourself

The ternary result is a written proof; the register of results says how it was checked, and no files for it are shipped. The locating-array table is checked with recipe 11 of the check guide, by hand or with a few lines of your own code. The search for a 12-row table is not shipped.

Where to go next

- Which other arrays were found? Chapter 14: more arrays.
- What is still open? Chapter 16: the ledger.

15 More arrays: small exact cases

More arrays: small exact cases

The question

Beyond the 29 main arrays, which small exact cases were found and checked?

Why it matters

Exact small cases show where published tables can be matched and where a question is still open.

What answering it would help with

Others may reuse the arrays, or compare them with tables they hold.

What our research shows

Of the small exact cases, three are computed results (two arrays and one SAT proof) and two cases stay open.

The 29 strength-four arrays of chapters 1 to 4 are the main collection. This chapter adds small exact cases that were also made and checked, each shipped in the book with its fingerprint in the fingerprint list.

Small exact cases

- A 22-row array with cyclic symmetry. A 22-row, 9-column array with 4 symbols and strength 2, found by assuming that the array is unchanged when its columns are shifted cyclically. The rows are cyclic shifts of three short rows, plus a row of all 3s. It matches the 22 rows in the tables, whose construction the package could not locate. Table 3 of Kokkala, Meagher, Naserasr, Nurmela, Östergård and Stevens (arXiv:1901.03594) gives 22 as a lower bound for uniform arrays with 8 and 9 columns and 4 symbols; whether a non-uniform 21-row array exists is open.
- A non-extendible array. A 12-row, 6-column binary strength-3 array that cannot take a seventh column (the criterion of chapter 5 gives 0 extensions). It is already known: it is, up to the permutation of columns, the complementing of columns and the order of rows, the representative 7 of the listing of Colbourn, Keri, Rivas Soriano and Schlage-Puchta (Discrete Applied Mathematics 158, 2010). We include it as a rediscovery, to credit that listing.

- Two constant rows. No binary strength-3 covering array with 12 rows and 9 columns has two constant rows (one all 0s and one all 1s). The CNF formula encodes the question with a standard symmetry-breaking order on the rows and columns; the solver CaDiCaL 2.1.2 reported that it is unsatisfiable, and the proof file was checked by drat-trim with the verdict s VERIFIED. The shipped formula has SHA-256 da9940c154e5335b94680645594f0d4c79184dec7b538bc21531c3cbe4b40ad3. drat-trim (Heule, Hunt and Wetzler) is obtained from its public repository; we did not ship it. No one else has rechecked this proof yet, and the symmetry-breaking order of the encoding is the one assumption a reader should look at.
- Open cases. Two cases stay open: whether a CA(15;2,21,3) exists (the smallest array known has 16 rows) and whether a CA(23;2,10,4) exists (the smallest array known has 24 rows). 125 symmetry types for the former were shown impossible, and 39 timeouts say nothing. All the 'unsatisfiable' answers rest on a solver's report.

Credit

These cases rest on other people's tables and tools: Colbourn, Groemping, Kokkala and co-authors, Colbourn, Keri, Rivas Soriano and Schlage-Puchta, and the SAT-solver authors Biere (CaDiCaL) and Heule, Hunt and Wetzler (drat-trim). See the credits page. The licences are as on the licence page.

Check it yourself

Recipe 3 of the check guide checks the arrays; recipe 6 checks the SAT proof with drat-trim.

Where to go next

- Are there more arrays at strength 2? Chapter 15: the strength-two collection.
- What did not hold? Chapter 16: the ledger.

16 The strength-2 collection: a summary

The strength-2 collection: a summary

The question

For strength 2 and alphabets of 3 to 18 symbols, at which widths are there arrays with fewer rows than the published table lists?

Why it matters

Strength-2 arrays are the most widely used covering arrays in test design.

What answering it would help with

Others can compare the listing with their own tables, and the plateaus show where the published table steps up.

What our research shows

In the sources we located, 124 of 1,631 arrays in the later collection and 69 of 720 in the older collection have fewer rows than the November 2024 table lists.

This chapter is a summary. It describes a large collection of strength-2 covering arrays (alphabets of 3 to 18 symbols). The 22 arrays that are the widest of each plateau are shipped in full in the strength-2 folder of the certificates folder, and a listing of every array with its fingerprint is in the strength-2 listing table of the statement folder. The other arrays are not shipped in this edition: the collection is large (about 180 megabytes with the working files), and the arrays other than the plateau tips are available on request. The chapter says which arrays have been fully rechecked and which have not.

What was found

A covering array of strength 2 is a table in which every pair of columns shows every pair of symbols. The published tables of upper bounds for the smallest

such tables (Colbourn, November 2024) give, for each number of symbols and each number of columns, a number of rows. The table has steps: the number of rows rises at certain widths. Where the base array is known, the arrays in this collection extend arrays built by the CAs package from published constructions. For the older collection the base construction was not kept (the listing says "not known" on 722 lines), and the origin of the alphabet-7 pair is unknown. We do not present any of them as new constructions.

- Size of the collection. 2,339 strength-2 arrays are described here (706 in the older collection, 1,631 in the later collection, and the alphabet-7 pair). The older collection also holds 14 arrays of strength 3, which are not in this count.
- Plateaus. Each below-table block starts exactly where the table steps up; the same row count holds for a few more columns. There are 20 plateaus and 195 below-table array cells in the collection (the alphabet-7 pair, below, is counted as two). A reason for this is not established here; further research is recommended.
- Three plateaus extended. Of the 13 plateaus tried (every plateau of the older collection except the 4-symbol, 42-row one), three were extended by one more column: with 5 symbols, 53 to 54 columns at 53 rows, 58 to 59 at 55 rows, and 221 to 222 at 65 rows. Six arrays were rechecked.

- A negative result. With a balance condition on the symbol counts of the new column, no extra column was found on any of the 13 plateau tips tried.
- A solver's 'no' is not a bound. A solver answer that no more columns fit is about one fixed array only. It is never a lower bound for the smallest array (see chapter 6).
- The comparison rule. Deleting columns from an array gives smaller arrays, so a table's bound at a width is the smallest row count among entries at that width or wider. The comparison used here is with Colbourn's November 2024 table as preserved by Groemping and read through Smolen's CSV file.

The two collections

Collection	Arrays	Below the table	Equal	Above	Full recheck
Later collection	1,631	124	212	1,295	every array, every pair of columns (1,631 of 1,631 valid; 211,758,899 column pairs)
Older collection	720 (706 of strength 2, 14 of strength 3)	69	98	553	no: checked once with the verifier that made them, and a literature check for the 69
Alphabet-7 pair	2	2	0	0	every pair of columns, by a separate program in Node

The 1,631 arrays of the later collection are, by alphabet size: 4 symbols 545, 5 symbols 471, 6 symbols 462, 7 symbols 153. Their comparison with the table (the published table we compared with (C. J. Colbourn's covering-array tables, as preserved by U. Groemping and read through C. Smolen's CSV file) lists more rows than ours at the widths of 124 of them; for none of the compared arrays does it list the same number of rows or fewer) is a comparison bounded by the sources located, which does not certify the current global state of the art. The two collections share no (strength, symbols, rows, columns) cell. The older collection's 69 below-table arrays have no full second check.

The plateaus

Below-table arrays by number of symbols and rows. "Rows" is the number of rows of the arrays, and the table needs one or more rows more at those widths.

Symbols	Rows	Columns	Arrays	Table's rows	Collection	Full second check
3	24	200	1	25	older collection	no
4	42	149 to 160	12	43	older collection	no
4	52	653 to 674	22	53	later collection	yes
5	53	53 to 54	3	54	older collection	no
5	55	56 to 59	5	56	older collection	no
5	65	221 to 222	3	68	older collection	no
5	69	253 to 265	13	70	later collection	yes
5	73	302 to 319	18	74	later collection	yes
7	96	65	1	101	older collection	no
7	103	80 to 81	2	105	older collection	no
7	109	89 to 91	3	111	older collection	no
7	115	102 to 106	5	116	older collection	no
7	121	115 to 118	4	122	older collection	no
7	133	498 to 510	13	138	older collection	no
7	145	626 to 645	20	146	later collection	yes
7	151	694 to 718	25	152	later collection	yes
7	157	801 to 826	26	158	later collection	yes
7	163	917	1	164	alphabet-7 pair	yes
7	169	1076	1	170	alphabet-7 pair	yes
8	126	82	1	132	older collection	no
8	148	122 to 124	3	151	older collection	no
8	176	713 to 725	13	182	older collection	no

For example, with 7 symbols and 163 rows, an array with 917 columns exists, where the table needs 164 rows for 917 columns; with 4 symbols and 52 rows, arrays exist for 653 to 674 columns, where the table needs 53 rows.

The alphabet-7 pair

A 163-row, 917-column array and a 169-row, 1,076-column array, both with 7 symbols. The published table we compared with (C. J. Colbourn's covering-array tables, as preserved by U. Groemping and read through C. Smolen's

CSV file) lists 164 and 170 rows at these widths; the difference is one row each. Their construction method is not known, so we cannot describe how they were built. Both are shipped in the strength-2 folder of the certificates folder and were checked over all 419,986 and 578,350 pairs of columns.

What we rechecked for this book

We rechecked the widest array of each of the 20 plateaus, and the alphabet-7 pair (22 arrays in all), against the definition of strength 2 (every pair of columns shows every pair of symbols); all 22 pass. That does not replace a full recheck of the other arrays of the older collection, which we have not done.

Not resolved

A report from another group (Numaro Research Report NUMARO-2026-007, July 2026) states 40 improvements over Colbourn's tables. We did not check its 32 unlisted lines. Some of our below-table cells could be anticipated by it. We therefore describe our collection as a comparison with the sources we located, not as a claim that the cells are new.

What this suggests

The plateaus indicate that the published table is one step behind the arrays found, just below each step. Whether the same holds in tables for larger strengths is not settled here; chapter 10 gives the rules for strength four, and further research is recommended.

Check it yourself

Recipe 3 of the check guide checks the 22 shipped arrays; the strength-2 listing table in the statement folder gives the fingerprint of every array.

Where to go next

- Which ideas did not hold, and what is still open? Chapter 16: the ledger.
- What question about adding a column lies behind the plateaus? Chapter 5: room for one more column.

17 The ledger: what did not hold, and what is still open

The ledger: what did not hold, and what is still open

The question

Which ideas tested here did not hold, and which questions are still open?

Why it matters

Results are easier to judge when the ideas that did not hold are shown beside them.

What answering it would help with

It marks the dead ends for others to avoid and lists the open questions to pick up.

What our research shows

Some conjectures and rules did not hold and are set out here as findings, and five questions remain open.

A list of results is more useful when it includes the ones that did not hold. This chapter collects them. Every entry also appears in the register of results.

What did not hold, and bounds that still stand

What it was	What happened
Two special branches of the strength-2 normal form (a cell of size two; two identical rows).	Both are covered by the reduction to duality questions, which covers every prefix. The identical-row case needs an extra free element and must not be generalised without it.
Ladder lower bounds 16 and 22 (from an 'affine model set' argument).	The four-isolated-models theorem gives the stronger bounds 17 and 23. Since that theorem is CONDITIONAL, 16 and 22 remain the lower bounds that do not depend on it.
At most two isolated models in a cross-intersecting monotone formula.	The four-isolated-models theorem gives a stronger bound (at most four). Until that theorem's proofs are regenerated, the bounds this result gives (6, 10, 16, 22, 22) remain the unconditional Ladder lower bounds at 8, 10, 12 and 16 rows; at 14 rows its 22 is replaced by the exact minimum 24.
'The canonical boundary formula is the cheapest pair with a given gap set.'	False: for the gap set $\{3,12\}$ the canonical cost is 8, but $F = \{5,10\}$, $G = \{6,9\}$ costs 4.
Rules that were tried: one predicted 15 edges on 12 rows (the result was 153), one that a single extra clause suffices (refuted on all three rungs), and a source-aware swap rule (48 candidates were unsound).	Refuted. Finite computations only.
'A row-irreducible strength-3 array has no room for a new column.'	False: a 12-row, 4-column row-irreducible array extends (28 ways). Exhaustive over the 65,536 subsets of the 4-cube.

Negative results

Things that did not work:

- the eighteen walls (chapter 8), and the constructions that were tried and stopped (a fitted compiler, uniform clause recipes and an orientation recipe);
- growth by cloning a column, which works and costs many rows per column (chapter 10);
- a balance condition for the strength-2 plateaus (chapter 15);
- the Mirror Route (chapter 12): no general reduction was established, and the exact Ladder values at 12 and 16 rows are open.

CONDITIONAL results, and results whose checks are still partial

- The four-isolated-models theorem. CONDITIONAL until its computer-assisted proofs are regenerated without unit deletions (eight of the

twelve original ten-variable proofs delete unit clauses, which a strict checker rejects). The lemma itself was confirmed by exhaustive enumeration up to n = 6.

- Strength-2 extension is not NP-hard. CONDITIONAL on the Exponential Time Hypothesis.
- The exact 14-row minimum (COMPUTED). 553 of the 768 sub-proofs were checked by two checkers and 215 by one. The statement stands on both.
- The hub-row theorem (PROVED-written; the construction PROVED-Lean). The base theorem was rechecked: the construction was rerun on all 1,024 clause families on five variables and 11 named sources (1,035 in all) with no disagreement, and its proof was rebuilt from the statement alone (chapter 9). It also has a Lean proof of the construction's correctness. Earlier rechecks covered the raising lemma, the distinct-row version and the ternary case.

Still open

- Does a 346th column exist for the 170 x 345 array? (chapter 11)
- What are the exact Ladder minima at 12 and 16 rows? (chapter 12)
- Can more rows be removed at any of the 29 widths? (chapter 10)
- What is the smallest possible number of rows at any of the 29 widths? (Lower bounds were computed at 21 of them: 201 to 203 and 325 to 342; chapter 10.)
- Do CA(15;2,21,3) and CA(23;2,10,4) exist? (chapter 14)

What nothing in this book shows

No array here is claimed to be the smallest possible. A comparison with a table is a comparison with one dated snapshot and the sources we located. A rediscovered object (the non-extendible 12-row array) is credited to the earlier listing; the strength-3 criterion is known. The Lean proofs establish the formal statements and say nothing about whether those statements capture the intended meaning; running times are proved in writing only, not in Lean. Where a result suggests something beyond its exact statement, the chapter says it 'indicates' it and recommends further research.

Check it yourself

Every entry here is also in the register of results, with how it was checked.

Where to go next

- Does a 346th column exist? Chapter 11: the 346th column.
- What are the exact Ladder values at 12 and 16 rows? Chapter 12: the Mirror Route and the Ladder.

18 The 48 Lean theorems: plain statements

The 48 Lean theorems: plain statements

All 48 theorems are in the Lean package shipped in this book (the library ArraysLean). Status of every row: PROVED-Lean. Each was compiled with Lean 4.34.1 against the Mathlib revision pinned in the package's lockfile; each uses only the standard axioms (propext, Classical.choice, Quot.sound); and the exported proofs (6,174 declarations) were re-checked by a second kernel checker, nanoda. The checkers named for this package are therefore the Lean kernel, nanoda and Lean's own replay checker (leanchecker, passed); no lean4lean run is claimed. The exact types are in the theorem types page of the package.

Each row starts with the question the theorem answers, then the plain statement, then the register result it supports.

What Lean shows here: each formal statement about finite tables and set families follows from the definitions in the package. What it does not show: that those definitions match the plain statements below (read the definitions to judge that), anything about running time (polynomial time is proved in writing only, not in Lean; see the chapters), or anything beyond the formal statement. The last column names the register result whose formal part the theorem covers; 'supporting' means a step used by other theorems that has no register row of its own. The links between theorems and register results were assigned by hand from the theorem names and statements.

#	Question it answers	Plain statement	Register result it supports
1	When can a finished table of strength t take one more column without adding any row?	A new column keeps the array at its strength exactly when, for every group of columns one smaller than the strength and every pattern on them, the rows showing that pattern use both 0 and 1 in the new column.	Class lemma: extension is proper two-colouring of the row-class hypergraph (every strength); Support criterion for strength 4; Pair-class formulation
2	When can a finished table of strength t take one more column without adding any row?	An array can take one more column exactly when the rows can be coloured 0/1 so that no required group of rows is all one colour.	Class lemma: extension is proper two-colouring of the row-class hypergraph (every strength); Pair-class formulation
3	At strength four, what exactly must a new column satisfy so that the table keeps strength four?	At strength four: a new column keeps strength four exactly when, for every choice of three old columns and every pattern on them, both 0 and 1 occur in the new column among the rows showing that pattern.	Support criterion for strength 4
4	Why is the bound at a width the smallest row count among entries of that width or wider?	An array of a given strength also has every smaller strength (as long as it has enough columns).	Table lookup rule: dropping columns preserves coverage (supporting)
5	A step used by other theorems.	In an array of strength t, every pattern on any set of at most t columns occurs (under the stated size condition).	supporting lemma
6	How many columns can a binary strength-four array with N rows have at most?	After flipping columns so that a chosen row is all zeros, two different columns of a strength-two array stay different.	Width ceiling for a fixed number of rows; Normal form of a strength-2 prefix (supporting)
7	How many columns can a binary strength-four array with N rows have at most?	A strength-two binary array with N rows has at most 2^(N-1) columns.	Width ceiling for a fixed number of rows
8	How many columns can a binary strength-four array with N rows have at most?	The same bound, at most 2^(N-1) columns, holds for a binary array of strength four with at least four columns.	Width ceiling for a fixed number of rows
9	Does the hardness hold at every fixed strength from 3 up?	The raising construction (add one column, copy the rows under 0, and add copies of rows with few zeros under 1) gives an array of one higher strength.	Raising lemma: every fixed strength t >= 3

#	Question it answers	Plain statement	Register result it supports
10	Does the hardness hold at every fixed strength from 3 up?	The raised array can take one more column at the higher strength exactly when the original array can take one more column at its strength.	Raising lemma: every fixed strength t >= 3
11	Does the hardness hold at every fixed strength from 3 up?	The raised array has exactly one more column and the stated number of rows.	Raising lemma: every fixed strength t >= 3
12	Does the hardness hold at every fixed strength from 3 up?	For a fixed strength, the number of padding rows the raising construction uses is bounded by a polynomial in the number of columns.	Raising lemma: every fixed strength t >= 3
13	Does the hardness hold at every fixed strength from 3 up?	The raised array has at most the stated polynomial number of rows.	Raising lemma: every fixed strength t >= 3
14	How does an antichain being maximal relate to a list of minimal hitting sets being complete?	Every set that hits all the given sets contains a smallest such set (one with nothing removable).	Maximality of an antichain is completeness of a list of minimal hitting sets (supporting)
15	How hard is it to decide whether a strength-2 table can take one more column?	A set hits the primal family exactly when it meets every given set and also every complement of a given set.	Strength-2 extension reduces to N-1 duality questions; Strength-2 extension and monotone non-duality are equivalent (supporting)
16	How hard is it to decide whether a strength-2 table can take one more column?	Under the stated conditions, a valid column exists exactly when the list of blockers is incomplete (a strength-two array extends exactly when a duality check fails).	Strength-2 extension reduces to N-1 duality questions; Strength-2 extension and monotone non-duality are equivalent
17	How hard is it to decide whether a strength-2 table can take one more column?	The families built in that reduction, and the candidate bad list, are within the stated size bounds.	Strength-2 extension reduces to N-1 duality questions
18	At strength 2, what does a table look like once one row is made the reference row and columns are flipped?	For a strength-two array written with one row all zeros, it extends exactly when its family of sets has a valid support.	Normal form of a strength-2 prefix; Strength-2 extension reduces to N-1 duality questions
19	At strength 2, what does a table look like once one row is made the reference	That array has strength two exactly when every two different sets meet in both required ways.	Normal form of a strength-2 prefix

#	Question it answers	Plain statement	Register result it supports
	row and columns are flipped?		
20	How hard is it to decide whether a strength-2 table can take one more column?	For such an array, extension is equivalent to the blocker list being different from the transversal family.	Strength-2 extension reduces to N-1 duality questions
21	Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?	A set hits every minimal transversal exactly when it contains an original edge.	Strength-2 extension and monotone non-duality are equivalent; Maximality of an antichain is completeness of a list of minimal hitting sets
22	Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?	Under the stated hypotheses, the symmetric family has a valid support exactly when the blocker list is incomplete.	Strength-2 extension and monotone non-duality are equivalent; Structured duality: the reverse reduction is one instance
23	Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?	Two nonempty families that minimally hit each other give a strength-two array of the symmetric form.	Strength-2 extension and monotone non-duality are equivalent; Structured duality: the reverse reduction is one instance
24	Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?	The symmetric array has strength two, and the statement that the blocker list is incomplete translates exactly into the statement that the array extends.	Strength-2 extension and monotone non-duality are equivalent; Structured duality: the reverse reduction is one instance
25	Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?	The reverse construction has exactly the stated numbers of rows and columns.	Strength-2 extension and monotone non-duality are equivalent
26	At strength 2, what does a table look like once one row is made the reference row and columns are flipped?	Flipping the 0/1 labels of any chosen columns does not change the strength.	Normal form of a strength-2 prefix (supporting); Projection rule (complementing a column)
27	At strength 2, what does a table look like once one row is made the reference row and columns are flipped?	Flipping chosen columns does not change whether the array can be extended.	Normal form of a strength-2 prefix (supporting)

#	Question it answers	Plain statement	Register result it supports
28	At strength 2, what does a table look like once one row is made the reference row and columns are flipped?	Reordering the rows does not change the strength.	Normal form of a strength-2 prefix (supporting)
29	At strength 2, what does a table look like once one row is made the reference row and columns are flipped?	Reordering the rows does not change whether the array can be extended.	Normal form of a strength-2 prefix (supporting)
30	At strength 2, what does a table look like once one row is made the reference row and columns are flipped?	The normalised family is the original array after the stated column flips and row reordering.	Normal form of a strength-2 prefix
31	How hard is it to decide whether a strength-2 table can take one more column?	Every strength-two input reduces, after normalising at a chosen row, to the question whether a blocker list is complete.	Strength-2 extension reduces to N-1 duality questions
32	How hard is it to decide whether a strength-2 table can take one more column?	The normalised reduction removes one row and the families stay within the displayed size bounds.	Strength-2 extension reduces to N-1 duality questions
33	Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?	Deleting the non-minimal edges of a family leaves a family that is no larger and has the same hitting sets.	Strength-2 extension and monotone non-duality are equivalent (supporting)
34	Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?	Deleting non-minimal edges does not change the minimal transversals.	Strength-2 extension and monotone non-duality are equivalent (supporting)
35	Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?	The minimal transversals of a family form a clutter (no edge contains another).	Strength-2 extension and monotone non-duality are equivalent (supporting)
36	Is strength-2 extension the same question as a famous	Taking minimal transversals twice gives back the clutter you started with.	Strength-2 extension and monotone non-duality are equivalent (supporting)

#	Question it answers	Plain statement	Register result it supports
	hitting-set puzzle (monotone duality)?		
37	How hard is it to decide whether a strength-2 table can take one more column?	The forward reduction produces two clutters, keeps the yes/no answer, and stays within the stated size bounds.	Strength-2 extension reduces to N-1 duality questions; Strength-2 extension and monotone non-duality are equivalent
38	How hard is it to decide whether a strength-2 table can take one more column?	Both families in that reduction contain only nonempty edges.	Strength-2 extension reduces to N-1 duality questions (supporting)
39	Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?	If a family equals the blocker family, a stated local pair condition holds.	Strength-2 extension and monotone non-duality are equivalent; Local minimality is checkable; completeness is not (12 versus 14)
40	Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?	A fixed small table has strength two and does extend (used as the yes instance).	Strength-2 extension and monotone non-duality are equivalent
41	Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?	For the stated input, the total reverse construction represents the statement that the blocker list differs from the family, correctly.	Strength-2 extension and monotone non-duality are equivalent; Structured duality: the reverse reduction is one instance
42	Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?	The total reverse table obeys the displayed bounds on rows, columns and their product.	Strength-2 extension and monotone non-duality are equivalent
43	Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?	Every total reverse table has at least four rows and two columns.	Strength-2 extension and monotone non-duality are equivalent (supporting)
44	If an outside column already splits a class of rows, how many classes can fail to be split?	If two rows of an array are equal, some extension gives those two rows opposite values.	Transfer lemma: outside columns colour every class; Distinct-row version stays NP-complete (supporting)
45	If an outside column already splits a class of rows, how many classes can fail to be split?	Every required class of rows shows both values in any column outside the columns that define the class.	Transfer lemma: outside columns colour every class

#	Question it answers	Plain statement	Register result it supports
46	If an outside column already splits a class of rows, how many classes can fail to be split?	If the required classes cannot all be split by one new column, the sets of columns that define them cover every column.	Transfer lemma: outside columns colour every class
47	If an outside column already splits a class of rows, how many classes can fail to be split?	An unsplittable family of classes, each defined by at most s columns, forces the number of columns to be at most s times the number of classes.	Transfer lemma: outside columns colour every class
48	If every set in a list passes the local test, is the whole list complete?	A hitting set is minimal exactly when each of its elements has an edge that meets the set only at that element.	Local minimality is checkable; completeness is not (12 versus 14)

Lean names, files and lines

```
 1  class_lemma                     Core.lean, line 23
 2  extension_iff_two_coloring      Core.lean, line 56
 3  support_criterion               Core.lean, line 60
 4  strength_mono                   Core.lean, line 74
 5  strength_on_small_set           Core.lean, line 84
 6  normalized_injective            Width.lean, line 11
 7  width_bound                     Width.lean, line 30
 8  strength_four_width_bound       Width.lean, line 40
 9  raised_strength                 Raising.lean, line 17
10  raised_extends_iff              Raising.lean, line 47
11  raised_counts                   Raising.lean, line 101
12  padding_bound                   RaisingSize.lean, line 7
13  raised_row_bound                RaisingSize.lean, line 45
14  minimal_hit_subset              Hypergraph.lean, line 38
15  hits_primal                     Hypergraph.lean, line 43
16  forward_nonddual                Hypergraph.lean, line 57
17  forward_size_bounds             Hypergraph.lean, line 124
18  anchored_extension_iff          Anchored.lean, line 9
19  anchored_strength_iff           Anchored.lean, line 63
20  anchored_forward_reduction      Anchored.lean, line 94
21  hits_transversals_iff           Reverse.lean, line 7
22  reverse_support_iff             Reverse.lean, line 36
23  sym_strength                    Reverse.lean, line 146
24  reverse_reduction               Reverse.lean, line 216
25  reverse_counts                  Reverse.lean, line 224
26  strength_flip_iff               Normalization.lean, line 9
27  extension_flip_iff              Normalization.lean, line 24
28  strength_row_equiv              Normalization.lean, line 35
29  extension_row_equiv             Normalization.lean, line 45
30  normalized_array_eq             Normalization.lean, line 81
31  full_forward_reduction          Normalization.lean, line 93
32  normalized_reduction_sizes      Normalization.lean, line 107
33  minimize_correct                Clutters.lean, line 13
34  transversals_minimize           Clutters.lean, line 33
35  transversals_clutter            Clutters.lean, line 40
36  transversals_involution         Clutters.lean, line 48
37  forward_clutter_reduction       Clutters.lean, line 68
38  forward_nonempty_edges          Clutters.lean, line 88
39  dual_implies_local              TotalReverse.lean, line 8
40  fixed_yes_correct               TotalReverse.lean, line 40
41  total_reverse_correct           TotalReverse.lean, line 86
42  total_reverse_sizes             TotalReverse.lean, line 99
43  reverse_promise_sizes           TotalReverse.lean, line 116
```

```
44  repeated_row_lemma               FiniteConsequences.lean, line 7
45  outside_column_transfer          FiniteConsequences.lean, line 33
46  uncolorable_classes_cover        FiniteConsequences.lean, line 46
47  uncolorable_class_count          FiniteConsequences.lean, line 60
48  minimal_transversal_local_test   LocalTests.lean, line 5
```

Where to go next

- How to build and check the Lean package: recipe 9 of the check guide.
- What each Lean-proved result means in plain words: the register of results.

19 Register of results

Register of results

Every result in this book, with its exact status. Each entry starts with the question it answers, then the answer, then its status and how it was checked. The answer column is the checked statement; what a result means beyond that statement is discussed in the chapters, where we say 'indicates' and recommend further research. The same data is in the results data file of this folder, and the counts in the README are computed from it.

In this register, one-column extension means the problem of whether a covering array can take one more column without adding rows.

The Lean files for the one-column extension theorems (the lean-hardness folder) and for the further register theorems (the lean-rules folder) are in those two folders of the book, which are not online and are available on request. The download zip holds the main lean folder, which also carries Rule.lean, Growth.lean and Parity.lean as they appear in lean-rules. The Lean statements of these theorems are printed in the theorem list (THEOREMS.md).

Status words: PROVED-Lean, PROVED-written, COMPUTED, CONJECTURED (believed, with the basis stated, and not yet confirmed by a check anyone can run), CONDITIONAL, OPEN, and, in the ledger of

chapter 16, REFUTED (tested and found false), SPECIAL CASES (covered by a more general result) and UNCONDITIONAL BOUND (a bound that stands without a stronger CONDITIONAL one).

Counts by status (all 117 rows): COMPUTED 27; CONDITIONAL 2; CONJECTURED 1; OPEN 4; PROVED-Lean 12; PROVED-written 65; REFUTED 3; SPECIAL CASES 1; UNCONDITIONAL BOUND 2.

Arrays and exact cases beyond the strength-four collection

Result	Question	Answer	Status	How it was checked	Chapter
No CA(12;3,9,2) with two constant rows	Can a binary strength-3 covering array with 12 rows and 9 columns contain two constant rows?	No binary strength-3 covering array with 9 columns and 12 rows contains two constant rows.	COMPUTED	Proof by a SAT solver, checked with drat-trim (verdict s VERIFIED); not yet rechecked a second time by anyone else.	14
CA(15;2,21,3) and CA(23;2,10,4) stay open	Do the covering arrays CA(15;2,21,3) and CA(23;2,10,4) exist?	Whether a 15-row, 21-column, 3-symbol strength-2 array exists (the smallest array known has 16 rows) and whether a 23-row, 10-column, 4-symbol one exists (the smallest array known has 24 rows) remains open. 125 symmetry types were shown impossible; 39 timeouts say nothing.	OPEN	All unsatisfiable answers rest on the solver's word.	14
CA(22;2,9,4) with cyclic symmetry	Is there a 22-row covering array with 9 columns and 4 symbols at strength 2?	A 22-row, 9-column, 4-symbol strength-2 covering array exists, found with a cyclic-shift assumption. It matches the 22 in the tables, whose construction the package could not locate.	COMPUTED	Two programs (Python and Node); rebuilt byte for byte. The array is shipped in the book.	14
A non-extendible CA(12;3,6,2)	Is there a 12-row, 6-column binary strength-3 array that cannot take a seventh column?	A balanced 12-row strength-3 binary array with 6 columns and 12 distinct rows cannot take a seventh binary column (all 4096 candidates checked). It	COMPUTED	A rediscovery of a published listing; no priority is claimed. The array is shipped in the book.	14

Result	Question	Answer	Status	How it was checked	Chapter
		is already known: representative 7 in the listing of Colbourn, Keri, Rivas Soriano and Schlage-Puchta, Discrete Applied Mathematics 158 (2010).			
A binary (1-bar,2)-locating array with 13 rows and 13 columns	Is there a binary table with 13 rows and 13 columns in which the 312 sets of rows showing each setting of each pair of switches are all non-empty and all different?	Yes: the 13 by 13 table in the locating-array folder is a binary (1-bar,2)-locating array. The tables we found in the literature we checked have 14 rows. The literature check is incomplete, because Shi, Fu, Wang and Yan (2021) was not read. We claim no priority.	COMPUTED	Two separately written programs, and recipe 11 of the check guide (a few lines of code you can write yourself): 312 different non-empty row sets of 312; flipping any of the 169 entries makes the check fail. The table is shipped.	13
13 rows as the minimum for a binary locating array with 13 columns	Does a binary (1-bar,2)-locating array with 12 rows and 13 columns exist?	Conjectured: no. In a SAT encoding of the problem, one solver answered UNSAT on 601,831 distinct cubes and SAT on none; 552,691 of them are the terminal cubes that close the search tree. If the search is complete and the encoding and its symmetry rules are right, 13 rows is the minimum for 13 columns.	CONJECTURED	Basis: one solver found no 12-row array in a SAT encoding of the problem; a second solver has re-checked a sample of the search, not the whole search, and the search is not part of this book. The sorting rules (double-lex symmetry breaking) lose no tables by a written argument that the solver does not check. The search is not shipped.	13

Hardness: strength-3 extension is NP-

complete, and every fixed strength

Result	Question	Answer	Status	How it was checked	Chapter
Strength-3 one-column extension is NP-complete (the hub-row reduction)	Is deciding whether a strength-3 table can take one more column NP-complete?	Deciding whether a binary strength-3 covering array can take one more column, with no new rows, is NP-complete. The reduction starts from positive 3-uniform not-all-equal satisfiability with m clauses and writes a strength-3 table with k = 4m+2 columns and N = O(mˆ3) rows (every row explicit). The output always has strength 3, and it extends if and only if the source is satisfiable. Membership in NP is the membership-in-NP result.	PROVED-written	Lean proves the two correctness statements (the output has strength 3 on every input; it extends exactly when the source is satisfiable); the Lean source is in the hardness folder, which is not online (available on request; check guide, recipe 10); the Lean package that is online does not contain these two theorems. Polynomial running time is proved in writing only, not in Lean (the polynomial-time note shipped with the book), and the written proof is shipped too. NP-completeness of the source problem (Schaefer 1978) is taken as given. 1,359 small source puzzles agree with brute force (36 unsatisfiable); 1,035 of them (all 1,024 clause families on five variables and 11 named sources, including the Fano plane) were rerun with no mismatch. A proof rebuilt from the theorem statement alone found no error. Lean 4.34.1 compiler and kernel only, run against the pinned Mathlib (no errors, no admitted steps, only the standard axioms). Second checkers: Lean's own replay checker (leanchecker) passed on the hardness folder, and nanoda 0.4.26 accepted an export of its 12 listed declarations (5,195 declarations checked). The status is PROVED-written because the polynomial-time argument and the NP-completeness of the source problem are not formalised.	9
Raising lemma: every	If a table of strength t is	Let M have strength t with at least t	PROVED-Lean	Lean proves the raising lemma and its size bounds (48-theorem package:	9

Result	Question	Answer	Status	How it was checked	Chapter
fixed strength t >= 3	raised to strength t+1 by the padding construction, does the ability to take one more column carry over?	columns. The raised array M' (add a column w; keep the old rows with w = 0; add every row with at most t+1 zeros with w = 1, each twice) has strength t+1, and M' extends at strength t+1 if and only if M extends at strength t. For a fixed t the number of padding rows is bounded by a polynomial in the number of columns (it grows like k^(t+1)).		theorems 9 to 13 of the Lean theorem table). The composition with the hub-row reduction, which gives NP-completeness at every fixed strength, is the next row of this register (PROVED-written). Rechecked on 202 small cases plus 19 full strength-four reductions. Checked by the Lean 4.34.1 kernel; the exported proofs of the whole 48-theorem package were also re-checked by a second kernel checker, nanoda (6,174 declarations). Lean's own replay checker (leanchecker) also passed. No lean4lean run.	
NP-completeness at every fixed strength t >= 3 (raising lemma composed with the hub-row reduction)	Does the hardness hold at every fixed strength from 3 up?	Composed with the hub-row reduction, one-column extension of a binary covering array of every fixed strength t >= 3 is NP-complete, by a polynomial reduction whose padding grows like k^(t+1) (polynomial for fixed t only; nothing is claimed for t growing with the input).	PROVED-written	The raising lemma and its size bounds are proved in Lean (previous row). The composition inherits the written polynomial-time argument of the hub-row reduction and the NP-completeness of its source problem (Schaefer 1978), neither of which is formalised. Rechecked on 202 small cases plus 19 full strength-four reductions.	9

Result	Question	Answer	Status	How it was checked	Chapter
Distinct-row version stays NP-complete	Does the hardness remain when all rows of the table are different?	The hub construction with every padding row written once (coloured by the parity of its number of zeros), twin variables removed beforehand, and a fixed small array output when at most one clause survives, gives a strength-3 table whose rows are all different and which extends exactly when the source is satisfiable.	PROVED-written	All 1,359 sources rechecked (strength, distinct rows, exact equivalence; the 36 negative sources stay negative).	9
Thin arrays and both properties at once	Does the hardness remain for thin tables, with about as many rows as columns?	The padding can be shrunk from about m^3 rows to O(log m), so the hard tables have about as many rows as columns, and with twin rows the rows stay distinct and the tables stay thin at every fixed strength.	PROVED-written	Read line by line in a separate reading (no mathematical error); computer checks agree with brute force on more than 2,000 small puzzles. Not yet rechecked a second time.	9
Ternary strength-2 one-column extension is NP-complete	Is one-column extension NP-complete for ternary tables at strength 2?	Deciding whether a ternary (three-symbol) strength-2 covering array can take one more column, with no new rows, is NP-complete.	PROVED-written	Rebuilt from the written proof alone: 7,404 small source puzzles against two complete solvers, 71 negative, no disagreement. One side remark in the source (about repeated rows when twin vertices occur) was found false; it is corrected in the proof file.	13

The ledger (chapter 16)

Result	Question	Answer	Status	How it was checked	Chapter
Two special branches of the normal form	Which special cases does the general normal form cover?	(1) A cell of size two reduces extension to T_0 differing from Tr(B); (2) two identical rows reduce it to G differing from Tr(G). Both are covered by the reduction of strength-2 extension to duality questions, which covers every prefix. The identical-row case also needs an extra free element and must not be generalised without it.	SPECIAL CASES	The working note itself says that its general reduction (the reduction of strength-2 extension to duality questions) covers both branches.	16
Affine model set, with Ladder bounds 16 and 22	Which Ladder lower bounds follow from the affine structure of isolated models?	A satisfiable CNF with n clauses and only isolated models has an affine model set, so the number of models is a power of two. It gives Ladder lower bounds 16 and 22 for 12 and 16 rows; the four-isolated-models theorem gives the stronger bounds 17 and 23. Since the four-isolated-models theorem is CONDITIONAL, 16 and 22 remain the lower bounds that do not depend on it.	UNCONDITIONAL BOUND	The stronger bounds 17 and 23 are CONDITIONAL; 16 and 22 remain the unconditional bounds until the proofs of the four-isolated-models theorem are regenerated.	16
Tight cross bound of two isolated assignments	How many isolated satisfying assignments can a cross-intersecting monotone formula have?	A cross-intersecting monotone CNF with n clauses has at most two isolated satisfying assignments. The four-isolated-models theorem, which is CONDITIONAL, gives a stronger bound; until the	UNCONDITIONAL BOUND	A stronger, CONDITIONAL bound comes from the four-isolated-models theorem.	16

Result	Question	Answer	Status	How it was checked	Chapter
		four-isolated-models theorem's proofs are regenerated, the bounds this gives (6, 10, 16, 22, 22) remain the unconditional Ladder lower bounds at 8, 10, 12 and 16 rows; at 14 rows its 22 is replaced by the exact minimum 24.			
Canonical pair is not always the cheapest	Is the canonical boundary formula always the cheapest pair with a given gap set?	The canonical boundary formula does not always give the cheapest pair with a given gap set: for the gap set {3,12} the canonical cost is 8, but F = {5,10}, G = {6,9} costs 4.	REFUTED	A counterexample: the gap set {3,12}.	16
Registered rules that failed	Did the rules that were stated in advance for the Ladder hold?	A rule predicting 15 edges on 12 rows produced 153; the hypothesis that one extra clause suffices (16, 22, 22) was refuted on all three rungs. Source-aware swap searches produced 48 candidates that were unsound.	REFUTED	Finite computations; none is a theorem.	16
Row-irreducible implies NO: false	Does a row-irreducible strength-3 array always lack room for a new column?	A 12-row, 4-column row-irreducible strength-3 array extends (delete 0000, 0011, 1101, 1110 from the 4-cube and colour by x1+x2+x3); an 11-row one also exists. Of 65,536 subsets of the 4-cube exactly 42 are row-irreducible: 40 extend, and two 8-row parity arrays do not.	REFUTED	Exhaustive; 70 DRAT proofs checked.	16

Result	Question	Answer	Status	How it was checked	Chapter
Counts for the older collection	How do the arrays of the older collection compare with the table?	The older strength-2 collection has 720 arrays: 69 below the table, 98 equal and 553 above (frontier not yet checked).	COMPUTED	Counted from the list of arrays (statement/strength2-arrays-listing.csv).	16

Open questions

Result	Question	Answer	Status	How it was checked	Chapter
Does a 346th column exist for the 170 x 345 array?	Does a 346th column exist for the 170 x 345 array?	Whether the 170-row, 345-column array supplied here can take a 346th column without adding rows (a 170 x 346 array) is open; no search for one has been run on it.	OPEN	Chapter 11 states the rule that such a column must satisfy.	16
Exact Ladder minima at 12 and 16 rows	What are the exact Ladder minima at 12 and 16 rows?	The smallest $\|F\|+\|G\|$ for the 12-row and 16-row Ladder tables is known only to lie between 16 and 70, and between 22 and 654 (between 17 and 23 at the lower ends if the four-isolated-models theorem is accepted).	OPEN	See the Ladder table and the four-isolated-models theorem.	16
The smallest possible number of rows	What is the smallest possible number of rows at any of the 29 widths?	For every width in the collection the smallest possible number of rows is unknown: the lower bounds we computed are 47 rows at widths 201 to 203 and 50 rows at widths 325 to 342 (21 of the 29 widths; none was computed at 200, 204 to 206, 223 or 343 to 345), and the arrays supplied have 143 to 170 rows.	OPEN	See the lower bounds on rows.	16

Strength 2: can one more column be added?

Result	Question	Answer	Status	How it was checked	Chapter
Normal form of a strength-2 prefix	At strength 2, what does a table look like once one row is made the reference row and columns are flipped?	Fix one row u. Each column becomes the set T_j of other rows that hold a 1 where u holds a 0 (after flipping columns so that u is all zeros). The table is a binary covering array of strength 2 exactly when the sets T_j form an intersecting antichain (any two sets meet, none contains another). It extends by one column exactly when {T_j} is not a maximal intersecting antichain.	PROVED-written	Proof read and accepted in a separate reading; 2,901 random prefixes (4 to 8 rows) compared with brute force, no disagreement. Lean covers the anchored form.	6
Strength-2 extension reduces to N-1 duality questions	How hard is it to decide whether a strength-2 table can take one more column?	A binary strength-2 table with N rows extends by one column exactly when, for at least one row w other than the reference row, the transversal family of a derived set family H_1 differs from the minimal members of a second family H_2' (both built from the table). That is N-1 monotone-duality questions, each of size polynomial in the table, so strength-2 extension can be decided in quasi-polynomial time (using the Fredman-Khachiyan algorithm).	PROVED-written	Accepted as a mathematical argument in a separate reading; two seeded exhaustive runs (1,169 and 1,176 random prefixes plus 450 structured ones each) all agree; a further comparison on 2,901 prefixes had 0 mismatches. Lean covers the per-row reduction, not the running-time statement.	6
No polynomial NP-hardness	Could strength-2	If a polynomial-time reduction took signed	CONDITIONAL	A written implication, valid only if the	6

Result	Question	Answer	Status	How it was checked	Chapter
reduction for strength-2 extension unless ETH fails	extension be NP-hard?	not-all-equal 3-SAT to strength-2 column extension, then not-all-equal 3-SAT would be solvable in time 2^{o(log^2 n)}, which contradicts the Exponential Time Hypothesis (ETH).		Exponential Time Hypothesis is true; the implication was found valid in a separate reading. Not an unconditional statement about strength 2.	
Strength-2 extension and monotone non-duality are equivalent	Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?	Deciding whether a binary strength-2 covering array can take one more column (CA-Ext) and deciding whether two monotone Boolean functions given as families are not dual (CO-DUAL) can each be turned into the other, with the answer preserved. The reverse construction has n0+3 rows; the correctness and the size bounds of both constructions are proved in Lean.	PROVED-Lean	Lean proves the correctness and the stated row, column and size bounds of both reductions (48-theorem package: theorems 24, 25, 37, 41 and 42 of the Lean theorem table). The time needed to carry the constructions out is a separate result, in the next row of this register (PROVED-written). Also compared exhaustively on 163 antichains, 27,524 pairs and 2,192 prefixes with no mismatch. The reverse direction is Eiter and Gottlob's Theorem 4.13 (credit: Seymour). Checked by the Lean 4.34.1 kernel; the exported proofs of the whole 48-theorem package were also re-checked by a second kernel checker, nanoda (6,174 declarations). Lean's own replay checker (leanchecker) also passed. No lean4lean run.	6
The reductions between	How long does it take to carry out the two	Both constructions of the previous result can be carried out in	PROVED-written	Written argument only; the running time is not formalised in Lean. The	6

Result	Question	Answer	Status	How it was checked	Chapter
strength-2 extension and monotone non-duality run in polynomial time	reductions between strength-2 extension and monotone non-duality?	polynomial time; the forward direction, from a covering array with N rows and k columns to a pair of families, takes O(kˆ2 Nˆ3) bit operations. Together with the previous result, each of the two problems reduces to the other in polynomial time, with the answer preserved.		correctness and size bounds it relies on are the Lean-proved ones of the previous row.	
One-row antichain extension lemma	In the family form of strength 2, exactly when can one more set be added?	A one-row table A(F) built from a family F is a valid strength-2 prefix exactly when F is a pairwise-intersecting antichain; it extends exactly when an incomparable transversal exists. If F is contained in its own transversal family Tr(F), it extends exactly when Tr(F) differs from F.	PROVED-written	Checked exhaustively inside the comparison for the equivalence of strength-2 extension and duality (no mismatch).	6
Maximality of an antichain is completeness of a list of minimal hitting sets	How does an antichain being maximal relate to a list of minimal hitting sets being complete?	An intersecting antichain F is maximal exactly when T_0 = Tr(B); there is a reverse construction with fresh points p, q and a coatom C.	PROVED-written	Three separate readings; no code.	6
Local minimality is checkable; completeness	If every set in a list passes the local test, is the whole list complete?	In a valid CA(8;2,10,2) with exactly four extensions, the list of 12 minimal sets passes every local minimality	PROVED-written	Hand derivation; not re-run. Lean covers the local test.	6

Result	Question	Answer	Status	How it was checked	Chapter
is not (12 versus 14)		check, yet the full transversal family has 14 members, so the list is incomplete.			
Fixed rank spread is tractable	Is the strength-2 question easy when the sizes of the sets stay within a fixed spread?	For a fixed rank spread w, the decision is polynomial. Still open: a fixed-parameter tractability bound in w for this method.	PROVED-written	Algorithm on paper; not implemented.	6
Exact count of extensions	How many extensions does a strength-2 prefix have?	The number of extensions of the prefix is 2W(F), where W(F) = 2^n - |up(F)| - |down(F)| + m.	PROVED-written	On paper; not implemented.	6
A rank-counting sufficient certificate	If a family has fewer sets of some size than there are sets of that size, must an incomparable set exist?	If C_t < binom(n,t), an incomparable t-set exists, so the prefix extends. This is sufficient only; a counter-example shows it is not necessary.	PROVED-written	A counter-example is given in the note.	6
Structured duality: the reverse reduction is one instance	Which structured duality questions turn into strength-2 tables, and what does that say about their hardness?	For the structured form, 'general non-duality' holds exactly when the prefix extends, with N = n+3 rows and k = b+t+1 columns; with the reduction of strength-2 extension to duality questions this gives polynomial-time Turing equivalence.	PROVED-written	Reading only; no computation. The note itself calls it a reconciliation of known results.	6
An easy region: the Lambda test	Is there a simple test that guarantees a strength-2	Let Lambda = sum over columns j of [2^(1-|S_j|) + 2^(1-|T_j|)]. If Lambda < 1, an extension exists and can	PROVED-written	Algorithm on paper; not implemented.	6

Result	Question	Answer	Status	How it was checked	Chapter
	extension and builds it in polynomial time?	be constructed deterministically in polynomial time.			
A shortcut that fails: a maximal intersecting antichain need not be self-dual	Is every maximal intersecting family of sets also self-dual?	Take F = all four triples of {a,b,c,d}. F is a maximal intersecting antichain, yet Tr(F) is the six pairs, so the 5x4 prefix does not extend. 'Maximal' does not mean 'equal to its own transversal family'.	PROVED-written	A four-point example, checked by hand and by computer.	6
Stuck arrays: small unsatisfiable cores of seven 'no more columns' cases	Why do some arrays accept no extra column, and is the reason small?	Seven arrays (3 at strength 2, 4 at strength 3; 5 to 8 symbols) for which a solver answered 'no column fits' each have a small unsatisfiable core; strength-2 cores use all pairs of columns, strength-3 cores are sparse (5 to 18 pairs). These are statements about a fixed prefix, never lower bounds on the smallest possible array.	COMPUTED	All seven were run again with a second solver (7 of 7 unsatisfiable); two cores were recomputed. Not every proof file was kept.	6
The strength-2 equivalence with monotone duality	Is strength-2 extension equivalent to monotone non-duality?	The reverse direction (non-duality gives an extension) is word for word Eiter and Gottlob's Theorem 4.13. The forward direction (any covering array gives a duality instance) is the part we did not find stated in the sources we read. One	PROVED-written	About 1.5 million cases compared with brute force, 0 disagreements; a separate check reproduced all 400 numbers.	6

Result	Question	Answer	Status	How it was checked	Chapter
		lemma (a step about the one-row table A(F) built from a family F) needs the hypothesis that A(F) is a strength-2 covering array; without it, the lemma fails for 310 families on 4 points.			

Strength 3: pair classes (the criterion, easy

cases, and what blocks the obvious routes)

Result	Question	Answer	Status	How it was checked	Chapter
Pair-class formulation	At strength 3, how is the question about one more column phrased through pair classes?	A binary strength-3 table extends by one column exactly when its rows can be coloured 0/1 so that every pair class E(a,b;x,y) (the rows with value x in column a and y in column b) contains both colours. Equivalently, a new column keeps strength 3 exactly when it takes both values on every pair class. This is the t = 3 case of the class lemma (which Lean proves in general).	PROVED-written	Proof read and accepted in a separate reading. The criterion itself appears in the literature (Kokkala 2017 thesis, section 3.5; Hiess 2024 thesis); we claim no priority for it.	7
Strength-3 extension is in NP; the classes form an antichain	Can a proposed new column at strength 3 be checked quickly, and how do the pair classes sit inside each other?	A proposed column is checked in O(N k^2) time. The pair classes have at least 2 rows each, and a table with k columns has exactly 4*binom(k,2) distinct classes, none contained in another.	PROVED-written	Proof read and accepted in a separate reading.	7
Padding lemma for intersecting Sperner hypergraphs	Can padding rows turn a family of sets into a table whose extension matches two-colouring of the sets?	For an intersecting Sperner hypergraph J, the table A_2(J) (with N = n + dQ(s) rows and k = 2s columns) is extendible exactly when J is properly two-colourable.	PROVED-written	Proof read and accepted; the 22 x 6 worked example has 420,244 valid new columns out of 2^22, and 300 random columns were compared, 0 mismatches.	7
Complement of duality reduces to	Does the complement of monotone	There is a polynomial-time map f such that f(J,G) is always a	PROVED-written	Proof read and accepted in a separate reading; 60 instances compared with	7

Result	Question	Answer	Status	How it was checked	Chapter
strength-3 extension	duality reduce to strength-3 extension?	binary strength-3 array and G differs from Tr(J) exactly when f(J,G) extends. So the complement of the duality problem reduces to strength-3 extension.		brute force, 0 mismatches. We claim no priority: the link between duality and colouring intersecting hypergraphs is in Elbassioni 2016 and Kokkala 2017.	
Decision orientation	In which direction should the yes and no answers of duality and extension be matched?	For an intersecting Sperner hypergraph H, H is not two-colourable if and only if H = Tr(H); G differs from Tr(H) exactly when F is two-colourable exactly when A(F) is extendible.	PROVED-written	Accepted on reading.	7
Exact formulation through bad transversals	Can strength-3 extension be stated exactly through a list of bad hitting sets?	Strength-3 extension is possible exactly when B(H) differs from Tr(H), exactly when the duality question for (H, B(H)) has answer NO, where B(H) is the list of bad minimal transversals.	PROVED-written	Accepted on reading; benchmark counts recomputed (cube 6/4/2, parity 4/4/0, m = 4: 1393/1393/0).	7
The bad-transversal list can be superpolynomial	How long can the list of bad hitting sets get?	For the incidence arrays H_m, |B(H_m)| >= 2^binom(m-1,2) on distinct-row NO arrays, a growth of 2^Theta(sqrt L) in the table size L. This blocks only the direct route 'list B(H), then use duality'.	PROVED-written	Accepted; values 8, 64, 1024 for m = 4, 5, 6 recomputed.	7
Transfer lemma: outside columns colour every class	If an outside column already splits a class of rows, how many	Every old column outside the column set that defines a pair class is a proper colouring of	PROVED-Lean	Lean proves the finite combinatorial statements; the sharpness of the bound on the	8

Result	Question	Answer	Status	How it was checked	Chapter
	classes can fail to be split?	that class. Hence a family of classes that cannot be split by any new column has defining column sets covering every old column, and so contains at least ceil(k/2) classes.		family J_m is written only. Also covered by a reading of the whole document on the eighteen walls (no false implication found). Checked by the Lean 4.34.1 kernel; the exported proofs of the whole 48-theorem package were also re-checked by a second kernel checker, nanoda (6,174 declarations). Lean's own replay checker (leanchecker) also passed. No lean4lean run.	
Private-support obstruction	Can each clause of a source problem be tied to its own private columns?	If each source edge is enforced on its own column support and the supports are disjoint, the Fano plane together with a disjoint triple cannot be represented.	PROVED-written	Computed on the named sources.	8
A logical bridge gadget (hypergraph level only)	At the level of hypergraphs, does a bridge gadget give a controlled number of extensions per source colouring?	Bridge vertices give exactly 2^d extensions per source colouring. This is a statement about hypergraphs, not an array construction.	PROVED-written	Counts checked on an example.	8
Off-edge label certificates and safe padding	Do label certificates make the extensions match the source colourings, and are such labels easy to find?	Extension projections equal the source colourings when a label certificate exists; deciding whether labels exist is itself NP-complete (it needs 3-uniform hypergraph two-colouring), so	PROVED-written	Counts checked on four examples.	8

Result	Question	Answer	Status	How it was checked	Chapter
		labels conceal the source problem.			
Washing: padding and duplicate rows free the colours	If a frame of old columns can already take a new column, does appending rows keep it so?	If a fixed old-column frame B is already extendible, appending any s rows keeps it extendible: any extension of B, followed by any of the 2^s assignments on the new rows, is an extension.	PROVED-written	Counts checked on four examples.	8
Union-bound test	Is there a counting test that guarantees a strength-3 table can take one more column?	If W = sum over classes E of 2^(1-|E|) is less than 1, an extension exists and can be found by a deterministic polynomial algorithm. Example: a 64x32 table has W = 31/512.	PROVED-written	Checked on the example.	7
Lovasz Local Lemma test (Moser-Tardos)	Does a finer test, from the Lovasz Local Lemma, guarantee an extension where the counting test is silent?	If weights x_E satisfy 2^(1-|E|) <= x_E * prod over overlapping F of (1 - x_F), an extension exists and a randomised resampling procedure finds it. Example: a 28x3 table has union-bound sum 1153/1024 > 1 yet passes this test.	PROVED-written	Recomputed on the example. Relies on the published Moser-Tardos theorem (arXiv:0903.0544), taken as given.	7
The shared design A_r (the [8,4,4] example)	Is there a table that has room for new columns but loses all of it when any single row is deleted?	The 16x8 design has exactly 56 extensions, and all 16 one-row deletions have 0; the lift fails for r >= 4.	PROVED-written	All 65,536 columns enumerated; a recheck agreed.	7
Shared-column equality stars:	Can small strength-3 tables force many	Polynomial-size binary strength-3 arrays can force arbitrarily many	PROVED-written	31 arrays checked (23 as promised, 8 that fail as	7

Result	Question	Answer	Status	How it was checked	Chapter
the part that works	copies of a variable to carry equal bits in any extension?	occurrence copies of a variable to carry equal extension bits. A single reference shared by two disjoint stars fails strength 3.		predicted); the 13 x 3 star has 534 extensions.	
Stars cannot be attached to a previous occurrence array	Can an equality network on a few old columns tie together copies of a variable that are far apart in Hamming distance?	Preserved occurrence codes at Hamming distance greater than s cannot be tied by an equality network supported on at most s old columns.	PROVED-written	Counts on the Fano plane and triples.	8
Complementary hubs enforce the wrong not-all-equal relation	Do complementary hubs enforce the right not-all-equal relation?	Exactly 2^n' - 2m consistent core assignments extend; all three unsatisfiable test sources become YES, so this gadget does not encode the source problem.	PROVED-written	Eight sources, zero false negatives.	8
Uniform shared flags give not-all-equal on all variables	Do shared flags give not-all-equal on all variables at once, and is that useful for single clauses?	On a shared flag c_x the construction enforces NAE(x_1,...,x_n): it accepts exactly the 2^n - 2 non-constant assignments. This is the wrong relation for a clause-by-clause encoding.	PROVED-written	Eight sources checked.	8
Conditional exact-preservation theorem for supplied gadget data	If the gadget data satisfied all the stated equations, would the construction be exact?	If gadget data B and one choice per source clause satisfy all the stated equations, the safe-padding construction is a strength-3 array and its projection of the extensions equals the	PROVED-written	The implication is proved; the equations could not be satisfied in the cases tried.	8

Result	Question	Answer	Status	How it was checked	Chapter
		set of assignments with NAE on every clause. The hypotheses could not be satisfied in the cases tried (finite refutations, 607,275 search nodes).			
Auxiliary selector encoding is not ordinary duality	Is the selector encoding with auxiliary variables the same as ordinary duality?	With a shared auxiliary variable z the witnesses are preserved but DUAL(F,G) = NO on every promised array: a spurious gap always remains.	PROVED-written	Six benchmarks and seven two-vertex hypergraphs.	8
Compression lower bounds for the forbidden DNF	Is there a lower bound on how small the forbidden formula of a proposed reduction can be?	Under the cofactor hypothesis, |G_A| >= |Tr(K_A)|. This is a property of a proposed reduction, not a statement about ETH.	PROVED-written	65,536 formulas checked; the bad-transversal count over 2^20 subsets.	8
Changing both duality families does not help	Does changing both duality families help?	A private-intersection family always has a false source gap; repairing G alone needs at least 3^M terms.	COMPUTED	Written up in the document on the eighteen walls.	8
Interacting auxiliaries: a fitted compiler and the 167x9 escape	Does a compiler fitted to the full solution sets give a valid reduction?	Exact source projections on all eight test sources, but the construction is not a polynomial-time reduction (it uses the full solution sets).	COMPUTED	31 arrays checked; one further check of our own only.	8
Uniform clause constructions fail	Do uniform clause constructions work?	Exact projection passes 1 of 28 primary and 0 of 28 auxiliary cases, and all 13 unsatisfiable sources give YES.	COMPUTED	68 matrices rechecked; the second recipe was not tested blind.	8

Result	Question	Answer	Status	How it was checked	Chapter
An orientation recipe	Does a recipe that fixes the orientation of each clause work for all clause sets?	The recipe repairs the sources {123,124,345} but is exact in only 6 of 28 cases. No second recipe was tried.	COMPUTED	Computed on the named sources.	8
The eighteen walls	Why do the obvious routes from small gadgets to a hardness proof fail?	Eighteen grouped failure results explaining why the obvious routes from small gadgets to a hardness proof fail. A separate reading found 'No false mathematical implication' and states that the document does not establish a universal obstruction to a hardness reduction. It counts 110 results: 22 universal walls, 41 template-specific failures and 47 finite observations.	PROVED-written	Read through (not checked by a proof assistant): 59 proved, 28 computed and 7 assumed among the 94 rows that carry an evidence label. Replaced as a route by the hub-row reduction.	8
Separate star tests are not a characterisation	Does testing each star separately decide extension?	Requiring each star to be separately feasible gives only AND over j of 'exists c_j'; extension needs one common c for all stars. (The 21x6 triangle fixture: all six stars are YES but the table is NO.)	PROVED-written	The 21 x 6 example checked.	8
No fixed VC-dimension bound; two-family gap feasibility is NP-complete	Is there a fixed bound on the VC-dimension of these families, and how hard is two-family gap feasibility?	Pair classes can shatter arbitrarily many rows; the general two-family gap problem is NP-complete but does not by itself build an array.	PROVED-written	Accepted on reading.	8

Result	Question	Answer	Status	How it was checked	Chapter
Worked example tables for chapters 5 to 7	Which small tables serve as worked examples for chapters 5 to 7?	A set of small YES and NO benchmark tables for extension, for example a 22 x 6 YES table (420,244 valid columns), a 21 x 6 NO table, an odd-parity 8 x 4 table, a 16 x 8 table with 56 extensions and a 28 x 3 table for the local-lemma test.	COMPUTED	Each worked example is small enough to recheck by exhaustive search.	7
Class lemma: extension is proper two-colouring of the row-class hypergraph (every strength)	When can a finished table of strength t take one more column without adding any row?	For a table of strength t, a new column keeps strength t exactly when it takes both values on every class E(S;x) with $\|S\| = t-1$. Hence an extension exists exactly when the rows can be two-coloured so that no required class is monochromatic.	PROVED-Lean	Lean theorems on the class lemma and the colouring form (listed in the Lean theorem table). The criterion is known (Kokkala 2017; Hiess 2024); we claim no priority for it. Checked by the Lean 4.34.1 kernel; the exported proofs of the whole 48-theorem package were also re-checked by a second kernel checker, nanoda (6,174 declarations). Lean's own replay checker (leanchecker) also passed. No lean4lean run.	5
The easy side of strength-3 extension	Which strength-3 tables have no room, and what are the easy cases?	Every proper column projection of a strength-3 array extends (the omitted column is the witness). With b the maximum intersection of two classes, extension is decidable in O(2^(8b) N k^2) time; $b = 1$ is graph bipartiteness.	PROVED-written	All 30 catalogue arrays tested were row-irreducible; reading plus computation.	7

Result	Question	Answer	Status	How it was checked	Chapter
		The smallest NO array has 8 rows (the rows x, y, z, x+y+z). Every class has between 2 and N-6 rows.			
A polynomial subclass: unit-row arrays	Is the question easy for tables made only of unit rows?	For arrays whose extra rows are unit rows, the table extends exactly when both constant columns are present; then exactly two extension columns exist.	PROVED-written	Described in the source as an elementary subclass.	7
Support criterion for strength 4	At strength four, what exactly must a new column satisfy so that the table keeps strength four?	A new column x preserves strength four exactly when, for every choice J of three old columns and every pattern a on J, the rows showing a on J include one with $x = 0$ and one with $x = 1$. (The same statement as the class lemma, one level up.)	PROVED-Lean	Lean theorem on the support criterion. This is the test used in every search for a new column of a strength-four array. Checked by the Lean 4.34.1 kernel; the exported proofs of the whole 48-theorem package were also re-checked by a second kernel checker, nanoda (6,174 declarations). Lean's own replay checker (leanchecker) also passed. No lean4lean run.	5

Strength four: rules, growth, drop-a-row and bounds

Result	Question	Answer	Status	How it was checked	Chapter
Residual-row completion rule	When can a kept block of rows be completed to a strength-four array with r more rows?	Let F be a fixed block of binary rows on a set of columns, and M the set of 4-column partial words that no row of F shows. Join two members of M when they ask for opposite bits in some common column. Then r further binary rows make F a strength-four array exactly when this conflict graph has a proper colouring with r colours. (Each colour class is merged into one row.)	PROVED-Lean	Lean theorem on the completion rule in its general partial-word form: the rule is stated for the set M of missing four-column patterns. The step from an array file to that set is a separate result, in the next row of this register (PROVED-written). Checked by the Lean 4.34.1 compiler and kernel against the pinned Mathlib (no errors, no admitted steps, standard axioms only). Second checkers: Lean's own replay checker (leanchecker) passed on the rules folder, and nanoda 0.4.26 accepted an export of its 29 listed declarations (7,503 declarations checked).	10
From array bytes to missing partial words	Does the completion rule apply to an actual array file, and not only to a set of patterns?	For a block F of rows read from an array file, listing for every choice of four columns the patterns that some row shows gives exactly the set M of four-column partial words that no row of F shows, to which the completion rule applies.	PROVED-written	Written step; not formalised in Lean. Tested on 960 exhaustive comparisons over 240 seeded systems, all agreeing.	10
Block-of-columns criterion	What must be checked when a block of new columns is added at once?	For a block of new columns the necessary and sufficient conditions include every choice of j new and 4-j old columns,	PROVED-written	Written classification of four-column sets by the number of new columns; no Lean.	10

Result	Question	Answer	Status	How it was checked	Chapter
		for $j = 1, 2, 3, 4$. Checking $j = 1$ alone is not enough.			
Projection rule	Does keeping only some columns of a strength-four array keep strength four?	Restricting a strength-four array to any set of distinct columns keeps strength four; permuting columns and complementing a column also keep coverage.	PROVED-Lean	Lean theorems: the projection theorem (rules folder; no axioms) covers restricting to any set of distinct columns and permuting columns; the theorem on flipping columns (theorem 26 of the Lean theorem table, 48-theorem package) covers complementing a column. These two theorems use different Lean definitions of strength four; the bridge file in the Lean folder proves that the definitions agree. Checked by the Lean 4.34.1 compiler and kernel against the pinned Mathlib (no errors, no admitted steps, standard axioms only). Second checkers: for the rules folder, Lean's own replay checker (leanchecker) and nanoda 0.4.26 (an export of its 29 listed declarations, 7,503 declarations checked); for the 48-theorem package, nanoda (6,174 declarations) and leanchecker. No lean4lean run.	10
Graph-colouring form of fixed-row replacement	How does the row-completion rule look as a graph-colouring question?	Replacing a chosen set of removed rows by m new rows can succeed exactly when the conflict graph of the patterns missing from	PROVED-written	A special case of the residual-row completion rule.	10

Result	Question	Answer	Status	How it was checked	Chapter
		the kept rows has a proper colouring with at most m colours.			
Width ceiling for a fixed number of rows	How many columns can a binary strength-four array with N rows have at most?	A binary covering array of strength 2, or of strength 4 with at least four columns, with N rows has at most 2^(N-1) columns.	PROVED-Lean	Lean theorems on the width bounds (48-theorem package). Checked by the Lean 4.34.1 kernel; the exported proofs of the whole 48-theorem package were also re-checked by a second kernel checker, nanoda (6,174 declarations). Lean's own replay checker (leanchecker) also passed. No lean4lean run.	10
A tempting invariant fails	If every new column splits every triple of the old columns, can such columns be added one after another?	On the 16-row full factorial, adding parity and then the identical parity column again gives two columns that both pass the seed support test yet never disagree; so 'keep splitting every triple' is not preserved by adding columns one at a time.	COMPUTED	Exhaustive over all 65,536 candidate columns of the 16-row table: only parity and its complement extend it.	10
Repeatable growth recurrence (clone and repair)	Can a strength-four array always be grown by one column, and at what cost in rows?	A binary strength-four array with k >= 4 columns can always be grown by one column by cloning a column and adding repair rows: if the other k-1 columns are given distinct m-bit labels (m = ceil(log2(k-1)) bits are enough), the grown array has k+1 columns and N+4m+4	PROVED-Lean	Lean theorems on cloning a column and repairing, and on the row and column counts, for any labelling of the other columns by distinct m-bit words. The choice m = ceil(log2(k-1)) and the comparison with the pinned table are the next row of this register (COMPUTED). Checked by the Lean 4.34.1 compiler and kernel against the	10

Result	Question	Answer	Status	How it was checked	Chapter
		rows. Repeating this gives $k_{(s+1)} = k_s + 1$ and $N_{(s+1)} = N_s + 4m + 4$.		pinned Mathlib (no errors, no admitted steps, standard axioms only). Second checkers: Lean's own replay checker (leanchecker) passed on the rules folder, and nanoda 0.4.26 accepted an export of its 29 listed declarations (7,503 declarations checked).	
Clone obstruction	How many repair rows does copying a column need?	Copying a column needs at least eight repair rows.	PROVED-Lean	Lean theorem that the eight patterns demanded by a copy need eight different rows (the copy case). The complement case is the next row of this register (PROVED-written). Checked by the Lean 4.34.1 compiler and kernel against the pinned Mathlib (no errors, no admitted steps, standard axioms only). Second checkers: Lean's own replay checker (leanchecker) passed on the rules folder, and nanoda 0.4.26 accepted an export of its 29 listed declarations (7,503 declarations checked).	10
Complementing a column needs at least eight repair rows	How many repair rows does complementing a column need?	Complementing a column (adding it with 0 and 1 swapped) needs at least eight repair rows.	PROVED-written	Follows from the copy case by flipping the column. Written argument only: no Lean statement joins the Lean theorem on the copy case with the Lean theorem on flipping a column.	10
No universal one-row extension (parity 16x5)	Is there a universal way to add one column and one row to a	The claim that every strength-four array admits a new column with at most one	PROVED-Lean	Lean theorem on the 16 even-parity rows. Checked by the Lean 4.34.1 compiler and kernel against the	10

Result	Question	Answer	Status	How it was checked	Chapter
	strength-four array?	added row is false: take the 16 even-parity rows on five bits. Special seed families are not excluded.		pinned Mathlib (no errors, no admitted steps, standard axioms only). Second checkers: Lean's own replay checker (leanchecker) passed on the rules folder, and nanoda 0.4.26 accepted an export of its 29 listed declarations (7,503 declarations checked).	
A full linear parity block is too large	Could a purely linear block of columns give a small strength-four array at width 345?	A binary linear block in which every four coefficient vectors are independent needs at least 65,536 rows at width 345 (the pinned bound is 174 there).	PROVED-Lean	Lean theorems on the full linear block and on width 345. Checked by the Lean 4.34.1 compiler and kernel against the pinned Mathlib (no errors, no admitted steps, standard axioms only). Second checkers: Lean's own replay checker (leanchecker) passed on the rules folder, and nanoda 0.4.26 accepted an export of its 29 listed declarations (7,503 declarations checked).	10
One-row replacement criterion	When can two removed rows be replaced by a single row?	Removing two rows and replacing them by one row succeeds exactly when the missing patterns, taken from the kept rows, never ask for opposite bits in the same column.	PROVED-written	Written proof; the criterion is stated as a written recipe in the check guide.	10
Clique limit	Can a clique of mutually conflicting patterns alone rule out large	The simple conflict-clique shortcut has at most fifteen vertices when at least one row is frozen (and at most sixteen with none), so	PROVED-written	Counting argument; not separately reviewed.	10

Result	Question	Answer	Status	How it was checked	Chapter
	row replacements?	a clique alone cannot refute large replacement scopes.			
Lower bounds on the number of rows	How few rows can a binary strength-four array have, according to published inequalities?	Using the ordinary fractional covering bound and an exact pair-intersection relaxation, the number of rows of a binary strength-four array is at least 47 at widths 201 to 203 and at least 50 at widths 325 to 342. The ordinary fractional bound alone is 16 at width 201. The gaps to the supplied arrays are 98 to 120 rows. These are calculations from the published inequalities (Kleitman-Spencer 1973; Roux 1987; Choi-Kim-Oh 2011/2012), not a new lower-bound technique.	COMPUTED	Exact rational certificates for all 21 cells; derivations not read by anyone outside this project.	10
Support antichain lemma	Can the clause set for a one-column extension contain duplicate or redundant clauses?	Distinct triple-pattern supports of a valid strength-four array form an antichain, so the clause set for a one-column extension has no duplicate or subsumed clauses.	PROVED-written	Proof on paper; not separately reviewed.	10
Strength five	Does a binary strength-five array with 16 columns and 97 rows exist?	Five attempts at a binary strength-five array with 16 columns and 97 rows ended with the builder missing 3 to 7	COMPUTED	Bounded search; no claim about existence.	10

Result	Question	Answer	Status	How it was checked	Chapter
		patterns. This is a bounded negative result.			

The 346th column

Result	Question	Answer	Status	How it was checked	Chapter
Certificate analogue for the 346 search (blocks of rows)	Can the search for a new column be split into blocks of rows that are checked separately?	Four statements hold for splitting the residual constraints of a one-column extension into blocks that each own some rows: a sufficient condition, an exact one (a column extends exactly when every block passes), the fact that a forced edge blocks extension, and monotonicity (enlarging the fixed set keeps residual edges residual).	PROVED-written	Proofs on paper.	11

The strength-four arrays (chapters 1 to 4)

Result	Question	Answer	Status	How it was checked	Chapter
The 29 binary strength-four arrays	Is each of the 29 supplied tables really a binary covering array of strength four?	Each of the 29 arrays shipped in this book (200 to 206, 223 and 325 to 345 columns) is a binary covering array of strength four: in every choice of four columns, all 16 patterns appear in at least one row.	COMPUTED	Every choice of four columns checked exhaustively by two separate programs (one in Python, one in C#); both found all 29 arrays to be covering arrays.	4
Comparison with the pinned table and the living table at 29 widths	Is each of the 29 strength-four arrays smaller than the upper bound listed for its width, in one fixed snapshot of Colbourn's tables and in Smolen's living table?	At each of the 29 widths, the array supplied has fewer rows than the upper bound for that width in the pinned snapshot of Colbourn's tables (margin 3 to 11 rows), and fewer rows than the smallest size listed at that	COMPUTED	Pinned bounds recomputed from the pinned comparison table (a CSV file, not on this site and available on request; the same file is public at the pinned commit of Smolen's ca-tools repository); all 29 agree. Living-table bounds read from Smolen's table at commit 0fc74c021747b6a160d421dca76bba5b6f3e1bf7.	3

Result	Question	Answer	Status	How it was checked	Chapter
		width or wider in Smolen's living ca-tools table at commit 0fc74c0 (margin 1 to 7 rows), under the lookup rule given on the statement page. Still open: the smallest possible size, and arrays outside these two tables.			

The Mirror Route and the Ladder

Result	Question	Answer	Status	How it was checked	Chapter
Maxima of isolated models f(n,d)	What is the largest number of isolated models of a cross-intersecting monotone formula?	Table of the largest number of isolated models f(n,d) for n up to 5, with refutation certificates (RUP) for this table up to n = 11; f(n,1) = 4. (The fresh encodings up to n = 12 that confirm the four-isolated-models theorem are a separate computation.)	COMPUTED	Exhaustive for n <= 6 (see the four-isolated-models theorem); certificates for larger n.	12
The Ladder table	How small can the pair of constraint families be for the tables on the Ladder (8 to 16 rows)?	For tables with 8, 10, 12, 14 and 16 rows (3, 4, 5, 6, 6 old columns), the smallest number of constraints |F|+|G| in a cross-intersecting pair that reproduces all valid colourings is 6 and 10 exactly for 8 and 10 rows, 24 exactly for 14 rows (the exact 14-row minimum), and for 12 and 16 rows lies between 16 and 70 and between 22 and 654 respectively (best pairs found: 3/3, 7/3, 27/43, 15/9, 304/350). The lower bounds 16 and 22 come from the affine model set bound and the earlier cross bound; the four-isolated-models theorem, which is CONDITIONAL, would raise them to 17 and 23.	COMPUTED	Displayed pairs verified over the whole coefficient cubes (up to 4,194,304 settings); the exact minima for 8 and 10 rows and the displayed bounds for the others were rederived. Only 369 of 10,635 listed file fingerprints were rechecked.	12
Critical-clause local lower bound	Does a valid colouring with rigid bits force a lower bound on	If u has p rigid one-bits and q rigid zero-bits, then |F| >= p and |G| >= q, for every u in the fibre of x. The critical-clause idea is	PROVED-written	Reading.	12

Result	Question	Answer	Status	How it was checked	Chapter
	the sizes of the two families?	Paturi, Pudlak and Zane (1997).			
Constant-coefficient elimination	Can the constant coefficient be dropped from the bound?	If a valid colouring is rigid in all n = d-1 non-constant directions and the pair has exactly n edges, both families ignore the constant coefficient.	PROVED-written	Reading.	12
Isolated distance lemma	In how many coordinates can two isolated models differ?	In a monotone positive/negative CNF with cross-intersecting supports and at most n+1 clauses, any two isolated models differ in at most five coordinates.	PROVED-written	Used by the four-isolated-models theorem.	12
Four isolated models	How many isolated models can a small monotone formula with cross-intersecting supports have?	A positive/negative monotone CNF on n variables with at most n+1 clauses and cross-intersecting positive and negative supports has at most four isolated models; if it has four, their bitwise XOR is zero (they form a translate of a 2-dimensional space). The bound is sharp. Consequence, which is conditional in the same way: Ladder lower bounds 17 and 23 for 12 and 16 rows (and 23 for 14 rows, now replaced by the exact 24 of the exact 14-row minimum).	CONDITIONAL	Computer-assisted proof. Confirmed by exhaustive enumeration for n up to 6 (n = 6: 26,324,608,236 clause sets) and by fresh SAT encodings up to n = 12. Eight of the 12 original ten-variable proof files delete unit clauses mid-proof, which a strict proof checker rejects; they must be regenerated without deletions before this is cited as a certified proof. Hence CONDITIONAL.	12
Fixed-gap reduction	Can coordinates that every valid column agrees on be fixed and removed?	For a gap family with p coordinates fixed to 1, q fixed to 0 and a free ones, the minimum mirror edge count equals p+q plus the minimum on the a-	PROVED-written	Reading.	12

Result	Question	Answer	Status	How it was checked	Chapter
		coordinate projection; one extra clause still eliminates the constant coefficient.			
Critical-cover rule	Can an optimal cover of the critical demands give a lower bound?	Replace each positive clause by a contained minimal transversal; an optimal cover of the critical demands lower-bounds each side.	PROVED-written	Stated before it was used.	12
Restricted minimum 24	What is the smallest total number of constraints for the 14-row table when one pattern per complementary pair is allowed?	For the 14-row table, with one non-constant pattern per complementary pair and both constants kept, the minimum edge count is 24.	COMPUTED	256^3 triples, 1,946 unsatisfiable cases; the restriction is lifted by the exact 14-row minimum.	12
The exact minimum is 24 (15 in F, 9 in G)	What is the exact smallest total number of constraints for the 14-row table?	For the 14-row, 6-old-column table with the 22-bit degree-2 decoder and all six valid colourings kept, the minimum total number of constraints is 24: 15 in F and 9 in G.	COMPUTED	Proof by 256 cases x 3 allocations (768 subcases), all unsatisfiable; 256 strict RUP refutations (1,181,768 additions) covering the 768 subcases; of the 768 subproofs, 553 were accepted by two checkers and 215 by one checker only; a 24-edge witness exhausted over 4,194,304 settings. A second check redid 7 checks.	12
Partial-isolation distance lemma	How far apart can models be that are isolated in only some directions?	Models isolated only in the n designated directions, with at most n+h clauses, differ in at most 3h+2 coordinates.	PROVED-written	A reading found 0 mathematical gaps and 4 wording gaps; the exact 14-row minimum does not depend on it.	12

Result	Question	Answer	Status	How it was checked	Chapter
From a cross-intersecting pair to an intersecting family; list selector	Can a cross-intersecting pair be turned into an intersecting family, and at what cost?	A cross-intersecting pair (F,G) becomes an intersecting family L whose proper colourings match the gaps of (F,G); a list of r patterns costs 1 + rd + binom(r,2) edges. The cost is polynomial in the supplied list only.	PROVED-written	2,700 lists checked. Standard final step (Gaur-Makino; Eiter-Makino-Gottlob).	12
Distinct-row decoder failure	Can every valid new column be written with the fixed quadratic decoder?	On the 16 distinct rows of the 4-cube, a valid new column lies outside the image of the fixed quadratic decoder: 7,842 valid columns, only 370 of them in the image. So the stronger promise 'keep every valid colouring with this decoder' is impossible in general.	PROVED-written	Exhaustive.	12
Witness selection	Does any pair of families have exactly a selected set of gaps?	No pair (F,G) of any size has exactly a given selected gap family in the cases examined; rail-code gap sets need |F|+|G| >= 2^floor(r/2).	PROVED-written	Rechecked (wording fixes: 53 failing selections, 22 distinct triples).	12
No general reduction to duality was established	Is there a general polynomial reduction from strength-3 extension to non-duality?	A general polynomial reduction from strength-3 extension to non-duality was NOT established. The positive results are the unit-row subclass and |B(H_m)| >= 2^binom(m-1,2) with automorphism group S_m.	PROVED-written	Partly computed (m = 4, 5, 6).	12

Result	Question	Answer	Status	How it was checked	Chapter
New predicates and non-uniform auxiliaries	Do new predicates or non-uniform auxiliaries complete the reduction?	A self-dual selector construction and an auxiliary-slice lemma were found; the attempt to complete the reduction did not close.	PROVED-written	Computed fits for small cases.	12

The strength-2 and strength-3 collections (summary)

Result	Question	Answer	Status	How it was checked	Chapter
Strength-2 and strength-3 arrays extending published arrays	How many strength-2 arrays extending published arrays were found and checked?	2,339 strength-2 arrays (alphabets 3 to 18) and a few strength-3 arrays that extend published arrays were found and checked. Where the base array is known, they extend arrays built by the CAs package; for the older collection and the alphabet-7 pair the base is not known. They are not presented as new constructions.	COMPUTED	The 1,631 arrays of the later collection were each rechecked (211,758,899 column pairs); the 706 strength-2 arrays of the older collection had one check each; the alphabet-7 pair had a second check by a separate program. See chapter 15 for which are which.	15
Plateaus: arrays one step behind the published table	At which widths does a published table stay one step behind the arrays we found?	In the published table, the number of rows steps up at certain widths. Just before each step, arrays with the old number of rows were found for a few more columns: 26 such plateaus, 445 below-table array cells and 310 cells equal to the table.	COMPUTED	Counts recomputed from the lists of arrays.	15
Plateaus extended by one more column	Which plateaus could be extended by one more column?	Of the 13 plateaus tried (every plateau of the older collection except the 4-symbol, 42-row one), three were extended by one more column: (2,5,53) from 53 to 54 columns, (2,5,55) from 58 to 59, (2,5,65) from 221 to 222.	COMPUTED	Six arrays rechecked.	15
Balanced extension: a negative result	Did a balance condition on the symbol counts gain any column on the plateau tips?	No extra column was found on any of the 13 plateau tips tried (as in the previous result) when the new column was required to have balanced symbol counts.	COMPUTED	Why it fails was not tested.	15
A solver's 'no column fits' is never	Is a solver's answer that no column fits	A solver answer that no single column can be appended to a	PROVED-written	Trivial; the whole proof is in chapter 15. About 75 such answers; the	15

Result	Question	Answer	Status	How it was checked	Chapter
a lower bound	ever a lower bound for the smallest possible array?	fixed prefix says nothing about the smallest possible array.		solver proof files are kept for one, available on request.	
Table lookup rule: dropping columns preserves coverage	Why is the bound at a width the smallest row count among entries of that width or wider?	An array with k columns gives arrays with fewer columns by deleting columns; so the upper bound at a width is the smallest row count among table entries at that width or wider. This is the rule used for every comparison in this book.	PROVED-written	Trivial; the whole proof is in chapter 15: deleting columns keeps every remaining set of t columns covered. (Lean covers projection and monotonicity of strength.)	15
The later collection of 1,631 strength-2 arrays	How many arrays of the later collection pass the full check, and how do they compare with the published rows?	1,631 strength-2 arrays (alphabet 4: 545, 5: 471, 6: 462, 7: 153) all pass a full pair check; 124 of them are below and 212 are equal to the published rows we located. This is a source-bounded comparison, not a certification of the current state of the art.	COMPUTED	Every array individually rechecked.	15
The older collection of 720 arrays	What do the 720 arrays of the older collection show against the November 2024 table?	720 arrays (706 strength 2, 14 strength 3) of which 69 are below, 98 equal and 553 above the November 2024 table.	COMPUTED	Counts recomputed from the lists of arrays. The 69 below-table arrays were checked once only, plus a literature check; there is no full second check.	15
The two alphabet-7 arrays 163 x 917 and 169 x 1076	Do 7-symbol strength-2 arrays of 163 x 917 and 169 x 1076 exist?	A 163-row, 917-column and a 169-row, 1076-column 7-symbol strength-2 array exist. The published table we compared with (C. J. Colbourn's covering-array tables, as preserved by U. Groemping and read through C. Smolen's CSV file) lists 164 and 170 rows at these widths; the difference is one row each. All 419,986 and	COMPUTED	Pair checks repeated by a second program. The construction method is not known.	15

Result	Question	Answer	Status	How it was checked	Chapter
		578,350 column pairs were checked, none missing.			

Lean names behind seven of the Lean-proved results, and the bridge file

These seven results state exactly what their Lean theorems cover; the written or computed parts have rows of their own. The Lean names, for readers who build the Lean files, are:

```
Strength-2 extension and monotone non-duality are equivalent
    forward_clutter_reduction, total_reverse_correct, total_reverse_sizes,
Raising lemma: every fixed strength t >= 3
    raised_strength, raised_extends_iff, raised_counts, padding_bound, rais
Residual-row completion rule
    ArraysRule.completion_iff_coloring (lean-rules)
Projection rule
    ArraysRule.projection (lean-rules); ArraysLean.strength_flip_iff (Array
Repeatable growth recurrence (clone and repair)
    ArraysGrowth.clone_repair_preserves, repeatable_growth, growth_row_coun
Clone obstruction
    ArraysGrowth.clone_requires_eight (lean-rules)
Bridge between the three Lean definitions of strength four (lean folder, Ar
    ArraysBridge.growth_covers_iff_strength, ArraysBridge.ca4_iff_strength_
    ArraysBridge.paritySeed_strength_four, ArraysBridge.tiny_not_strength_f
Transfer lemma: outside columns colour every class
    outside_column_transfer, uncolorable_classes_cover, uncolorable_class_c
```

Where to go next

- The Lean theorems behind the Lean-proved rows: the Lean theorem table.
- What failed and what is open: chapter 16, the ledger.

20 Theorems in this book

Theorems in this book

Every theorem and lemma in this book, with its exact statement, status and credit. Maths is written in LaTeX notation, between dollar signs.

The entries follow the book's own order: the register of results (statement/RESULTS.md; chapters 3 to 16), the 48 Lean theorems (statement/LEAN-THEOREMS.md), then the open questions. Statements are the register's statement text, with only the symbols written as LaTeX. The status words are the register's: PROVED-Lean, PROVED-written, COMPUTED, CONDITIONAL, OPEN and, for the ledger, SPECIAL CASES and UNCONDITIONAL BOUND. The Lean names are internal labels, not terms of the problem. Items the register marks REFUTED are claims tested and found false, and are not listed here as results (they are in the ledger, chapter 16).

Computation 1: Comparison with the pinned table and the living table at 29 widths

Credit: Builds on: Charles Colbourn's covering array tables, as preserved by Ulrike Groemping and turned into a CSV file by Christopher Smolen; Smolen's living ca-tools table; the table's entries at widths 325 to 345 are attributed to J. Torres-Jimenez and co-authors. This result (Sonny): computed.

At each of the 29 widths, the array supplied has fewer rows than the upper bound for that width in the pinned snapshot of Colbourn's tables (margin 3 to 11 rows), and fewer rows than the smallest size listed at that width or wider in Smolen's living ca-tools table at commit 0fc74c0 (margin 1 to 7 rows), under the lookup rule given on the statement page. Still open: the smallest possible size, and arrays outside these two tables.

Status: COMPUTED. How it was checked, as the register gives it: Pinned bounds recomputed from the pinned comparison table (a CSV file, not on this site and available on request; the same file is public at the pinned commit

of Smolen's ca-tools repository); all 29 agree. Living-table bounds read from Smolen's table at commit 0fc74c021747b6a160d421dca76bba5b6f3e1bf7.

Proof: chapters/03-the-comparison-table.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-03-the-comparison-table); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 2: The 29 binary strength-four arrays

Credit: Builds on: Paley's quadratic-residue construction, R. E. A. C. Paley (1933); the post-optimization approach of Peyman Nayeri, Charles Colbourn and Goran Konjevod (2009; 2013), which the row-removal-and-repair step follows; the criterion of Janne I. Kokkala (doctoral thesis, Aalto University, 2017, section 3.5) and Hiess (master's thesis, TU Wien, 2024); Christopher Smolen's arrays CA(141; 4, 199, 2) and CA(165; 4, 324, 2) from ca-tools (CC BY 4.0), the starting arrays at widths 200 to 206 and 325 to 328. This result (Sonny): computed. Method step, in the book's words (chapter 10): For 23 of the 29 arrays, rows were removed one at a time and each removal was repaired by changing cells (chapter 5). Chapter 5 describes the step as done in this project: For widths 202 to 206 I went on to 148 rows and 206 columns, then removed rows one at a time, repairing each removal by changing cells in a local search, until 145 rows were left.

Each of the 29 arrays shipped in this book (200 to 206, 223 and 325 to 345 columns) is a binary covering array of strength four: in every choice of four columns, all 16 patterns appear in at least one row.

Status: COMPUTED. How it was checked, as the register gives it: Every choice of four columns checked exhaustively by two separate programs (one in Python, one in C#); both found all 29 arrays to be covering arrays.

Proof: chapters/04-how-the-check-works.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-04-how-the-check-works); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 3: Class lemma: extension is proper two-colouring of the row-class hypergraph (every strength)

Credit: The criterion is known (Kokkala 2017; Hiess 2024); the book claims no priority for it. This result: Sonny.

For a table of strength t, a new column keeps strength t exactly when it takes both values on every class $E(S;x)$ with $|S| = t - 1$. Hence an extension exists exactly when the rows can be two-coloured so that no required class is monochromatic.

Status: PROVED-Lean. How it was checked, as the register gives it: Lean theorems on the class lemma and the colouring form (listed in the Lean theorem table). The criterion is known (Kokkala 2017; Hiess 2024); we claim no priority for it. Checked by the Lean 4.34.1 kernel; the exported proofs of the whole 48-theorem package were also re-checked by a second kernel checker, nanoda (6,174 declarations). Lean's own replay checker (leanchecker) also passed. No lean4lean run.

```
-- ArraysLean.class_lemma
-- ArraysLean.extension_iff_two_coloring
```

Proof: chapters/05-room-for-one-more-column.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-05-room-for-one-more-column); Lean theorems named in the code block above; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 4: Support criterion for strength 4

Credit: Tools: Lean 4; leanchecker; the nanoda kernel checker; lean4lean. This result (Sonny): proved (Lean).

A new column x preserves strength four exactly when, for every choice J of three old columns and every pattern a on J, the rows showing a on J include one with $x = 0$ and one with $x = 1$. (The same statement as the class lemma, one level up.)

Status: PROVED-Lean. How it was checked, as the register gives it: Lean theorem on the support criterion. This is the test used in every search for a new column of a strength-four array. Checked by the Lean 4.34.1 kernel; the exported proofs of the whole 48-theorem package were also re-checked by a second kernel checker, nanoda (6,174 declarations). Lean's own replay checker (leanchecker) also passed. No lean4lean run.

```
-- ArraysLean.support_criterion
```

Proof: chapters/05-room-for-one-more-column.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-05-room-for-one-more-column); Lean theorems named in the code block above; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 5: Normal form of a strength-2 prefix

Credit: Tools: Lean 4. This result (Sonny): proved (written).

Fix one row u. Each column becomes the set T_j of other rows that hold a 1 where u holds a 0 (after flipping columns so that u is all zeros). The table is a binary covering array of strength 2 exactly when the sets T_j form an intersecting antichain (any two sets meet, none contains another). It extends by one column exactly when $\{T_j\}$ is not a maximal intersecting antichain.

Status: PROVED-written. How it was checked, as the register gives it: Proof read and accepted in a separate reading; 2,901 random prefixes (4 to 8 rows) compared with brute force, no disagreement. Lean covers the anchored form.

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Theorem 6: Strength-2 extension reduces to N-1 duality questions

Credit: Tools: Lean 4. This result (Sonny): proved (written).

A binary strength-2 table with N rows extends by one column exactly when, for at least one row w other than the reference row, the transversal family of a derived set family H_1 differs from the minimal members of a second family $H_2^{'}$ (both built from the table). That is $N-1$ monotone-duality questions, each of size polynomial in the table, so strength-2 extension can be decided in quasi-polynomial time (using the Fredman-Khachiyan algorithm).

Status: PROVED-written. How it was checked, as the register gives it: Accepted as a mathematical argument in a separate reading; two seeded exhaustive runs (1,169 and 1,176 random prefixes plus 450 structured ones each) all agree; a further comparison on 2,901 prefixes had 0 mismatches. Lean covers the per-row reduction, not the running-time statement.

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 7: No polynomial NP-hardness reduction for strength-2 extension unless ETH fails

Credit: Builds on: the Exponential Time Hypothesis of Impagliazzo and Paturi (1999, 2001). This result (Sonny): conditional.

If a polynomial-time reduction took signed not-all-equal 3-SAT to strength-2 column extension, then not-all-equal 3-SAT would be solvable in time $2^{o(\log^2 n)}$, which contradicts the Exponential Time Hypothesis (ETH).

Status: CONDITIONAL. How it was checked, as the register gives it: A written implication, valid only if the Exponential Time Hypothesis is true; the implication was found valid in a separate reading. Not an unconditional statement about strength 2.

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Theorem 8: Strength-2 extension and monotone non-duality are equivalent

Credit: Eiter and Gottlob (1995) proved the equivalences between hypergraph saturation and the transversal-hypergraph (duality) problem. In this project's dictionary, a binary strength-2 covering array together with its column complements is a simple self-complemented hypergraph, and it takes one more column exactly when that hypergraph is not saturated. The reverse direction is their Theorem 4.13, crediting Seymour. The dictionary to covering arrays, the explicit constructions and sizes, and the Lean proofs: Sonny.

Deciding whether a binary strength-2 covering array can take one more column (CA-Ext) and deciding whether two monotone Boolean functions given as families are not dual (CO-DUAL) can each be turned into the other, with the answer preserved. The reverse construction has $n_0 + 3$ rows; the correctness and the size bounds of both constructions are proved in Lean.

Status: PROVED-Lean. How it was checked, as the register gives it: Lean proves the correctness and the stated row, column and size bounds of both reductions (48-theorem package: theorems 24, 25, 37, 41 and 42 of the Lean theorem table). The time needed to carry the constructions out is a separate result, in the next row of this register (PROVED-written). Also compared exhaustively on 163 antichains, 27,524 pairs and 2,192 prefixes with no mismatch. The reverse direction is Eiter and Gottlob's Theorem 4.13 (credit: Seymour). Checked by the Lean 4.34.1 kernel; the exported proofs of the whole 48-theorem package were also re-checked by a second kernel checker, nanoda (6,174 declarations). Lean's own replay checker (leanchecker) also passed. No lean4lean run.

```
-- ArraysLean.reverse_reduction
-- ArraysLean.reverse_counts
-- ArraysLean.forward_clutter_reduction
-- ArraysLean.total_reverse_correct
-- ArraysLean.total_reverse_sizes
```

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; Lean theorems named in the code block above; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 9: The reductions between strength-2 extension and monotone non-duality run in polynomial time

Credit: Tools: Lean 4. This result (Sonny): how it was checked, as the register gives it proved (written).

Both constructions of the previous result can be carried out in polynomial time; the forward direction, from a covering array with N rows and k columns to a pair of families, takes $O(k^2 N^3)$ bit operations. Together with the previous result, each of the two problems reduces to the other in polynomial time, with the answer preserved.

Status: PROVED-written. How it was checked, as the register gives it: Written argument only; the running time is not formalised in Lean. The correctness and size bounds it relies on are the Lean-proved ones of the previous row.

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 10: One-row antichain extension lemma

Credit: This result (Sonny): proved (written).

A one-row table $A(F)$ built from a family F is a valid strength-2 prefix exactly when F is a pairwise-intersecting antichain; it extends exactly when an incomparable transversal exists. If F is contained in its own transversal family $\mathrm{Tr}(F)$, it extends exactly when $\mathrm{Tr}(F)$ differs from F.

Status: PROVED-written. How it was checked, as the register gives it: Checked exhaustively inside the comparison for the equivalence of strength-2 extension and duality (no mismatch).

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 11: Maximality of an antichain is completeness of a list of minimal hitting sets

Credit: This result (Sonny): proved (written).

An intersecting antichain F is maximal exactly when $T_0 = \mathrm{Tr}(B)$; there is a reverse construction with fresh points p, q and a coatom C.

Status: PROVED-written. How it was checked, as the register gives it: Three separate readings; no code.

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 12: Local minimality is checkable; completeness is not (12 versus 14)

Credit: Tools: Lean 4. This result (Sonny): proved (written).

In a valid $\mathrm{CA}(8; 2, 10, 2)$ with exactly four extensions, the list of 12 minimal sets passes every local minimality check, yet the full transversal family has 14 members, so the list is incomplete.

Status: PROVED-written. How it was checked, as the register gives it: Hand derivation; not re-run. Lean covers the local test.

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 13: Fixed rank spread is tractable

Credit: This result (Sonny): proved (written).

For a fixed rank spread w, the decision is polynomial. Still open: a fixed-parameter tractability bound in w for this method.

Status: PROVED-written. How it was checked, as the register gives it: Algorithm on paper; not implemented.

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 14: Exact count of extensions

Credit: This result (Sonny): proved (written).

The number of extensions of the prefix is $2W(F)$, where
$W(F) = 2^n - |\text{up}(F)| - |\text{down}(F)| + m$.

Status: PROVED-written. How it was checked, as the register gives it: On paper; not implemented.

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 15: A rank-counting sufficient certificate

Credit: This result (Sonny): proved (written).

If $C_t < \binom{n}{t}$, an incomparable t-set exists, so the prefix extends. This is sufficient only; a counter-example shows it is not necessary.

Status: PROVED-written. How it was checked, as the register gives it: A counter-example is given in the note.

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 16: Structured duality: the reverse reduction is one instance

Credit: Eiter and Gottlob (1995; Theorem 4.13, crediting Seymour): the reduction from duality to saturation (an extension question for the covering array, in this project's dictionary), and the equivalence of the two. The explicit construction with n+3 rows and b+t+1 columns, as one forced branch of the all-prefix reduction: Sonny.

For the structured form, 'general non-duality' holds exactly when the prefix extends, with $N = n + 3$ rows and $k = b + t + 1$ columns; with the reduction of strength-2 extension to duality questions this gives polynomial-time Turing equivalence.

Status: PROVED-written. How it was checked, as the register gives it: Reading only; no computation. The note itself calls it a reconciliation of known results.

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 17: An easy region: the Lambda test

Credit: This result (Sonny): proved (written).

Let $\Lambda = \sum_j \left[2^{1-|S_j|} + 2^{1-|T_j|}\right]$ (the sum over columns j). If $\Lambda < 1$, an extension exists and can be constructed deterministically in polynomial time.

Status: PROVED-written. How it was checked, as the register gives it: Algorithm on paper; not implemented.

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 18: A shortcut that fails: a maximal intersecting antichain need not be self-dual

Credit: This result (Sonny): proved (written).

Take $F =$ all four triples of $\{a, b, c, d\}$. F is a maximal intersecting antichain, yet $\mathrm{Tr}(F)$ is the six pairs, so the 5×4 prefix does not extend. 'Maximal' does not mean 'equal to its own transversal family'.

Status: PROVED-written. How it was checked, as the register gives it: A four-point example, checked by hand and by computer.

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 19: Stuck arrays: small unsatisfiable cores of seven 'no more columns' cases

Credit: Tools: SAT solvers. This result (Sonny): computed.

Seven arrays (3 at strength 2, 4 at strength 3; 5 to 8 symbols) for which a solver answered 'no column fits' each have a small unsatisfiable core; strength-2 cores use all pairs of columns, strength-3 cores are sparse (5 to 18 pairs). These are statements about a fixed prefix, never lower bounds on the smallest possible array.

Status: COMPUTED. How it was checked, as the register gives it: All seven were run again with a second solver (7 of 7 unsatisfiable); two cores were recomputed. Not every proof file was kept.

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 20: The strength-2 equivalence with monotone duality

Credit: Sonny; the credits page gives no separate line for this result.

The reverse direction (non-duality gives an extension) is word for word Eiter and Gottlob's Theorem 4.13. The forward direction (any covering array gives a duality instance) is the part we did not find stated in the sources we read. One lemma (a step about the one-row table $A(F)$ built from a family F) needs the hypothesis that $A(F)$ is a strength-2 covering array; without it, the lemma fails for 310 families on 4 points.

Status: PROVED-written. How it was checked, as the register gives it: About 1.5 million cases compared with brute force, 0 disagreements; a separate check reproduced all 400 numbers.

Proof: chapters/06-strength-two-and-duality.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-06-strength-two-and-duality); proofs/strength-two-and-duality.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 21: Pair-class formulation

Credit: Sonny; the credits page gives no separate line for this result.

A binary strength-3 table extends by one column exactly when its rows can be coloured 0/1 so that every pair class $E(a, b; x, y)$ (the rows with value x in column a and y in column b) contains both colours. Equivalently, a new column keeps strength 3 exactly when it takes both values on every pair class. This is the $t = 3$ case of the class lemma (which Lean proves in general).

Status: PROVED-written. How it was checked, as the register gives it: Proof read and accepted in a separate reading. The criterion itself appears in the literature (Kokkala 2017 thesis, section 3.5; Hiess 2024 thesis); we claim no priority for it.

Proof: chapters/07-strength-three-pair-classes.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-07-strength-three-pair-classes); proofs/strength-three-pair-classes.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 22: Strength-3 extension is in NP; the classes form an antichain

Credit: This result (Sonny): proved (written).

A proposed column is checked in $O(Nk^2)$ time. The pair classes have at least 2 rows each, and a table with k columns has exactly $4\binom{k}{2}$ distinct classes, none contained in another.

Status: PROVED-written. How it was checked, as the register gives it: Proof read and accepted in a separate reading.

Proof: chapters/07-strength-three-pair-classes.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-07-strength-three-pair-classes); proofs/strength-three-pair-classes.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 23: Padding lemma for intersecting Sperner hypergraphs

Credit: This result (Sonny): proved (written).

For an intersecting Sperner hypergraph J, the table $A_2(J)$ (with $N = n + dQ(s)$ rows and $k = 2s$ columns) is extendible exactly when J is properly two-colourable.

Status: PROVED-written. How it was checked, as the register gives it: Proof read and accepted; the 22 x 6 worked example has 420,244 valid new columns out of 2^22, and 300 random columns were compared, 0 mismatches.

Proof: chapters/07-strength-three-pair-classes.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-07-strength-three-pair-classes); proofs/strength-three-pair-classes.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Theorem 24: Complement of duality reduces to strength-3 extension

Credit: Sonny; the credits page gives no separate line for this result.

There is a polynomial-time map f such that $f(J, G)$ is always a binary strength-3 array and G differs from $\mathrm{Tr}(J)$ exactly when $f(J, G)$ extends. So the complement of the duality problem reduces to strength-3 extension.

Status: PROVED-written. How it was checked, as the register gives it: Proof read and accepted in a separate reading; 60 instances compared with brute force, 0 mismatches. We claim no priority: the link between duality and colouring intersecting hypergraphs is in Elbassioni 2016 and Kokkala 2017.

Proof: chapters/07-strength-three-pair-classes.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-07-strength-three-pair-classes); proofs/strength-three-pair-classes.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 25: Decision orientation

Credit: This result (Sonny): proved (written).

For an intersecting Sperner hypergraph H, H is not two-colourable if and only if $H = \mathrm{Tr}(H)$; G differs from $\mathrm{Tr}(H)$ exactly when F is two-colourable exactly when $A(F)$ is extendible.

Status: PROVED-written. How it was checked, as the register gives it: Accepted on reading.

Proof: chapters/07-strength-three-pair-classes.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-07-strength-three-pair-classes); proofs/strength-three-pair-classes.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 26: Exact formulation through bad transversals

Credit: This result (Sonny): proved (written).

Strength-3 extension is possible exactly when $B(H)$ differs from $\mathrm{Tr}(H)$, exactly when the duality question for $(H, B(H))$ has answer NO, where $B(H)$ is the list of bad minimal transversals.

Status: PROVED-written. How it was checked, as the register gives it: Accepted on reading; benchmark counts recomputed (cube 6/4/2, parity 4/4/0, m = 4: 1393/1393/0).

Proof: chapters/07-strength-three-pair-classes.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-07-strength-three-pair-classes); proofs/strength-three-pair-classes.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 27: The bad-transversal list can be superpolynomial

Credit: This result (Sonny): proved (written).

For the incidence arrays H_m, $|B(H_m)| \geq 2^{\binom{m-1}{2}}$ on distinct-row NO arrays, a growth of $2^{\Theta(\sqrt{L})}$ in the table size L. This blocks only the direct route 'list $B(H)$, then use duality'.

Status: PROVED-written. How it was checked, as the register gives it: Accepted; values 8, 64, 1024 for m = 4, 5, 6 recomputed.

Proof: chapters/07-strength-three-pair-classes.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-07-strength-three-pair-classes); proofs/strength-three-pair-classes.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 28: Union-bound test

Credit: This result (Sonny): proved (written).

If $W = \sum_E 2^{1-|E|}$ (the sum over classes E) is less than 1, an extension exists and can be found by a deterministic polynomial algorithm. Example: a 64×32 table has $W = 31/512$.

Status: PROVED-written. How it was checked, as the register gives it: Checked on the example.

Proof: chapters/07-strength-three-pair-classes.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-07-strength-three-pair-classes); proofs/strength-three-pair-classes.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 29: Lovasz Local Lemma test (Moser-Tardos)

Credit: Sonny; the credits page gives no separate line for this result.

If weights x_E satisfy $2^{1-|E|} \le x_E \prod_F (1 - x_F)$ (the product over overlapping F), an extension exists and a randomised resampling procedure finds it. Example: a 28×3 table has union-bound sum $1153/1024 > 1$ yet passes this test.

Status: PROVED-written. How it was checked, as the register gives it: Recomputed on the example. Relies on the published Moser-Tardos theorem (arXiv:0903.0544), taken as given.

Proof: chapters/07-strength-three-pair-classes.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-07-strength-three-pair-classes); proofs/strength-three-pair-classes.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 30: The shared design A_r (the [8,4,4] example)

Credit: This result (Sonny): proved (written).

The 16×8 design has exactly 56 extensions, and all 16 one-row deletions have 0; the lift fails for $r \geq 4$.

Status: PROVED-written. How it was checked, as the register gives it: All 65,536 columns enumerated; a recheck agreed.

Proof: chapters/07-strength-three-pair-classes.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-07-strength-three-pair-classes); proofs/strength-three-pair-classes.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 31: Shared-column equality stars: the part that works

Credit: This result (Sonny): proved (written).

Polynomial-size binary strength-3 arrays can force arbitrarily many occurrence copies of a variable to carry equal extension bits. A single reference shared by two disjoint stars fails strength 3.

Status: PROVED-written. How it was checked, as the register gives it: 31 arrays checked (23 as promised, 8 that fail as predicted); the 13 x 3 star has 534 extensions.

Proof: chapters/07-strength-three-pair-classes.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-

07-strength-three-pair-classes); proofs/strength-three-pair-classes.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 32: Worked example tables for chapters 5 to 7

Credit: This result (Sonny): computed.

A set of small YES and NO benchmark tables for extension, for example a 22×6 YES table (420,244 valid columns), a 21×6 NO table, an odd-parity 8×4 table, a 16×8 table with 56 extensions and a 28×3 table for the local-lemma test.

Status: COMPUTED. How it was checked, as the register gives it: Each worked example is small enough to recheck by exhaustive search.

Proof: chapters/07-strength-three-pair-classes.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-07-strength-three-pair-classes); proofs/strength-three-pair-classes.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 33: The easy side of strength-3 extension

Credit: This result (Sonny): proved (written).

Every proper column projection of a strength-3 array extends (the omitted column is the witness). With b the maximum intersection of two classes, extension is decidable in $O(2^{8b}Nk^2)$ time; $b = 1$ is graph bipartiteness. The smallest NO array has 8 rows (the rows x, y, z, $x + y + z$). Every class has between 2 and $N - 6$ rows.

Status: PROVED-written. How it was checked, as the register gives it: All 30 catalogue arrays tested were row-irreducible; reading plus computation.

Proof: chapters/07-strength-three-pair-classes.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-07-strength-three-pair-classes); proofs/strength-three-pair-classes.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 34: A polynomial subclass: unit-row arrays

Credit: This result (Sonny): proved (written).

For arrays whose extra rows are unit rows, the table extends exactly when both constant columns are present; then exactly two extension columns exist.

Status: PROVED-written. How it was checked, as the register gives it: Described in the source as an elementary subclass.

Proof: chapters/07-strength-three-pair-classes.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-07-strength-three-pair-classes); proofs/strength-three-pair-classes.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 35: Transfer lemma: outside columns colour every class

Credit: Tools: Lean 4; leanchecker; the nanoda kernel checker; lean4lean. This result (Sonny): proved (Lean).

Every old column outside the column set that defines a pair class is a proper colouring of that class. Hence a family of classes that cannot be split by any new column has defining column sets covering every old column, and so contains at least $\lceil k/2 \rceil$ classes.

Status: PROVED-Lean. How it was checked, as the register gives it: Lean proves the finite combinatorial statements; the sharpness of the bound on the family J_m is written only. Also covered by a reading of the whole document on the eighteen walls (no false implication found). Checked by the Lean

4.34.1 kernel; the exported proofs of the whole 48-theorem package were also re-checked by a second kernel checker, nanoda (6,174 declarations). Lean's own replay checker (leanchecker) also passed. No lean4lean run.

```
-- ArraysLean.repeated_row_lemma
-- ArraysLean.outside_column_transfer
-- ArraysLean.uncolorable_classes_cover
-- ArraysLean.uncolorable_class_count
```

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; Lean theorems named in the code block above; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 36: Private-support obstruction

Credit: This result (Sonny): how it was checked, as the register gives it computed.

If each source edge is enforced on its own column support and the supports are disjoint, the Fano plane together with a disjoint triple cannot be represented.

Status: PROVED-written. How it was checked, as the register gives it: Computed on the named sources.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 37: A logical bridge gadget (hypergraph level only)

Credit: This result (Sonny): proved (written).

Bridge vertices give exactly 2^d extensions per source colouring. This is a statement about hypergraphs, not an array construction.

Status: PROVED-written. How it was checked, as the register gives it: Counts checked on an example.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 38: Off-edge label certificates and safe padding

Credit: This result (Sonny): proved (written).

Extension projections equal the source colourings when a label certificate exists; deciding whether labels exist is itself NP-complete (it needs 3-uniform hypergraph two-colouring), so labels conceal the source problem.

Status: PROVED-written. How it was checked, as the register gives it: Counts checked on four examples.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 39: Washing: padding and duplicate rows free the colours

Credit: This result (Sonny): proved (written).

If a fixed old-column frame B is already extendible, appending any s rows keeps it extendible: any extension of B, followed by any of the 2^s assignments on the new rows, is an extension.

Status: PROVED-written. How it was checked, as the register gives it: Counts checked on four examples.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 40: Stars cannot be attached to a previous occurrence array

Credit: This result (Sonny): proved (written).

Preserved occurrence codes at Hamming distance greater than s cannot be tied by an equality network supported on at most s old columns.

Status: PROVED-written. How it was checked, as the register gives it: Counts on the Fano plane and triples.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 41: Complementary hubs enforce the wrong not-all-equal relation

Credit: This result (Sonny): proved (written).

Exactly $2^{n'} - 2m$ consistent core assignments extend; all three unsatisfiable test sources become YES, so this gadget does not encode the source problem.

Status: PROVED-written. How it was checked, as the register gives it: Eight sources, zero false negatives.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-

08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 42: Uniform shared flags give not-all-equal on all variables

Credit: This result (Sonny): proved (written).

On a shared flag c_x the construction enforces NAE($x_1, \ldots, x_n$): it accepts exactly the $2^n - 2$ non-constant assignments. This is the wrong relation for a clause-by-clause encoding.

Status: PROVED-written. How it was checked, as the register gives it: Eight sources checked.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Theorem 43: Conditional exact-preservation theorem for supplied gadget data

Credit: This result (Sonny): proved (written).

If gadget data B and one choice per source clause satisfy all the stated equations, the safe-padding construction is a strength-3 array and its projection of the extensions equals the set of assignments with NAE on every clause. The hypotheses could not be satisfied in the cases tried (finite refutations, 607,275 search nodes).

Status: PROVED-written. How it was checked, as the register gives it: The implication is proved; the equations could not be satisfied in the cases tried.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in

statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 44: Auxiliary selector encoding is not ordinary duality

Credit: This result (Sonny): proved (written).

With a shared auxiliary variable z the witnesses are preserved but $\mathrm{DUAL}(F, G) = \mathrm{NO}$ on every promised array: a spurious gap always remains.

Status: PROVED-written. How it was checked, as the register gives it: Six benchmarks and seven two-vertex hypergraphs.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 45: Compression lower bounds for the forbidden DNF

Credit: This result (Sonny): proved (written).

Under the cofactor hypothesis, $|G_A| \geq |\mathrm{Tr}(K_A)|$. This is a property of a proposed reduction, not a statement about ETH.

Status: PROVED-written. How it was checked, as the register gives it: 65,536 formulas checked; the bad-transversal count over 2^20 subsets.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 46: Changing both duality families does not help

Credit: This result (Sonny): computed.

For the one kind of construction this wall concerns (its scope in the book's table of walls: template): a private-intersection family always has a false source gap; repairing G alone needs at least 3^M terms.

Status: COMPUTED. How it was checked, as the register gives it: Written up in the document on the eighteen walls.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 47: Interacting auxiliaries: a fitted compiler and the 167x9 escape

Credit: This result (Sonny): computed.

Exact source projections on all eight test sources, but the construction is not a polynomial-time reduction (it uses the full solution sets).

Status: COMPUTED. How it was checked, as the register gives it: 31 arrays checked; one further check of our own only.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 48: Uniform clause constructions fail

Credit: This result (Sonny): computed.

Exact projection passes 1 of 28 primary and 0 of 28 auxiliary cases, and all 13 unsatisfiable sources give YES.

Status: COMPUTED. How it was checked, as the register gives it: 68 matrices rechecked; the second recipe was not tested blind.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 49: An orientation recipe

Credit: This result (Sonny): how it was checked, as the register gives it computed.

The recipe repairs the sources $\{123, 124, 345\}$ but is exact in only 6 of 28 cases. No second recipe was tried.

Status: COMPUTED. How it was checked, as the register gives it: Computed on the named sources.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 50: The eighteen walls

Credit: This result (Sonny): proved (written).

Eighteen grouped failure results explaining why the obvious routes from small gadgets to a hardness proof fail. A separate reading found 'No false mathematical implication' and states that the document does not establish a universal obstruction to a hardness reduction. It counts 110 results: 22 universal walls, 41 template-specific failures and 47 finite observations.

Status: PROVED-written. How it was checked, as the register gives it: Read through (not checked by a proof assistant): 59 proved, 28 computed and 7 assumed among the 94 rows that carry an evidence label. Replaced as a route by the hub-row reduction.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 51: Separate star tests are not a characterisation

Credit: This result (Sonny): proved (written).

Requiring each star to be separately feasible gives only $\bigwedge_j \exists c_j$; extension needs one common c for all stars. (The 21×6 triangle fixture: all six stars are YES but the table is NO.)

Status: PROVED-written. How it was checked, as the register gives it: The 21 x 6 example checked.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Theorem 52: No fixed VC-dimension bound; two-family gap feasibility is NP-complete

Credit: This result (Sonny): proved (written).

Pair classes can shatter arbitrarily many rows; the general two-family gap problem is NP-complete but does not by itself build an array.

Status: PROVED-written. How it was checked, as the register gives it: Accepted on reading.

Proof: chapters/08-the-eighteen-walls.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-08-the-eighteen-walls); proofs/eighteen-walls.md; register entry in

statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Theorem 53: Strength-3 one-column extension is NP-complete (the hub-row reduction)

Credit: Builds on: the NP-completeness of not-all-equal satisfiability, Schaefer (1978); the question as posed by Garn and Simos (2017). This result (Sonny): proved (written).

Deciding whether a binary strength-3 covering array can take one more column, with no new rows, is NP-complete. The reduction starts from positive 3-uniform not-all-equal satisfiability with m clauses and writes a strength-3 table with $k = 4m + 2$ columns and $N = O(m^3)$ rows (every row explicit). The output always has strength 3, and it extends if and only if the source is satisfiable. Membership in NP is the membership-in-NP result.

Status: PROVED-written. How it was checked, as the register gives it: Lean proves the two correctness statements (the output has strength 3 on every input; it extends exactly when the source is satisfiable); the Lean source is in the hardness folder, which is not online (available on request; check guide, recipe 10); the Lean package that is online does not contain these two theorems. Polynomial running time is proved in writing only, not in Lean (the polynomial-time note shipped with the book), and the written proof is shipped too. NP-completeness of the source problem (Schaefer 1978) is taken as given. 1,359 small source puzzles agree with brute force (36 unsatisfiable); 1,035 of them (all 1,024 clause families on five variables and 11 named sources, including the Fano plane) were rerun with no mismatch. A proof rebuilt from the theorem statement alone found no error. Lean 4.34.1 compiler and kernel only, run against the pinned Mathlib (no errors, no admitted steps, only the standard axioms). Second checkers: Lean's own replay checker (leanchecker) passed on the hardness folder, and nanoda 0.4.26 accepted an export of its 12 listed declarations (5,195 declarations checked). The status is PROVED-written because the polynomial-time argument and the NP-completeness of the source problem are not formalised.

```
-- OneColumnExtension.Input.output_strength
-- OneColumnExtension.Input.output_extends_iff
```

```
-- name: OneColumnExtension.Input.output_strength
-- lean-hardness/OneColumnExtension.lean, line 741
theorem output_strength (H : Input) : StrengthThree H.Output.entry
```

```
-- name: OneColumnExtension.Input.output_extends_iff
-- lean-hardness/OneColumnExtension.lean, line 751
theorem output_extends_iff (H : Input) : Extends H.Output.entry ↔ H.Source
```

Proof: chapters/09-strength-three-is-np-complete.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-09-strength-three-is-np-complete); proofs/hardness-variants.md; hub-row/PROOF.md; hub-row/POLYNOMIAL-TIME.md; Lean theorems named in the code block above; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 54: Raising lemma: every fixed strength t >= 3

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker; lean4lean. This result (Sonny): proved (Lean).

Let M have strength t with at least t columns. The raised array M' (add a column w; keep the old rows with $w = 0$; add every row with at most $t + 1$ zeros with $w = 1$, each twice) has strength $t + 1$, and M' extends at strength $t + 1$ if and only if M extends at strength t. For a fixed t the number of padding rows is bounded by a polynomial in the number of columns (it grows like k^{t+1}).

Status: PROVED-Lean. How it was checked, as the register gives it: Lean proves the raising lemma and its size bounds (48-theorem package: theorems 9 to 13 of the Lean theorem table). The composition with the hub-row reduction, which gives NP-completeness at every fixed strength, is the next

row of this register (PROVED-written). Rechecked on 202 small cases plus 19 full strength-four reductions. Checked by the Lean 4.34.1 kernel; the exported proofs of the whole 48-theorem package were also re-checked by a second kernel checker, nanoda (6,174 declarations). Lean's own replay checker (leanchecker) also passed. No lean4lean run.

```
-- ArraysLean.raised_strength
-- ArraysLean.raised_extends_iff
-- ArraysLean.raised_counts
-- ArraysLean.padding_bound
-- ArraysLean.raised_row_bound
```

Proof: chapters/09-strength-three-is-np-complete.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-09-strength-three-is-np-complete); proofs/hardness-variants.md; Lean theorems named in the code block above; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 55: NP-completeness at every fixed strength t >= 3 (raising lemma composed with the hub-row reduction)

Credit: Builds on: the NP-completeness of not-all-equal satisfiability, Schaefer (1978), through the hub-row reduction. Tools: Lean 4. This result (Sonny): proved (written).

Composed with the hub-row reduction, one-column extension of a binary covering array of every fixed strength $t \geq 3$ is NP-complete, by a polynomial reduction whose padding grows like k^{t+1} (polynomial for fixed t only; nothing is claimed for t growing with the input).

Status: PROVED-written. How it was checked, as the register gives it: The raising lemma and its size bounds are proved in Lean (previous row). The composition inherits the written polynomial-time argument of the hub-row reduction and the NP-completeness of its source problem (Schaefer 1978), neither of which is formalised. Rechecked on 202 small cases plus 19 full strength-four reductions.

Proof: chapters/09-strength-three-is-np-complete.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-09-strength-three-is-np-complete); proofs/hardness-variants.md; hub-row/PROOF.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Theorem 56: Distinct-row version stays NP-complete

Credit: This result (Sonny): proved (written).

The hub construction with every padding row written once (coloured by the parity of its number of zeros), twin variables removed beforehand, and a fixed small array output when at most one clause survives, gives a strength-3 table whose rows are all different and which extends exactly when the source is satisfiable.

Status: PROVED-written. How it was checked, as the register gives it: All 1,359 sources rechecked (strength, distinct rows, exact equivalence; the 36 negative sources stay negative).

Proof: chapters/09-strength-three-is-np-complete.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-09-strength-three-is-np-complete); proofs/hardness-variants.md; hub-row/PROOF.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 57: Thin arrays and both properties at once

Credit: This result (Sonny): proved (written).

The padding can be shrunk from about m^3 rows to $O(\log m)$, so the hard tables have about as many rows as columns, and with twin rows the rows stay distinct and the tables stay thin at every fixed strength.

Status: PROVED-written. How it was checked, as the register gives it: Read line by line in a separate reading (no mathematical error); computer checks

agree with brute force on more than 2,000 small puzzles. Not yet rechecked a second time.

Proof: chapters/09-strength-three-is-np-complete.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-09-strength-three-is-np-complete); proofs/hardness-variants.md; hub-row/PROOF.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 58: Residual-row completion rule

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Let F be a fixed block of binary rows on a set of columns, and M the set of 4-column partial words that no row of F shows. Join two members of M when they ask for opposite bits in some common column. Then r further binary rows make F a strength-four array exactly when this conflict graph has a proper colouring with r colours. (Each colour class is merged into one row.)

Status: PROVED-Lean. How it was checked, as the register gives it: Lean theorem on the completion rule in its general partial-word form: the rule is stated for the set M of missing four-column patterns. The step from an array file to that set is a separate result, in the next row of this register (PROVED-written). Checked by the Lean 4.34.1 compiler and kernel against the pinned Mathlib (no errors, no admitted steps, standard axioms only). Second checkers: Lean's own replay checker (leanchecker) passed on the rules folder, and nanoda 0.4.26 accepted an export of its 29 listed declarations (7,503 declarations checked).

```
-- ArraysRule.completion_iff_coloring
-- ArraysRule.residual_completion
-- ArraysRule.residual_coloring
```

```
-- name: ArraysRule.completion_iff_coloring
-- lean/Rule.lean, line 12
theorem completion_iff_coloring {D C : Type} (P : D → C → Option Bool) (r :
    Completes P r ↔ Colorable P r
```

```
-- name: ArraysRule.residual_completion
-- lean/Rule.lean, line 46
theorem residual_completion {D C S : Type} (P : D → C → Option Bool)
    (fixed : S → C → Bool) (r : Nat) :
    (∃ rows : Fin r → C → Bool, ∀ d,
      (∃ s, Realizes (fixed s) (P d)) ∨ (∃ i, Realizes (rows i) (P d))) ↔
    Completes (fun d : { d // ¬ ∃ s, Realizes (fixed s) (P d) } => P d.val
```

```
-- name: ArraysRule.residual_coloring
-- lean/Rule.lean, line 66
theorem residual_coloring {D C S : Type} (P : D → C → Option Bool)
    (fixed : S → C → Bool) (r : Nat) :
    (∃ rows : Fin r → C → Bool, ∀ d,
      (∃ s, Realizes (fixed s) (P d)) ∨ (∃ i, Realizes (rows i) (P d))) ↔
    Colorable (fun d : { d // ¬ ∃ s, Realizes (fixed s) (P d) } => P d.val
```

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; Lean theorems named in the code block above; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 59: From array bytes to missing partial words

Credit: Tools: Lean 4. This result (Sonny): how it was checked, as the register gives it proved (written).

For a block F of rows read from an array file, listing for every choice of four columns the patterns that some row shows gives exactly the set M of four-

column partial words that no row of F shows, to which the completion rule applies.

Status: PROVED-written. How it was checked, as the register gives it: Written step; not formalised in Lean. Tested on 960 exhaustive comparisons over 240 seeded systems, all agreeing.

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 60: Block-of-columns criterion

Credit: Tools: Lean 4. This result (Sonny): how it was checked, as the register gives it proved (written).

For a block of new columns the necessary and sufficient conditions include every choice of j new and $4-j$ old columns, for $j = 1, 2, 3, 4$. Checking $j = 1$ alone is not enough.

Status: PROVED-written. How it was checked, as the register gives it: Written classification of four-column sets by the number of new columns; no Lean.

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 61: Projection rule

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker; lean4lean. This result (Sonny): proved (Lean).

Restricting a strength-four array to any set of distinct columns keeps strength four; permuting columns and complementing a column also keep coverage.

Status: PROVED-Lean. How it was checked, as the register gives it: Lean theorems: the projection theorem (rules folder; no axioms) covers restricting to any set of distinct columns and permuting columns; the theorem on flipping columns (theorem 26 of the Lean theorem table, 48-theorem package) covers complementing a column. These two theorems use different Lean definitions of strength four; the bridge file in the Lean folder proves that the definitions agree. Checked by the Lean 4.34.1 compiler and kernel against the pinned Mathlib (no errors, no admitted steps, standard axioms only). Second checkers: for the rules folder, Lean's own replay checker (leanchecker) and nanoda 0.4.26 (an export of its 29 listed declarations, 7,503 declarations checked); for the 48-theorem package, nanoda (6,174 declarations) and leanchecker. No lean4lean run.

```
-- ArraysRule.projection
-- ArraysLean.strength_flip_iff
```

```
-- name: ArraysRule.projection
-- lean/Rule.lean, line 77
theorem projection {R C K : Type} (A : R → C → Bool) (f : K → C)
    (hf : ∀ a b, f a = f b → a = b) (h : CA4 A) :
    CA4 (fun r k => A r (f k))
```

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; Lean theorems named in the code block above; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 62: Graph-colouring form of fixed-row replacement

Credit: This result (Sonny): proved (written).

Replacing a chosen set of removed rows by m new rows can succeed exactly when the conflict graph of the patterns missing from the kept rows has a proper colouring with at most m colours.

Status: PROVED-written. How it was checked, as the register gives it: A special case of the residual-row completion rule.

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 63: Width ceiling for a fixed number of rows

Credit: Tools: Lean 4; leanchecker; the nanoda kernel checker; lean4lean. This result (Sonny): proved (Lean).

A binary covering array of strength 2, or of strength 4 with at least four columns, with N rows has at most 2^{N-1} columns.

Status: PROVED-Lean. How it was checked, as the register gives it: Lean theorems on the width bounds (48-theorem package). Checked by the Lean 4.34.1 kernel; the exported proofs of the whole 48-theorem package were also re-checked by a second kernel checker, nanoda (6,174 declarations). Lean's own replay checker (leanchecker) also passed. No lean4lean run.

```
-- ArraysLean.normalized_injective
-- ArraysLean.width_bound
-- ArraysLean.strength_four_width_bound
```

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; Lean theorems named in the code block above; register entry in

statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 64: A tempting invariant fails

Credit: This result (Sonny): computed.

On the 16-row full factorial, adding parity and then the identical parity column again gives two columns that both pass the seed support test yet never disagree; so 'keep splitting every triple' is not preserved by adding columns one at a time.

Status: COMPUTED. How it was checked, as the register gives it: Exhaustive over all 65,536 candidate columns of the 16-row table: only parity and its complement extend it.

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 66: Repeatable growth recurrence (clone and repair)

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

A binary strength-four array with $k \geq 4$ columns can always be grown by one column by cloning a column and adding repair rows: if the other $k-1$ columns are given distinct m-bit labels ($m = \lceil \log_2(k-1) \rceil$ bits are enough), the grown array has $k+1$ columns and $N + 4m + 4$ rows. Repeating this gives $k_{s+1} = k_s + 1$ and $N_{s+1} = N_s + 4m + 4$.

Status: PROVED-Lean. How it was checked, as the register gives it: Lean theorems on cloning a column and repairing, and on the row and column counts, for any labelling of the other columns by distinct m-bit words. The choice m = ceil(log2(k-1)) and the comparison with the pinned table are the

next row of this register (COMPUTED). Checked by the Lean 4.34.1 compiler and kernel against the pinned Mathlib (no errors, no admitted steps, standard axioms only). Second checkers: Lean's own replay checker (leanchecker) passed on the rules folder, and nanoda 0.4.26 accepted an export of its 29 listed declarations (7,503 declarations checked).

```
-- ArraysGrowth.clone_repair_preserves
-- ArraysGrowth.repeatable_growth
-- ArraysGrowth.growth_row_count
-- ArraysGrowth.growth_column_count
```

```
-- name: ArraysGrowth.clone_repair_preserves
-- lean/Growth.lean, line 22
theorem clone_repair_preserves {R T C : Type} [Fintype C] [DecidableEq C]
    (A : R → Option C → Bool) (B : T → C → Bool)
    (ha : Covers A 4) (hb : Covers B 2) : Covers (cloneRepair A B) 4
```

```
-- name: ArraysGrowth.repeatable_growth
-- lean/Growth.lean, line 155
theorem repeatable_growth {R C : Type} [Fintype C] [DecidableEq C]
    (A : R → Option C → Bool) (ha : Covers A 4)
    (label : C → Fin m → Bool) (hi : Function.Injective label) :
    Covers (cloneRepair A (labelRows label)) 4
```

```
-- name: ArraysGrowth.growth_row_count
-- lean/Growth.lean, line 161
theorem growth_row_count {R : Type} [Fintype R] (m : Nat) :
    Fintype.card (Sum R (Bool × Sum Bool (Fin m × Bool))) =
      Fintype.card R + (4*m+4)
```

```
-- name: ArraysGrowth.growth_column_count
-- lean/Growth.lean, line 165
theorem growth_column_count {C : Type} [Fintype C] :
    Fintype.card (Sum Bool C) = Fintype.card (Option C) + 1
```

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-

10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; Lean theorems named in the code block above; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 68: Clone obstruction

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Copying a column needs at least eight repair rows.

Status: PROVED-Lean. How it was checked, as the register gives it: Lean theorem that the eight patterns demanded by a copy need eight different rows (the copy case). The complement case is the next row of this register (PROVED-written). Checked by the Lean 4.34.1 compiler and kernel against the pinned Mathlib (no errors, no admitted steps, standard axioms only). Second checkers: Lean's own replay checker (leanchecker) passed on the rules folder, and nanoda 0.4.26 accepted an export of its 29 listed declarations (7,503 declarations checked).

```
-- ArraysGrowth.clone_demands_injective
-- ArraysGrowth.clone_requires_eight
```

```
-- name: ArraysGrowth.clone_demands_injective
-- lean/Growth.lean, line 87
theorem clone_demands_injective : Function.Injective CloneDemand
```

```
-- name: ArraysGrowth.clone_requires_eight
-- lean/Growth.lean, line 99
theorem clone_requires_eight {R : Type} [Fintype R]
    (rows : R → Fin 4 → Bool)
    (h : ∀ d : Bool × Bool × Bool, ∃ r, rows r = CloneDemand d) :
    8 ≤ Fintype.card R
```

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-

10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; Lean theorems named in the code block above; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 69: Complementing a column needs at least eight repair rows

Credit: Tools: Lean 4. This result (Sonny): proved (written).

Complementing a column (adding it with 0 and 1 swapped) needs at least eight repair rows.

Status: PROVED-written. How it was checked, as the register gives it: Follows from the copy case by flipping the column. Written argument only: no Lean statement joins the Lean theorem on the copy case with the Lean theorem on flipping a column.

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 70: No universal one-row extension (parity 16x5)

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The claim that every strength-four array admits a new column with at most one added row is false: take the 16 even-parity rows on five bits. Special seed families are not excluded.

Status: PROVED-Lean. How it was checked, as the register gives it: Lean theorem on the 16 even-parity rows. Checked by the Lean 4.34.1 compiler and kernel against the pinned Mathlib (no errors, no admitted steps, standard axioms only). Second checkers: Lean's own replay checker

(leanchecker) passed on the rules folder, and nanoda 0.4.26 accepted an export of its 29 listed declarations (7,503 declarations checked).

```
-- ArraysGrowth.parity_cannot_add_column_and_one_row
```

```
-- name: ArraysGrowth.parity_cannot_add_column_and_one_row
-- lean/Parity.lean, line 30
theorem parity_cannot_add_column_and_one_row (z : Fin 32) (x : Fin 16 → Boo
    ¬ Covers (oneRowExtension z x b) 4
```

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; Lean theorems named in the code block above; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 71: A full linear parity block is too large

Credit: C. R. Rao (1947), Rao's lower bound $1 + k + k(k-1)/2$ on the rows of a binary orthogonal array of strength 4; a linear block in which every four coefficient vectors are independent, taken in full, is such an array with 2^m rows. That step, the count at width 345 and the Lean check: Sonny.

A binary linear block in which every four coefficient vectors are independent needs at least 65,536 rows at width 345 (the pinned bound is 174 there).

Status: PROVED-Lean. How it was checked, as the register gives it: Lean theorems on the full linear block and on width 345. Checked by the Lean 4.34.1 compiler and kernel against the pinned Mathlib (no errors, no admitted steps, standard axioms only). Second checkers: Lean's own replay checker (leanchecker) passed on the rules folder, and nanoda 0.4.26 accepted an export of its 29 listed declarations (7,503 declarations checked).

```
-- ParityCost.full_linear_block_bound
-- ParityCost.width_345_requires_65536
```

```
-- name: ParityCost.full_linear_block_bound
-- lean-rules/ParityCost.lean, line 56
theorem full_linear_block_bound {k d : Nat} (h : Fin k → (Fin d → ZMod 2))
    (hind : NoSmallDependency h) : 1+k+k.choose 2 ≤ 2^d
```

```
-- name: ParityCost.width_345_requires_65536
-- lean-rules/ParityCost.lean, line 74
theorem width_345_requires_65536 {d : Nat} (h : Fin 345 → (Fin d → ZMod 2))
    (hind : NoSmallDependency h) : 65536 ≤ 2^d
```

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; Lean theorems named in the code block above; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 74: One-row replacement criterion

Credit: This result (Sonny): how it was checked, as the register gives it proved (written).

Removing two rows and replacing them by one row succeeds exactly when the missing patterns, taken from the kept rows, never ask for opposite bits in the same column.

Status: PROVED-written. How it was checked, as the register gives it: Written proof; the criterion is stated as a written recipe in the check guide.

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 75: Clique limit

Credit: This result (Sonny): proved (written).

The simple conflict-clique shortcut has at most fifteen vertices when at least one row is frozen (and at most sixteen with none), so a clique alone cannot refute large replacement scopes.

Status: PROVED-written. How it was checked, as the register gives it: Counting argument; not separately reviewed.

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 76: Lower bounds on the number of rows

Credit: Sonny; the credits page gives no separate line for this result.

Using the ordinary fractional covering bound and an exact pair-intersection relaxation, the number of rows of a binary strength-four array is at least 47 at widths 201 to 203 and at least 50 at widths 325 to 342. The ordinary fractional bound alone is 16 at width 201. The gaps to the supplied arrays are 98 to 120 rows. These are calculations from the published inequalities (Kleitman-Spencer 1973; Roux 1987; Choi-Kim-Oh 2011/2012), not a new lower-bound technique.

Status: COMPUTED. How it was checked, as the register gives it: Exact rational certificates for all 21 cells; derivations not read by anyone outside this project.

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; register entry in statement/RESULTS.md

(https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 77: Support antichain lemma

Credit: This result (Sonny): proved (written).

Distinct triple-pattern supports of a valid strength-four array form an antichain, so the clause set for a one-column extension has no duplicate or subsumed clauses.

Status: PROVED-written. How it was checked, as the register gives it: Proof on paper; not separately reviewed.

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 80: Strength five

Credit: This result (Sonny): computed.

Five attempts at a binary strength-five array with 16 columns and 97 rows ended with the builder missing 3 to 7 patterns. This is a bounded negative result.

Status: COMPUTED. How it was checked, as the register gives it: Bounded search; no claim about existence.

Proof: chapters/10-strength-four-rules-and-bounds.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-10-strength-four-rules-and-bounds); proofs/strength-four-rules.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 83: Certificate analogue for the 346 search (blocks of rows)

Credit: This result (Sonny): proved (written).

Four statements hold for splitting the residual constraints of a one-column extension into blocks that each own some rows: a sufficient condition, an exact one (a column extends exactly when every block passes), the fact that a forced edge blocks extension, and monotonicity (enlarging the fixed set keeps residual edges residual).

Status: PROVED-written. How it was checked, as the register gives it: Proofs on paper.

Proof: chapters/11-the-346th-column.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-11-the-346th-column); proofs/strength-four-rules.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 85: Maxima of isolated models f(n,d)

Credit: This result (Sonny): computed.

Table of the largest number of isolated models $f(n,d)$ for n up to 5, with refutation certificates (RUP) for this table up to $n = 11$; $f(n,1) = 4$. (The fresh encodings up to $n = 12$ that confirm the four-isolated-models theorem are a separate computation.)

Status: COMPUTED. How it was checked, as the register gives it: Exhaustive for n <= 6 (see the four-isolated-models theorem); certificates for larger n.

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md

(https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 86: The Ladder table

Credit: This result (Sonny): computed.

For tables with 8, 10, 12, 14 and 16 rows (3, 4, 5, 6, 6 old columns), the smallest number of constraints $|F| + |G|$ in a cross-intersecting pair that reproduces all valid colourings is 6 and 10 exactly for 8 and 10 rows, 24 exactly for 14 rows (the exact 14-row minimum), and for 12 and 16 rows lies between 16 and 70 and between 22 and 654 respectively (best pairs found: 3/3, 7/3, 27/43, 15/9, 304/350). The lower bounds 16 and 22 come from the affine model set bound and the earlier cross bound; the four-isolated-models theorem, which is CONDITIONAL, would raise them to 17 and 23.

Status: COMPUTED. How it was checked, as the register gives it: Displayed pairs verified over the whole coefficient cubes (up to 4,194,304 settings); the exact minima for 8 and 10 rows and the displayed bounds for the others were rederived. Only 369 of 10,635 listed file fingerprints were rechecked.

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 87: Critical-clause local lower bound

Credit: Builds on: the critical-clause idea of Paturi, Pudlák and Zane (1997). This result (Sonny): proved (written).

If u has p rigid one-bits and q rigid zero-bits, then $|F| \geq p$ and $|G| \geq q$, for every u in the fibre of x. The critical-clause idea is Paturi, Pudlak and Zane (1997).

Status: PROVED-written. How it was checked, as the register gives it: Reading.

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 88: Constant-coefficient elimination

Credit: This result (Sonny): proved (written).

If a valid colouring is rigid in all $n = d - 1$ non-constant directions and the pair has exactly n edges, both families ignore the constant coefficient.

Status: PROVED-written. How it was checked, as the register gives it: Reading.

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 89: Isolated distance lemma

Credit: This result (Sonny): proved (written).

In a monotone positive/negative CNF with cross-intersecting supports and at most $n + 1$ clauses, any two isolated models differ in at most five coordinates.

Status: PROVED-written. How it was checked, as the register gives it: Used by the four-isolated-models theorem.

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 90: Four isolated models

Credit: Sonny; the credits page gives no separate line for this result.

A positive/negative monotone CNF on n variables with at most $n + 1$ clauses and cross-intersecting positive and negative supports has at most four isolated models; if it has four, their bitwise XOR is zero (they form a translate of a 2-dimensional space). The bound is sharp. Consequence, which is conditional in the same way: Ladder lower bounds 17 and 23 for 12 and 16 rows (and 23 for 14 rows, now replaced by the exact 24 of the exact 14-row minimum).

Status: CONDITIONAL. How it was checked, as the register gives it: Computer-assisted proof. Confirmed by exhaustive enumeration for n up to 6 ($n = 6$: 26,324,608,236 clause sets) and by fresh SAT encodings up to $n = 12$. Eight of the 12 original ten-variable proof files delete unit clauses mid-proof, which a strict proof checker rejects; they must be regenerated without deletions before this is cited as a certified proof. Hence CONDITIONAL.

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 91: Fixed-gap reduction

Credit: This result (Sonny): proved (written).

For a gap family with p coordinates fixed to 1, q fixed to 0 and a free ones, the minimum mirror edge count equals $p + q$ plus the minimum on the a-coordinate projection; one extra clause still eliminates the constant coefficient.

Status: PROVED-written. How it was checked, as the register gives it: Reading.

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 92: Critical-cover rule

Credit: This result (Sonny): proved (written).

Replace each positive clause by a contained minimal transversal; an optimal cover of the critical demands lower-bounds each side.

Status: PROVED-written. How it was checked, as the register gives it: Stated before it was used.

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 93: Restricted minimum 24

Credit: This result (Sonny): computed.

For the 14-row table, with one non-constant pattern per complementary pair and both constants kept, the minimum edge count is 24.

Status: COMPUTED. How it was checked, as the register gives it: 256^3 triples, 1,946 unsatisfiable cases; the restriction is lifted by the exact 14-row minimum.

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 94: The exact minimum is 24 (15 in F, 9 in G)

Credit: This result (Sonny): computed.

For the 14-row, 6-old-column table with the 22-bit degree-2 decoder and all six valid colourings kept, the minimum total number of constraints is 24: 15 in F and 9 in G.

Status: COMPUTED. How it was checked, as the register gives it: Proof by 256 cases x 3 allocations (768 subcases), all unsatisfiable; 256 strict RUP refutations (1,181,768 additions) covering the 768 subcases; of the 768 subproofs, 553 were accepted by two checkers and 215 by one checker only; a 24-edge witness exhausted over 4,194,304 settings. A second check redid 7 checks.

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 95: Partial-isolation distance lemma

Credit: This result (Sonny): proved (written).

Models isolated only in the n designated directions, with at most $n + h$ clauses, differ in at most $3h + 2$ coordinates.

Status: PROVED-written. How it was checked, as the register gives it: A reading found 0 mathematical gaps and 4 wording gaps; the exact 14-row minimum does not depend on it.

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 96: From a cross-intersecting pair to an intersecting family; list selector

Credit: Sonny; the credits page gives no separate line for this result.

A cross-intersecting pair (F, G) becomes an intersecting family L whose proper colourings match the gaps of (F, G); a list of r patterns costs $1 + rd + \binom{r}{2}$ edges. The cost is polynomial in the supplied list only.

Status: PROVED-written. How it was checked, as the register gives it: 2,700 lists checked. Standard final step (Gaur-Makino; Eiter-Makino-Gottlob).

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 97: Distinct-row decoder failure

Credit: This result (Sonny): proved (written).

On the 16 distinct rows of the 4-cube, a valid new column lies outside the image of the fixed quadratic decoder: 7,842 valid columns, only 370 of them in the image. So the stronger promise 'keep every valid colouring with this decoder' is impossible in general.

Status: PROVED-written. How it was checked, as the register gives it: Exhaustive.

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 98: Witness selection

Credit: This result (Sonny): proved (written).

No pair (F, G) of any size has exactly a given selected gap family in the cases examined; rail-code gap sets need $|F| + |G| \geq 2^{\lfloor r/2 \rfloor}$.

Status: PROVED-written. How it was checked, as the register gives it: Rechecked (wording fixes: 53 failing selections, 22 distinct triples).

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 99: No general reduction to duality was established

Credit: This result (Sonny): proved (written).

A general polynomial reduction from strength-3 extension to non-duality was NOT established. The positive results are the unit-row subclass and

$|B(H_m)| \geq 2^{\binom{m-1}{2}}$ with automorphism group S_m.

Status: PROVED-written. How it was checked, as the register gives it: Partly computed (m = 4, 5, 6).

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 100: New predicates and non-uniform auxiliaries

Credit: This result (Sonny): how it was checked, as the register gives it computed.

A self-dual selector construction and an auxiliary-slice lemma were found; the attempt to complete the reduction did not close.

Status: PROVED-written. How it was checked, as the register gives it: Computed fits for small cases.

Proof: chapters/12-the-mirror-route-and-the-ladder.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-12-the-mirror-route-and-the-ladder); proofs/ladder-lemmas.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Theorem 101: Ternary strength-2 one-column extension is NP-complete

Credit: This result (Sonny): proved (written).

Deciding whether a ternary (three-symbol) strength-2 covering array can take one more column, with no new rows, is NP-complete.

Status: PROVED-written. How it was checked, as the register gives it: Rebuilt from the written proof alone: 7,404 small source puzzles against two

complete solvers, 71 negative, no disagreement. Rows can repeat when twin vertices occur, and the argument does not use distinct rows.

Proof: chapters/13-other-kinds-of-tables.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-13-other-kinds-of-tables); proofs/ternary-strength-two.md; register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 102: No CA(12;3,9,2) with two constant rows

Credit: Tools: drat-trim (Heule, Hunt and Wetzler); SAT solvers. This result (Sonny): computed.

No binary strength-3 covering array with 9 columns and 12 rows contains two constant rows.

Status: COMPUTED. How it was checked, as the register gives it: Proof by a SAT solver, checked with drat-trim (verdict s VERIFIED); not yet rechecked a second time by anyone else.

Proof: chapters/14-more-arrays.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-14-more-arrays); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 103: CA(22;2,9,4) with cyclic symmetry

Credit: This result (Sonny): computed.

A 22-row, 9-column, 4-symbol strength-2 covering array exists, found with a cyclic-shift assumption. It matches the 22 in the tables, whose construction the package could not locate.

Status: COMPUTED. How it was checked, as the register gives it: Two programs (Python and Node); rebuilt byte for byte. The array is shipped in the book.

Proof: chapters/14-more-arrays.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-14-more-arrays); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 104: A non-extendible CA(12;3,6,2)

Credit: Proved earlier by: Colbourn, Keri, Rivas Soriano, Schlage-Puchta (Discrete Applied Mathematics 158 (2010)). This result (Sonny): computed.

A balanced 12-row strength-3 binary array with 6 columns and 12 distinct rows cannot take a seventh binary column (all 4096 candidates checked). It is already known: representative 7 in the listing of Colbourn, Keri, Rivas Soriano and Schlage-Puchta, Discrete Applied Mathematics 158 (2010).

Status: COMPUTED. How it was checked, as the register gives it: A rediscovery of a published listing; no priority is claimed. The array is shipped in the book.

Proof: chapters/14-more-arrays.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-14-more-arrays); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 108: Strength-2 and strength-3 arrays extending published arrays

Credit: Builds on: published base arrays; where the base is known, arrays built by the constructions of Ulrike Groemping's CAs package for R. Tools: the CAs package for R (Groemping). This result (Sonny): computed.

2,339 strength-2 arrays (alphabets 3 to 18) and a few strength-3 arrays that extend published arrays were found and checked. Where the base array is known, they extend arrays built by the CAs package; for the older collection and the alphabet-7 pair the base is not known. They are not presented as new constructions.

Status: COMPUTED. How it was checked, as the register gives it: The 1,631 arrays of the later collection were each rechecked (211,758,899 column pairs); the 706 strength-2 arrays of the older collection had one check each; the alphabet-7 pair had a second check by a separate program. See chapter 15 for which are which.

Proof: chapters/15-the-strength-two-collection.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-15-the-strength-two-collection); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 109: Plateaus: arrays one step behind the published table

Credit: This result (Sonny): computed.

In the published table, the number of rows steps up at certain widths. Just before each step, arrays with the old number of rows were found for a few more columns: 26 such plateaus, 445 below-table array cells and 310 cells equal to the table.

Status: COMPUTED. How it was checked, as the register gives it: Counts recomputed from the lists of arrays.

Proof: chapters/15-the-strength-two-collection.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-15-the-strength-two-collection); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 110: Plateaus extended by one more column

Credit: This result (Sonny): computed.

Of the 13 plateaus tried (every plateau of the older collection except the 4-symbol, 42-row one), three were extended by one more column: $(2, 5, 53)$ from 53 to 54 columns, $(2, 5, 55)$ from 58 to 59, $(2, 5, 65)$ from 221 to 222.

Status: COMPUTED. How it was checked, as the register gives it: Six arrays rechecked.

Proof: chapters/15-the-strength-two-collection.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-15-the-strength-two-collection); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 111: Balanced extension: a negative result

Credit: This result (Sonny): computed.

No extra column was found on any of the 13 plateau tips tried (as in the previous result) when the new column was required to have balanced symbol counts.

Status: COMPUTED. How it was checked, as the register gives it: Why it fails was not tested.

Proof: chapters/15-the-strength-two-collection.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-15-the-strength-two-collection); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 112: A solver's 'no column fits' is never a lower bound

Credit: This result (Sonny): proved (written).

A solver answer that no single column can be appended to a fixed prefix says nothing about the smallest possible array.

Status: PROVED-written. How it was checked, as the register gives it: Trivial; the whole proof is in chapter 15. About 75 such answers; the solver proof files are kept for one, available on request.

Proof: chapters/15-the-strength-two-collection.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-15-the-strength-two-collection); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Lemma 113: Table lookup rule: dropping columns preserves coverage

Credit: Tools: Lean 4. This result (Sonny): proved (written).

An array with k columns gives arrays with fewer columns by deleting columns; so the upper bound at a width is the smallest row count among table entries at that width or wider. This is the rule used for every comparison in this book.

Status: PROVED-written. How it was checked, as the register gives it: Trivial; the whole proof is in chapter 15: deleting columns keeps every remaining set of t columns covered. (Lean covers projection and monotonicity of strength.)

Proof: chapters/15-the-strength-two-collection.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-15-the-strength-two-collection); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 114: The later collection of 1,631 strength-2 arrays

Credit: This result (Sonny): computed.

1,631 strength-2 arrays (alphabet 4: 545, 5: 471, 6: 462, 7: 153) all pass a full pair check; 124 of them are below and 212 are equal to the published rows we located. This is a source-bounded comparison, not a certification of the current state of the art.

Status: COMPUTED. How it was checked, as the register gives it: Every array individually rechecked.

Proof: chapters/15-the-strength-two-collection.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-15-the-strength-two-collection); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 115: The older collection of 720 arrays

Credit: This result (Sonny): computed.

720 arrays (706 strength 2, 14 strength 3) of which 69 are below, 98 equal and 553 above the November 2024 table.

Status: COMPUTED. How it was checked, as the register gives it: Counts recomputed from the lists of arrays. The 69 below-table arrays were checked once only, plus a literature check; there is no full second check.

Proof: chapters/15-the-strength-two-collection.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-15-the-strength-two-collection); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 116: The two alphabet-7 arrays 163 x 917 and 169 x 1076

Credit: Sonny; the credits page gives no separate line for this result.

A 163-row, 917-column and a 169-row, 1076-column 7-symbol strength-2 array exist. The published table we compared with (C. J. Colbourn's covering-array tables, as preserved by U. Groemping and read through C. Smolen's CSV file) lists 164 and 170 rows at these widths; the difference is one row each. All 419,986 and 578,350 column pairs were checked, none missing.

Status: COMPUTED. How it was checked, as the register gives it: Pair checks repeated by a second program. The construction method is not known.

Proof: chapters/15-the-strength-two-collection.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-15-the-strength-two-collection); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 117: Two special branches of the normal form

Credit: This result (Sonny): proved (written).

(1) A cell of size two reduces extension to T_0 differing from $\mathrm{Tr}(B)$; (2) two identical rows reduce it to G differing from $\mathrm{Tr}(G)$. Both are covered by the reduction of strength-2 extension to duality questions, which covers every prefix. The identical-row case also needs an extra free element and must not be generalised without it.

Status: SPECIAL CASES. How it was checked, as the register gives it: The working note itself says that its general reduction (the reduction of strength-2 extension to duality questions) covers both branches.

Proof: chapters/16-the-ledger.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-16-the-ledger); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 118: Affine model set, with Ladder bounds 16 and 22

Credit: This result (Sonny): proved (written).

A satisfiable CNF with n clauses and only isolated models has an affine model set, so the number of models is a power of two. It gives Ladder lower bounds 16 and 22 for 12 and 16 rows; the four-isolated-models theorem gives the stronger bounds 17 and 23. Since the four-isolated-models theorem is CONDITIONAL, 16 and 22 remain the lower bounds that do not depend on it.

Status: UNCONDITIONAL BOUND. How it was checked, as the register gives it: The stronger bounds 17 and 23 are CONDITIONAL; 16 and 22 remain the unconditional bounds until the proofs of the four-isolated-models theorem are regenerated.

Proof: chapters/16-the-ledger.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-16-the-ledger); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Proposition 119: Tight cross bound of two isolated assignments

Credit: This result (Sonny): proved (written).

A cross-intersecting monotone CNF with n clauses has at most two isolated satisfying assignments. The four-isolated-models theorem, which is CONDITIONAL, gives a stronger bound; until the four-isolated-models theorem's proofs are regenerated, the bounds this gives (6, 10, 16, 22, 22) remain the unconditional Ladder lower bounds at 8, 10, 12 and 16 rows; at 14 rows its 22 is replaced by the exact minimum 24.

Status: UNCONDITIONAL BOUND. How it was checked, as the register gives it: A stronger, CONDITIONAL bound comes from the four-isolated-models theorem.

Proof: chapters/16-the-ledger.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-16-the-ledger); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Computation 120: Counts for the older collection

Credit: This result (Sonny): computed.

The older strength-2 collection has 720 arrays: 69 below the table, 98 equal and 553 above (frontier not yet checked).

Status: COMPUTED. How it was checked, as the register gives it: Counted from the list of arrays (statement/strength2-arrays-listing.csv).

Proof: chapters/16-the-ledger.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-16-the-ledger); register entry in statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Theorem 122: Lean theorem 1 of 48, class lemma

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Appending a column preserves the stated strength exactly when it splits every lower-dimensional row class.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "When can a finished table of strength t take one more column without adding any row?"; register result it supports: Class lemma: extension is proper two-colouring of the row-class hypergraph (every strength); Support criterion for strength 4; Pair-class formulation.

```
-- name: ArraysLean.class_lemma
-- ArraysLean/Core.lean, line 23
theorem class_lemma {R C : Type*} (M : R → C → Bool) (c : R → Bool) (s : ℕ
    (hM : Strength M (s + 1)) : Strength (append M c) (s + 1) ↔ Splits M c
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 123: Lean theorem 2 of 48, extension iff two coloring

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

An extension exists exactly when a two-coloring splits all required row classes.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "When can a finished table of strength t take one more column without adding any row?"; register result it supports: Class lemma: extension is proper two-colouring of the row-class hypergraph (every strength); Pair-class formulation.

```
-- name: ArraysLean.extension_iff_two_coloring
-- ArraysLean/Core.lean, line 56
theorem extension_iff_two_coloring {R C : Type*} (M : R → C → Bool) (s : ℕ
    (hM : Strength M (s + 1)) : Extends M (s + 1) ↔ ∃ c, Splits M c s
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 124: Lean theorem 3 of 48, support criterion

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

A strength-four column extension is equivalent to both colors occurring in every specified three-column class.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-

checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "At strength four, what exactly must a new column satisfy so that the table keeps strength four?"; register result it supports: Support criterion for strength 4.

```
-- name: ArraysLean.support_criterion
-- ArraysLean/Core.lean, line 60
theorem support_criterion {R C : Type*} (M : R → C → Bool) (c : R → Bool)
    (hM : Strength M 4) : Strength (append M c) 4 ↔
      ∀ S : Finset C, S.card = 3 → ∀ x : C → Bool,
        (∃ r, (∀ j ∈ S, M r j = x j) ∧ c r = false) ∧
        (∃ r, (∀ j ∈ S, M r j = x j) ∧ c r = true)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 125: Lean theorem 4 of 48, strength mono

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Strength decreases to any smaller order when the column count supports the larger order.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Why is the bound at a width the smallest row count among entries of that width or wider?"; register result it supports: Table lookup rule: dropping columns preserves coverage (supporting).

```
-- name: ArraysLean.strength_mono
-- ArraysLean/Core.lean, line 74
theorem strength_mono {R C : Type*} [Fintype C] (M : R → C → Bool)
    {s t : ℕ} (hst : s ≤ t) (hcard : t ≤ Fintype.card C)
    (hM : Strength M t) : Strength M s
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 126: Lean theorem 5 of 48, strength on small set

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

A strength-t array covers every assignment on a set of at most t columns under the stated size bound.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "A step used by other theorems."; register result it supports: supporting lemma.

```
-- name: ArraysLean.strength_on_small_set
-- ArraysLean/Core.lean, line 84
theorem strength_on_small_set {R C : Type*} [Fintype C] (M : R → C → Bool)
    {t : ℕ} (hcard : t ≤ Fintype.card C) (hM : Strength M t)
    (S : Finset C) (hS : S.card ≤ t) (x : C → Bool) : Covers M S x
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 127: Lean theorem 6 of 48, normalized injective

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Distinct columns of a strength-two array remain distinct after normalization at a selected row.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "How many columns can a binary strength-four array with N rows have at most?"; register result it supports: Width ceiling for a fixed number of rows; Normal form of a strength-2 prefix (supporting).

```
-- name: ArraysLean.normalized_injective
-- ArraysLean/Width.lean, line 11
theorem normalized_injective {R C : Type*} (M : R → C → Bool) (r0 : R)
    (hM : Strength M 2) : Function.Injective (normalized M r0)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 128: Lean theorem 7 of 48, width bound

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

A finite strength-two Boolean array with a selected row has at most two to the remaining row count columns.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-

checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "How many columns can a binary strength-four array with N rows have at most?"; register result it supports: Width ceiling for a fixed number of rows.

```
-- name: ArraysLean.width_bound
-- ArraysLean/Width.lean, line 30
theorem width_bound {R C : Type*} [Fintype R] [Fintype C]
    (M : R → C → Bool) (r0 : R) (hM : Strength M 2) :
    Fintype.card C ≤ 2 ^ (Fintype.card R - 1)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 129: Lean theorem 8 of 48, strength four width bound

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

A strength-four Boolean array with at least four columns satisfies the same width bound.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "How many columns can a binary strength-four array with N rows have at most?"; register result it supports: Width ceiling for a fixed number of rows.

```
-- name: ArraysLean.strength_four_width_bound
-- ArraysLean/Width.lean, line 40
theorem strength_four_width_bound {R C : Type*} [Fintype R] [Fintype C]
    (M : R → C → Bool) (r0 : R) (hk : 4 ≤ Fintype.card C)
    (hM : Strength M 4) : Fintype.card C ≤ 2 ^ (Fintype.card R - 1)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 130: Lean theorem 9 of 48, raised strength

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The defined raising construction increases strength by one.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Does the hardness hold at every fixed strength from 3 up?"; register result it supports: Raising lemma: every fixed strength t >= 3.

```
-- name: ArraysLean.raised_strength
-- ArraysLean/Raising.lean, line 17
theorem raised_strength {R C : Type*} [Fintype C] [DecidableEq C] (M : R →
    (t : ℕ) (hM : Strength M t) : Strength (raised M t) (t + 1)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 131: Lean theorem 10 of 48, raised extends iff

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The raised array extends at the next order exactly when the original array extends at its order.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Does the hardness hold at every fixed strength from 3 up?"; register result it supports: Raising lemma: every fixed strength t >= 3.

```
-- name: ArraysLean.raised_extends_iff
-- ArraysLean/Raising.lean, line 47
theorem raised_extends_iff {R C : Type*} [Fintype C] [DecidableEq C] (M : F
    (s : ℕ) (hM : Strength M (s + 1)) :
    Extends (raised M (s + 1)) (s + 2) ↔ Extends M (s + 1)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 132: Lean theorem 11 of 48, raised counts

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The raised construction has one extra column and the stated exact padded row count.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Does the hardness hold at every fixed strength from 3 up?"; register result it supports: Raising lemma: every fixed strength t >= 3.

```
-- name: ArraysLean.raised_counts
-- ArraysLean/Raising.lean, line 101
theorem raised_counts {R C : Type*} [Fintype R] [Fintype C] [DecidableEq C]
    Fintype.card (Option C) = Fintype.card C + 1 ∧
    Fintype.card (R ⊕ (Padding C t × Bool)) =
      Fintype.card R + 2 * Fintype.card (Padding C t)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 133: Lean theorem 12 of 48, padding bound

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The number of padding objects is bounded by the stated polynomial for fixed order.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Does the hardness hold at every fixed strength from 3 up?"; register result it supports: Raising lemma: every fixed strength t >= 3.

```
-- name: ArraysLean.padding_bound
-- ArraysLean/RaisingSize.lean, line 7
theorem padding_bound (C : Type*) [Fintype C] (t : ℕ) :
    Fintype.card (Padding C t) ≤ (Fintype.card C + 1) ^ (t + 1)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 134: Lean theorem 13 of 48, raised row bound

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The raised construction obeys the stated row-count upper bound.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Does the hardness hold at every fixed strength from 3 up?"; register result it supports: Raising lemma: every fixed strength t >= 3.

```
-- name: ArraysLean.raised_row_bound
-- ArraysLean/RaisingSize.lean, line 45
theorem raised_row_bound {R C : Type*} [Fintype R] [Fintype C] (t : ℕ) :
    Fintype.card (R ⊕ (Padding C t × Bool)) ≤
      Fintype.card R + 2 * (Fintype.card C + 1) ^ (t + 1)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 135: Lean theorem 14 of 48, minimal hit subset

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Every hitting set contains an inclusion-minimal hitting subset.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "How does an

antichain being maximal relate to a list of minimal hitting sets being complete?"; register result it supports: Maximality of an antichain is completeness of a list of minimal hitting sets (supporting).

```
-- name: ArraysLean.minimal_hit_subset
-- ArraysLean/Hypergraph.lean, line 38
theorem minimal_hit_subset {V : Type*} (G : Finset (Finset V)) (X : Finset
    (hX : Hits G X) : ∃ Y, Y ⊆ X ∧ MinHit G Y
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 136: Lean theorem 15 of 48, hits primal

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Hitting the primal family means meeting both each given set and its complement.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "How hard is it to decide whether a strength-2 table can take one more column?"; register result it supports: Strength-2 extension reduces to N-1 duality questions; Strength-2 extension and monotone non-duality are equivalent (supporting).

```
-- name: ArraysLean.hits_primal
-- ArraysLean/Hypergraph.lean, line 43
theorem hits_primal {V C : Type*} [Fintype V] [Fintype C]
    (F : C → Finset V) (X : Finset V) : Hits (primal F) X ↔
    ∀ j, (∃ v, v ∈ X ∧ v ∈ F j) ∧ (∃ v, v ∈ X ∧ v ∉ F j)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 137: Lean theorem 16 of 48, forward nonddual

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Under the crossing and nonempty-set hypotheses, a valid support exists exactly when the candidate blocker list is incomplete.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "How hard is it to decide whether a strength-2 table can take one more column?"; register result it supports: Strength-2 extension reduces to N-1 duality questions; Strength-2 extension and monotone non-duality are equivalent.

```
-- name: ArraysLean.forward_nonddual
-- ArraysLean/Hypergraph.lean, line 57
theorem forward_nonddual {V C : Type*} [Fintype V] [Fintype C]
    (F : C → Finset V)
    (hcross : ∀ i j, i ≠ j →
      (∃ v, v ∈ F i ∧ v ∈ F j) ∧ (∃ v, v ∈ F i ∧ v ∉ F j))
    (hne : ∀ j, (F j).Nonempty) :
    (∃ X, ValidSupport F X) ↔ transversals (primal F) ≠ badList F
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 138: Lean theorem 17 of 48, forward size bounds

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The primal family and candidate bad list have the stated size bounds.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "How hard is it to decide whether a strength-2 table can take one more column?"; register result it supports: Strength-2 extension reduces to N-1 duality questions.

```
-- name: ArraysLean.forward_size_bounds
-- ArraysLean/Hypergraph.lean, line 124
theorem forward_size_bounds {V C : Type*} [Fintype V] [Fintype C]
    (F : C → Finset V) :
    (primal F).card ≤ 2 * Fintype.card C ∧
    (badList F).card ≤ Fintype.card C * Fintype.card V
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 139: Lean theorem 18 of 48, anchored extension iff

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

A strength-two anchored array extends exactly when its family has a valid support.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run

is claimed. The table of plain statements gives: question "At strength 2, what does a table look like once one row is made the reference row and columns are flipped?"; register result it supports: Normal form of a strength-2 prefix; Strength-2 extension reduces to N-1 duality questions.

```
-- name: ArraysLean.anchored_extension_iff
-- ArraysLean/Anchored.lean, line 9
theorem anchored_extension_iff {V C : Type*} [Fintype V] [DecidableEq V]
    (F : C → Finset V) (hM : Strength (anchored F) 2) :
    Extends (anchored F) 2 ↔ ∃ X, ValidSupport F X
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 140: Lean theorem 19 of 48, anchored strength iff

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The anchored array has strength two exactly when every distinct pair of sets meets in both required ways.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "At strength 2, what does a table look like once one row is made the reference row and columns are flipped?"; register result it supports: Normal form of a strength-2 prefix.

```
-- name: ArraysLean.anchored_strength_iff
-- ArraysLean/Anchored.lean, line 63
theorem anchored_strength_iff {V C : Type*} [DecidableEq V] (F : C → Finset
    Strength (anchored F) 2 ↔ ∀ i j, i ≠ j →
      (∃ v, v ∈ F i ∧ v ∈ F j) ∧ (∃ v, v ∈ F i ∧ v ∉ F j)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 141: Lean theorem 20 of 48, anchored forward reduction

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

For the stated anchored strength-two input, extension is equivalent to inequality with the blocker list.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "How hard is it to decide whether a strength-2 table can take one more column?"; register result it supports: Strength-2 extension reduces to N-1 duality questions.

```
-- name: ArraysLean.anchored_forward_reduction
-- ArraysLean/Anchored.lean, line 94
theorem anchored_forward_reduction {V C : Type*} [Fintype V] [Fintype C]
    [DecidableEq V] (F : C → Finset V) (hk : 2 ≤ Fintype.card C)
    (hM : Strength (anchored F) 2) :
    Extends (anchored F) 2 ↔ transversals (primal F) ≠ badList F
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 142: Lean theorem 21 of 48, hits transversals iff

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

A set hits every minimal transversal exactly when it contains an original edge.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?"; register result it supports: Strength-2 extension and monotone non-duality are equivalent; Maximality of an antichain is completeness of a list of minimal hitting sets.

```
-- name: ArraysLean.hits_transversals_iff
-- ArraysLean/Reverse.lean, line 7
theorem hits_transversals_iff {V : Type*} [Fintype V] [DecidableEq V]
    (G : Finset (Finset V)) (X : Finset V) :
    Hits (transversals G) X ↔ ∃ A ∈ G, A ⊆ X
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 143: Lean theorem 22 of 48, reverse support iff

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Under the stated minimal-transversal hypotheses, the symmetric family has a valid support exactly when the blocker list is incomplete.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Is strength-2

extension the same question as a famous hitting-set puzzle (monotone duality)?"; register result it supports: Strength-2 extension and monotone non-duality are equivalent; Structured duality: the reverse reduction is one instance.

```
-- name: ArraysLean.reverse_support_iff
-- ArraysLean/Reverse.lean, line 36
theorem reverse_support_iff {V : Type*} [Fintype V] [DecidableEq V]
    (G H : Finset (Finset V)) (hGn : G.Nonempty)
    (hH : ∀ B ∈ H, MinHit G B) :
    (∃ X, ValidSupport (symFamily G H) X) ↔ transversals G ≠ H
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 144: Lean theorem 23 of 48, sym strength

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Two nonempty families that minimally hit each other give a strength-two symmetric anchored array.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?"; register result it supports: Strength-2 extension and monotone non-duality are equivalent; Structured duality: the reverse reduction is one instance.

```
-- name: ArraysLean.sym_strength
-- ArraysLean/Reverse.lean, line 146
theorem sym_strength {V : Type*} [Fintype V] [DecidableEq V]
    (G H : Finset (Finset V)) (hGn : G.Nonempty) (hHn : H.Nonempty)
    (hG : ∀ A ∈ G, MinHit H A) (hH : ∀ B ∈ H, MinHit G B) :
    Strength (anchored (symFamily G H)) 2
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 145: Lean theorem 24 of 48, reverse reduction

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The symmetric anchored construction has strength two and translates blocker inequality into extension.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?"; register result it supports: Strength-2 extension and monotone non-duality are equivalent; Structured duality: the reverse reduction is one instance.

```
-- name: ArraysLean.reverse_reduction
-- ArraysLean/Reverse.lean, line 216
theorem reverse_reduction {V : Type*} [Fintype V] [DecidableEq V]
    (G H : Finset (Finset V)) (hGn : G.Nonempty) (hHn : H.Nonempty)
    (hG : ∀ A ∈ G, MinHit H A) (hH : ∀ B ∈ H, MinHit G B) :
    Strength (anchored (symFamily G H)) 2 ∧
    (Extends (anchored (symFamily G H)) 2 ↔ transversals G ≠ H)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 146: Lean theorem 25 of 48, reverse counts

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The reverse construction has the stated exact row and column counts.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?"; register result it supports: Strength-2 extension and monotone non-duality are equivalent.

```
-- name: ArraysLean.reverse_counts
-- ArraysLean/Reverse.lean, line 224
theorem reverse_counts {V : Type*} [Fintype V] [DecidableEq V]
    (G H : Finset (Finset V)) :
    Fintype.card (Option (V ⊕ Bool)) = Fintype.card V + 3 ∧
    Fintype.card (Option (Edge G ⊕ Edge H)) = G.card + H.card + 1
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 147: Lean theorem 26 of 48, strength flip iff

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Flipping any chosen columns preserves strength.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "At strength 2, what does a table look like once one row is made the reference row and columns are flipped?"; register result it supports: Normal form of a strength-2 prefix (supporting); Projection rule (complementing a column).

```
-- name: ArraysLean.strength_flip_iff
-- ArraysLean/Normalization.lean, line 9
theorem strength_flip_iff {R C : Type*} (M : R → C → Bool) (mask : C → Bool
    (t : ℕ) : Strength (flip M mask) t ↔ Strength M t
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 148: Lean theorem 27 of 48, extension flip iff

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Flipping any chosen columns preserves extendibility.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "At strength 2, what does a table look like once one row is made the reference row and columns are flipped?"; register result it supports: Normal form of a strength-2 prefix (supporting).

```
-- name: ArraysLean.extension_flip_iff
-- ArraysLean/Normalization.lean, line 24
theorem extension_flip_iff {R C : Type*} (M : R → C → Bool) (mask : C → Boo
    (t : ℕ) : Extends (flip M mask) t ↔ Extends M t
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 149: Lean theorem 28 of 48, strength row equiv

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Reindexing rows by an equivalence preserves strength.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "At strength 2, what does a table look like once one row is made the reference row and columns are flipped?"; register result it supports: Normal form of a strength-2 prefix (supporting).

```
-- name: ArraysLean.strength_row_equiv
-- ArraysLean/Normalization.lean, line 35
theorem strength_row_equiv {R R' C : Type*} (M : R → C → Bool) (e : R' ≃ R
    (t : ℕ) : Strength (fun r => M (e r)) t ↔ Strength M t
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 150: Lean theorem 29 of 48, extension row equiv

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Reindexing rows by an equivalence preserves extendibility.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "At strength 2, what does a table look like once one row is made the reference row and columns are flipped?"; register result it supports: Normal form of a strength-2 prefix (supporting).

```
-- name: ArraysLean.extension_row_equiv
-- ArraysLean/Normalization.lean, line 45
theorem extension_row_equiv {R R' C : Type*} (M : R → C → Bool) (e : R' ≃
    (t : ℕ) : Extends (fun r => M (e r)) t ↔ Extends M t
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 151: Lean theorem 30 of 48, normalized array eq

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The anchored normalized family is the original array after the specified flips and row reindexing.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run

is claimed. The table of plain statements gives: question "At strength 2, what does a table look like once one row is made the reference row and columns are flipped?"; register result it supports: Normal form of a strength-2 prefix.

```
-- name: ArraysLean.normalized_array_eq
-- ArraysLean/Normalization.lean, line 81
theorem normalized_array_eq {R C : Type*} [Fintype R] [DecidableEq R]
    (M : R → C → Bool) (r0 : R) :
    anchored (normalizedFamily M r0) =
      fun r => flip M (M r0) (anchorEquiv r0 r)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 152: Lean theorem 31 of 48, full forward reduction

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Every stated strength-two input reduces to blocker inequality after normalization at a selected row.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "How hard is it to decide whether a strength-2 table can take one more column?"; register result it supports: Strength-2 extension reduces to N-1 duality questions.

```
-- name: ArraysLean.full_forward_reduction
-- ArraysLean/Normalization.lean, line 93
theorem full_forward_reduction {R C : Type*} [Fintype R] [Fintype C]
    [DecidableEq R] (M : R → C → Bool) (r0 : R)
    (hk : 2 ≤ Fintype.card C) (hM : Strength M 2) :
    Extends M 2 ↔
      transversals (primal (normalizedFamily M r0)) ≠ badList (normalizedFa
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 153: Lean theorem 32 of 48, normalized reduction sizes

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The normalized reduction removes one row and satisfies the displayed family-size bounds.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "How hard is it to decide whether a strength-2 table can take one more column?"; register result it supports: Strength-2 extension reduces to N-1 duality questions.

```
-- name: ArraysLean.normalized_reduction_sizes
-- ArraysLean/Normalization.lean, line 107
theorem normalized_reduction_sizes {R C : Type*} [Fintype R] [Fintype C]
    [DecidableEq R] (M : R → C → Bool) (r0 : R) :
    Fintype.card {r : R // r ≠ r0} = Fintype.card R - 1 ∧
    (primal (normalizedFamily M r0)).card ≤ 2 * Fintype.card C ∧
    (badList (normalizedFamily M r0)).card ≤ Fintype.card C * (Fintype.card
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 154: Lean theorem 33 of 48, minimize correct

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Removing nonminimal edges produces a clutter of no greater size with the same hitting sets.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?"; register result it supports: Strength-2 extension and monotone non-duality are equivalent (supporting).

```
-- name: ArraysLean.minimize_correct
-- ArraysLean/Clutters.lean, line 13
theorem minimize_correct {V : Type*} (G : Finset (Finset V)) :
    Clutter (minimize G) ∧ (minimize G).card ≤ G.card ∧
    ∀ X, Hits (minimize G) X ↔ Hits G X
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 155: Lean theorem 34 of 48, transversals minimize

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Removing nonminimal edges leaves the minimal transversals unchanged.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?"; register result it supports: Strength-2 extension and monotone non-duality are equivalent (supporting).

```
-- name: ArraysLean.transversals_minimize
-- ArraysLean/Clutters.lean, line 33
theorem transversals_minimize {V : Type*} [Fintype V] (G : Finset (Finset V
    transversals (minimize G) = transversals G
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 156: Lean theorem 35 of 48, transversals clutter

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The family of minimal transversals is a clutter.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?"; register result it supports: Strength-2 extension and monotone non-duality are equivalent (supporting).

```
-- name: ArraysLean.transversals_clutter
-- ArraysLean/Clutters.lean, line 40
theorem transversals_clutter {V : Type*} [Fintype V] (G : Finset (Finset V)
    Clutter (transversals G)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 157: Lean theorem 36 of 48, transversals involution

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Taking minimal transversals twice recovers a clutter.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?"; register result it supports: Strength-2 extension and monotone non-duality are equivalent (supporting).

```
-- name: ArraysLean.transversals_involution
-- ArraysLean/Clutters.lean, line 48
theorem transversals_involution {V : Type*} [Fintype V] [DecidableEq V]
    (G : Finset (Finset V)) (hG : Clutter G) : transversals (transversals G
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 158: Lean theorem 37 of 48, forward clutter reduction

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The normalized forward reduction produces two clutters, preserves the decision equivalence, and satisfies the stated size bounds.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "How hard is it to decide whether a strength-2 table can take one more column?"; register result it supports: Strength-2 extension reduces to N-1 duality questions; Strength-2 extension and monotone non-duality are equivalent.

```
-- name: ArraysLean.forward_clutter_reduction
-- ArraysLean/Clutters.lean, line 68
theorem forward_clutter_reduction {R C : Type*} [Fintype R] [Fintype C]
    [DecidableEq R] (M : R → C → Bool) (r0 : R)
    (hk : 2 ≤ Fintype.card C) (hM : Strength M 2) :
    Clutter (minimize (primal (normalizedFamily M r0))) ∧
    Clutter (badList (normalizedFamily M r0)) ∧
    (Extends M 2 ↔ transversals (minimize (primal (normalizedFamily M r0)))
      badList (normalizedFamily M r0)) ∧
    (minimize (primal (normalizedFamily M r0))).card ≤ 2 * Fintype.card C ∧
    (badList (normalizedFamily M r0)).card ≤ Fintype.card C * (Fintype.card
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 159: Lean theorem 38 of 48, forward nonempty edges

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Both families in the stated normalized forward reduction contain only nonempty edges.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "How hard is it to decide whether a strength-2 table can take one more column?"; register result it supports: Strength-2 extension reduces to N-1 duality questions (supporting).

```
-- name: ArraysLean.forward_nonempty_edges
-- ArraysLean/Clutters.lean, line 88
theorem forward_nonempty_edges {R C : Type*} [Fintype R] [Fintype C]
    [DecidableEq R] (M : R → C → Bool) (r0 : R)
    (hk : 2 ≤ Fintype.card C) (hM : Strength M 2) :
    (∀ T ∈ minimize (primal (normalizedFamily M r0)), T.Nonempty) ∧
    (∀ X ∈ badList (normalizedFamily M r0), X.Nonempty)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 160: Lean theorem 39 of 48, dual implies local

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Equality with the blocker family implies the stated local pair condition for the nonempty-edge clutter input.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Is strength-2

extension the same question as a famous hitting-set puzzle (monotone duality)?"; register result it supports: Strength-2 extension and monotone non-duality are equivalent; Local minimality is checkable; completeness is not (12 versus 14).

```
-- name: ArraysLean.dual_implies_local
-- ArraysLean/TotalReverse.lean, line 8
theorem dual_implies_local {V : Type*} [Fintype V] [DecidableEq V]
    (G H : Finset (Finset V)) (hGC : Clutter G)
    (hGE : ∀ A ∈ G, A.Nonempty) (hHE : ∀ B ∈ H, B.Nonempty)
    (heq : transversals G = H) : LocalPair G H
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 161: Lean theorem 40 of 48, fixed yes correct

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The fixed positive table has strength two and admits an extension.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?"; register result it supports: Strength-2 extension and monotone non-duality are equivalent.

```
-- name: ArraysLean.fixed_yes_correct
-- ArraysLean/TotalReverse.lean, line 40
theorem fixed_yes_correct : Strength fixedYes 2 ∧ Extends fixedYes 2
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 162: Lean theorem 41 of 48, total reverse correct

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

For the stated clutter input, the total reverse construction correctly represents blocker inequality.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?"; register result it supports: Strength-2 extension and monotone non-duality are equivalent; Structured duality: the reverse reduction is one instance.

```
-- name: ArraysLean.total_reverse_correct
-- ArraysLean/TotalReverse.lean, line 86
theorem total_reverse_correct {V : Type} [Fintype V] [DecidableEq V]
    (G H : Finset (Finset V)) (hGC : Clutter G)
    (hGE : ∀ A ∈ G, A.Nonempty) (hHE : ∀ B ∈ H, B.Nonempty) :
    ReductionCorrect (reverseTable G H) (transversals G ≠ H)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 163: Lean theorem 42 of 48, total reverse sizes

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

The total reverse table obeys the displayed row, column, and product size bounds.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?"; register result it supports: Strength-2 extension and monotone non-duality are equivalent.

```
-- name: ArraysLean.total_reverse_sizes
-- ArraysLean/TotalReverse.lean, line 99
theorem total_reverse_sizes {V : Type} [Fintype V] [DecidableEq V]
    (G H : Finset (Finset V)) :
    Fintype.card (reverseTable G H).Row ≤ Fintype.card V + 4 ∧
    Fintype.card (reverseTable G H).Col ≤ G.card + H.card + 2 ∧
    Fintype.card (reverseTable G H).Row * Fintype.card (reverseTable G H).C
      (Fintype.card V + 4) * (G.card + H.card + 2)
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 164: Lean theorem 43 of 48, reverse promise sizes

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Every total reverse table has at least four rows and two columns.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "Is strength-2 extension the same question as a famous hitting-set puzzle (monotone duality)?"; register result it supports: Strength-2 extension and monotone non-duality are equivalent (supporting).

```
-- name: ArraysLean.reverse_promise_sizes
-- ArraysLean/TotalReverse.lean, line 116
theorem reverse_promise_sizes {V : Type} [Fintype V] [DecidableEq V]
    (G H : Finset (Finset V)) :
    4 ≤ Fintype.card (reverseTable G H).Row ∧
    2 ≤ Fintype.card (reverseTable G H).Col
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 165: Lean theorem 44 of 48, repeated row lemma

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

For equal distinct rows, any extension can be chosen to give the two rows opposite colors in the stated order.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "If an outside column already splits a class of rows, how many classes can fail to be split?"; register result it supports: Transfer lemma: outside columns colour every class; Distinct-row version stays NP-complete (supporting).

```
-- name: ArraysLean.repeated_row_lemma
-- ArraysLean/FiniteConsequences.lean, line 7
theorem repeated_row_lemma {R C : Type*} (M : R → C → Bool) (s : ℕ)
    (hM : Strength M (s + 1)) (r0 r1 : R) (hne : r0 ≠ r1)
    (heq : ∀ j, M r0 j = M r1 j) :
    Extends M (s + 1) ↔ ∃ c, Strength (append M c) (s + 1) ∧
      c r0 = false ∧ c r1 = true
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 166: Lean theorem 45 of 48, outside column transfer

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

Every required lower-order row class realizes either value on a column outside its defining set.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "If an outside column already splits a class of rows, how many classes can fail to be split?"; register result it supports: Transfer lemma: outside columns colour every class.

```
-- name: ArraysLean.outside_column_transfer
-- ArraysLean/FiniteConsequences.lean, line 33
theorem outside_column_transfer {R C : Type*} (M : R → C → Bool) (s : ℕ)
    (hM : Strength M (s + 1)) (S : Finset C) (hS : S.card = s)
    (x : C → Bool) (j : C) (hj : j ∉ S) (b : Bool) :
    ∃ r, (∀ z ∈ S, M r z = x z) ∧ M r j = b
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 167: Lean theorem 46 of 48, uncolorable classes cover

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

If the selected row classes cannot all be split, their defining column sets cover every column.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "If an outside column already splits a class of rows, how many classes can fail to be split?"; register result it supports: Transfer lemma: outside columns colour every class.

```
-- name: ArraysLean.uncolorable_classes_cover
-- ArraysLean/FiniteConsequences.lean, line 46
theorem uncolorable_classes_cover {R C I : Type*} (M : R → C → Bool) (s : ℕ
    (hM : Strength M (s + 1)) (S : I → Finset C) (hS : ∀ i, (S i).card = s)
    (x : I → C → Bool)
    (hno : ¬ ∃ c : R → Bool, ∀ i b, ∃ r,
      (∀ z ∈ S i, M r z = x i z) ∧ c r = b) :
    ∀ j, ∃ i, j ∈ S i
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 168: Lean theorem 47 of 48, uncolorable class count

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

An unsplittable finite family of s-column classes forces at most s times its family size columns.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "If an outside column already splits a class of rows, how many classes can fail to be split?"; register result it supports: Transfer lemma: outside columns colour every class.

```
-- name: ArraysLean.uncolorable_class_count
-- ArraysLean/FiniteConsequences.lean, line 60
theorem uncolorable_class_count {R C I : Type*} [Fintype C] [Fintype I]
    (M : R → C → Bool) (s : ℕ) (hM : Strength M (s + 1))
    (S : I → Finset C) (hS : ∀ i, (S i).card = s) (x : I → C → Bool)
    (hno : ¬ ∃ c : R → Bool, ∀ i b, ∃ r,
      (∀ z ∈ S i, M r z = x i z) ∧ c r = b) :
    Fintype.card C ≤ s * Fintype.card I
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Theorem 169: Lean theorem 48 of 48, minimal transversal local test

Credit: Tools: Lean 4 and Mathlib; leanchecker; the nanoda kernel checker. This result (Sonny): proved (Lean).

A hitting set is minimal exactly when each of its vertices has an edge meeting the set only there.

Status: PROVED-Lean. Compiled with Lean 4.34.1 against the pinned Mathlib with only the standard axioms (propext, Classical.choice, Quot.sound); the exported proofs of the whole 48-theorem package were re-checked by nanoda (6,174 declarations); leanchecker passed; no lean4lean run is claimed. The table of plain statements gives: question "If every set in a list passes the local test, is the whole list complete?"; register result it supports: Local minimality is checkable; completeness is not (12 versus 14).

```
-- name: ArraysLean.minimal_transversal_local_test
-- ArraysLean/LocalTests.lean, line 5
theorem minimal_transversal_local_test {V : Type*} [DecidableEq V]
    (G : Finset (Finset V)) (X : Finset V) :
    MinHit G X ↔ Hits G X ∧
      ∀ x ∈ X, ∃ T ∈ G, X ∩ T = {x}
```

Proof: Lean theorem named in the code block above (file and line given there); lean/THEOREMS.md; statement/LEAN-THEOREMS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-lean-theorems).

Open questions

CA(15;2,21,3) and CA(23;2,10,4) stay open

Credit: Sonny.

Whether a 15-row, 21-column, 3-symbol strength-2 array exists (the smallest array known has 16 rows) and whether a 23-row, 10-column, 4-symbol one exists (the smallest array known has 24 rows) remains open. 125 symmetry types were shown impossible; 39 timeouts say nothing.

Status: OPEN. All unsatisfiable answers rest on the solver's word.

Proof: none; see chapters/14-more-arrays.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-14-more-arrays) and statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Does a 346th column exist for the 170 x 345 array?

Credit: Sonny.

Whether the 170-row, 345-column array supplied here can take a 346th column without adding rows (a 170×346 array) is open; no search for one has been run on it.

Status: OPEN. Chapter 11 states the rule that such a column must satisfy.

Proof: none; see chapters/16-the-ledger.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-16-the-ledger) and statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

Exact Ladder minima at 12 and 16 rows

Credit: Sonny.

The smallest $|F| + |G|$ for the 12-row and 16-row Ladder tables is known only to lie between 16 and 70, and between 22 and 654 (between 17 and 23 at the lower ends if the four-isolated-models theorem is accepted).

Status: OPEN. See the Ladder table and the four-isolated-models theorem.

Proof: none; see chapters/16-the-ledger.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-16-the-ledger) and statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

The smallest possible number of rows

Credit: Sonny.

For every width in the collection the smallest possible number of rows is unknown: the lower bounds we computed are 47 rows at widths 201 to 203 and 50 rows at widths 325 to 342 (21 of the 29 widths; none was computed at

200, 204 to 206, 223 or 343 to 345), and the arrays supplied have 143 to 170 rows.

Status: OPEN. See the lower bounds on rows.

Proof: none; see chapters/16-the-ledger.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-chapters-16-the-ledger) and statement/RESULTS.md (https://gocheckyourself.org/read/covering-arrays/experts/#ch-statement-results).

21 Check it yourself: covering arrays, the one-column extension theory, and the Lean files

Check it yourself: covering arrays, the one-column extension theory, and the Lean files

The question. Can a stranger confirm, without trusting us, that each array really is a covering array, that the counts in the book are right, and that the Lean files say what we say they do?

The short answer. Yes, for the arrays and the small cases: every check below is a written recipe that gives the file format, the exact condition, the expected answer and a standard public tool or a few lines of code you can write yourself. Nothing here depends on how the arrays were made.

What is claimed

Each of the 29 arrays in the certificates folder is a binary covering array of strength four: for every choice of four columns, all 16 patterns of 0s and 1s appear in at least one row. The widths are 200 to 206, 223 and 325 to 345 columns, one array at each; the sizes, and how each array was made, are on the statement page.

At each of these 29 widths, the array supplied has fewer rows than the upper bound for that width in the pinned comparison table (Charles J. Colbourn's covering array tables, November 2024 version, read through Christopher

Smolen's ca-tools CSV file at commit 5ad91a4925e55e4a40a0332ae6d1aa66777e2429, SHA-256 30e5f10e226f881a413dd0a61f98bd7dac6d4dfb4916cf36505bdc31844833c0, under the lookup rule on the statement page), and fewer rows than the smallest size in Smolen's living table at commit 0fc74c021747b6a160d421dca76bba5b6f3e1bf7, under the same rule.

The table in the locating-array folder is a binary (1-bar,2)-locating array with 13 rows and 13 columns (recipe 11). The tables we found in the literature we checked have 14 rows; that literature check is incomplete, because one paper (Shi, Fu, Wang and Yan, 2021) was not read. A search for a 12-row table with 13 columns found none: one solver answered UNSAT on 601,831 distinct cubes and SAT on none; 552,691 of them are the terminal cubes that close the search tree, and the other 49,140 lie below a cube already answered UNSAT. That 13 rows is the minimum for 13 columns is CONJECTURED: one solver found no 12-row array in a SAT encoding of the problem; a second solver has re-checked a sample of the search, not the whole search, and the search is not part of this book. The 13 by 13 table itself is COMPUTED and can be checked from the file.

Besides these 29 arrays, this book contains further results, each with its own status in the register of results (117 rows, also as a data file):

- 48 Lean theorems about extending covering arrays (the Lean theorem table gives each in plain words);
- Lean files for the hub-row hardness construction (the hardness folder) and for the strength-four rules, growth and parity results (the rules folder);
- 22 strength-2 arrays (the widest array of each of the 20 plateaus, and the alphabet-7 pair), and the small exact cases;
- 5 small fixture tables for the extension recipe.

Everything else in the register is a written proof or a finite computation whose files are not all shipped; the register says which.

Status

Checked by two separately written checkers of ours, one in Python and one in C#, which are not part of this book, each testing every choice of four columns. Both found all 29 arrays to be covering arrays. Preliminary: our pull request 46, which adds these 29 arrays to Christopher Smolen's ca-tools table, was merged by him (see the credits); the rest of the findings has not yet been read by anyone outside this project.

Status of the further material. Lean: the 48-theorem package was built from a fresh directory, passed a per-theorem axiom check (only propositional extensionality, classical choice and quotient soundness), and its exported proofs (6,174 declarations) were re-checked by a second kernel checker, nanoda; Lean's own replay checker (leanchecker) also passed. The files in the hardness and rules folders were compiled with Lean 4.34.1 against the pinned Mathlib; they compiled without errors, without admitted steps, and every listed theorem depends only on the standard axioms (some on none); Lean's own replay checker (leanchecker) also passed on both folders, and the second kernel checker nanoda 0.4.26 accepted exports of their listed declarations (hardness folder: 12 declarations, 5,195 declarations checked; rules folder: 29 declarations, 7,503 declarations checked). They have not had a fresh-directory build with a cache, and no one outside this project has read them. The strength-2 plateau tips and small cases pass the definition-based check of recipe 3 below, which has not been cross-checked against a second implementation for strength 2 and 3 in this book (the alphabet-7 pair was also checked by a second program). The Lean theorems establish formal statements; they do not by themselves show that a formal statement means what its plain version says.

Conditional or incomplete. The four-isolated-models theorem is CONDITIONAL until its proofs are regenerated; the claim that strength-2 extension is not NP-hard is conditional on the Exponential Time Hypothesis; the older strength-2 collection has no full second check (chapter 15); the register says how each result was checked.

Commands

Each recipe states the file format, the condition to test, the answer to expect and a tool. You can follow every step with public tools or code you write yourself. Except where this guide says a recipe's files are not online (available on request): Recipe 7: the comparison table bounds; Recipe 10: the hardness and rules Lean folders. Each recipe also comes as a step-by-step page with commands for Windows, macOS and Linux and the exact output to expect. Run the commands from the top folder of this book. Python 3.10 or later is enough for any program you write yourself; none needs more than the standard library.

Where the files are, relative to the top folder of this book:

```
certificates/<k>-columns-<n>-rows/array.txt      the 29 strength-four array
certificates/strength-2/                          further arrays of strengt
certificates/small-exact/                         three small exact cases
certificates/fixtures/                            five small tables
certificates/SHA256SUMS.txt                       fingerprints of everythin
table/                                            the pinned comparison tab
hub-row/                                          written proof and data of
lean/                                             the Lean package (the dow
lean-hardness/  lean-rules/                       two further Lean folders
```

Recipe 1: the files are the ones we published

Tool: any SHA-256 program.

```
sha256sum -c certificates/SHA256SUMS.txt (macOS: shasum -a 256 -c certifica
```

On Windows, the CertUtil program with the -hashfile option and SHA256 prints the same hash for one file; compare it with the list. Expected: every one of the 113 lines is OK. The same list holds the hash of every small data file next to an array, of the five fixtures and of the formula and proof file of recipe 6. The 29 hashes of the strength-four arrays are repeated in the table under Expected output. The Lean package has its own list (recipe 9), and the

hardness and rules Lean files are listed with their hashes in the Lean files table of the statement folder.

Recipe 2: the 29 strength-four arrays

File format. Each array file is plain ASCII. One row per line. Each line ends with a single line feed (byte 0x0A), including the last line. No carriage returns, no header, no spaces, no blank lines. Every character of a row is 0 or 1. Every row has the same length k. Reject the file if any of these fails.

Size. The number of lines is N and the line length is k, as given in the folder name (for example 201-columns-145-rows) and in the small data file next to the array (fields rows, columns, strength 4, alphabet [0, 1]). The sha256 field of that data file must equal the hash from recipe 1.

Condition. For every set of four column positions $a < b < c < d$ (there are C(k, 4) such sets) and for every pattern (p1, p2, p3, p4) in {0,1}^4 there must be a row r with A[r][a] = p1, A[r][b] = p2, A[r][c] = p3 and A[r][d] = p4. The array passes if this holds for all C(k, 4) column sets and all 16 patterns. It fails otherwise; a useful report names one column set (zero-based positions) and one missing pattern. Nothing may be sampled: every column set must be examined. For scale: $k = 201$ gives 65,998,350 column sets; $k = 345$ gives 580,078,170 column sets, that is 9,281,250,720 pairs of a column set and a pattern. A direct program in a compiled language checks all 29 arrays in about 5 minutes on one core of an ordinary computer; an interpreted language takes much longer. Any correct method is acceptable.

Expected. All 29 arrays pass. Each has 3 to 11 fewer rows than the upper bound for its width in the pinned snapshot (154 rows for widths 200 to 206, 160 for 223, 173 and 174 for the rest) and 1 to 7 fewer than the smallest size in the living table. Still open: the smallest possible array.

What a pass shows and does not show. A pass shows that each published file is a strength-four covering array of the stated size. It does not show that a smaller array is impossible, and it does not check how the arrays were made; each data file states the construction, and chapter 5 describes it.

Recipe 3: other strengths and alphabets

The same recipe with strength t and symbol set {0, ..., v-1}, where t and v are in the small data file next to the array (fields strength and alphabet). The file has one row per line and one digit per entry (the shipped alphabets have at most 8 symbols), with no separators and a final line feed. Condition: for every set of t columns and every one of the v^t patterns there is a row that shows it. Expected: all pass.

Folder	Arrays	Strength	Symbols
strength-2	22 (20 plateau tips and the alphabet-7 pair; chapter 15)	2	3 to 8
small-exact, 22 rows x 9 columns	1	2	4
small-exact, 12 rows x 6 columns	1	3	2

Recipe 4: whether a small table can take one more column

Input: a table in the format above, with N rows, k columns and strength t. Condition (the class lemma of chapter 5, proved in Lean): a 0/1 vector c of length N can be appended as a new column, keeping strength t, exactly when for every set S of t-1 columns and every pattern x on S, the rows whose entries on S equal x contain both a 0 and a 1 of c. The number of valid vectors c can be counted by trying all 2^N vectors when N is at most about 24; a column and its complement count as two. Expected counts, with $t = 3$:

Table	Rows x columns	Valid new columns
the table of all 16 patterns of four switches (fixtures folder)	16 x 4	7,842
the parity table (all patterns of x, y, z and their sum mod 2)	8 x 4	0
the extended Hamming code (fixtures folder)	16 x 8	56
the 4-cube without the rows 0000, 0011, 1101 and 1110	12 x 4	28
the non-extendible array (small-exact folder)	12 x 6	0

Deleting any one row of the 16 x 8 table leaves a table with 0 valid new columns (all 16 cases).

Recipe 5: whether two rows can be replaced by one

Input: any strength-four array of this book, in the format of recipe 2, and two row numbers a and b (counted from 1). Condition: delete rows a and b, and list every four-column pattern that no remaining row shows. The two rows can be replaced by one row exactly when no two listed patterns ask for opposite bits in a column they share; then the row that takes every listed bit (and anything in the other columns) works (the one-row replacement criterion, chapter 10). Expected: the answer depends on the pair. A no concerns those two rows alone; it does not say that a smaller array is impossible. The listing step takes about as long as recipe 2 for one array.

Recipe 6: the exact case with a DRAT proof

Files: a CNF formula in DIMACS format (header line p cnf 8316 25331) and a DRAT proof file, both in the small-exact folder. The question the book says the formula decides: no binary strength-3 covering array with 12 rows and 9 columns has two constant rows (one all 0s, one all 1s). What the file itself shows: the opening 18 clauses are unit clauses setting variables 1 to 9 to false and variables 10 to 18 to true, which fits two constant rows of a 9-column table. The book does not ship a program that builds the formula; a reader who wants to confirm that it says what the chapter claims can write an encoding of the same question and compare the verdict (unsatisfiable). Tool: drat-trim (Heule, Hunt and Wetzler), obtained from its public repository.

```
drat-trim certificates/small-exact/ca12-t3-k9-v2-two-constant-rows/ca12_3_9
```

Expected: the output contains the line s VERIFIED. The formula file has SHA-256 da9940c154e5335b94680645594f0d4c79184dec7b538bc21531c3cbe4b40ad3. (We checked it with a drat-trim build.)

Recipe 7: the comparison table bounds

Our copy of the pinned CSV is not online; it is available on request. The same file is public in Christopher Smolen's ca-tools repository on GitHub at the pinned commit; chapter 3 links it. File: the pinned CSV, with SHA-256 30e5f10e226f881a413dd0a61f98bd7dac6d4dfb4916cf36505bdc31844833c0 and

a header row t, v, k, N, tier, and further columns. Condition: for a width w, take the rows with t = 4, v = 2, snapshot_id equal to colbourn-2024-11-ug (the rows from Colbourn's tables) and k at least w; the upper bound is the smallest N among them. Expected: 154 for widths 200 to 206, 160 for 223, 173 for 325 to 338 and 174 for 339 to 345, each larger than the number of rows of the array at that width. Rows of the same file with other snapshot values were added later and are not part of the snapshot. The living table is read with tier equal to CITED or VERIFIED in place of the snapshot condition, as its check script does: at commit 0fc74c021747b6a160d421dca76bba5b6f3e1bf7 that gives 148, 149, 154, 173 and 174 over the same widths.

Recipe 8: the hub-row construction against brute force

The construction and its proof are in the hub-row folder (the written proof and the written polynomial-time argument). Condition: for a small source (a list of clauses of three variables each), build the table by the construction, test strength 3 directly on all triples of columns, decide extension by complete search over new columns, and decide the source by brute force over all assignments. Expected: the table has strength 3 in every case, and it extends exactly when the source is satisfiable. The data folder lists 11 named sources with the expected size and outcome of each (the Fano plane among them). Over all 1,024 clause families on five variables and these 11 sources (1,035 in all) we found agreement throughout.

Recipe 9: the Lean package

Tool: Lean 4 and Mathlib, with the steps of the building page in the Lean package.

```
cd lean
grep -E '^[0-9a-f]{64}  ' SHA256SUMS.txt | sha256sum -c -  # Linux
grep -E '^[0-9a-f]{64}  ' SHA256SUMS.txt | shasum -a 256 -c -   # macOS
lake exe cache get
lake build
lake env lean Axioms.lean > axioms-actual.txt
grep "depends on axioms" EXPECTED-AXIOMS.txt | diff - axioms-actual.txt
```

The Lean source, configuration and lockfile bytes in the package are byte for byte the files that were verified, except for one comment line in the header of ArraysLean/Core.lean (edited and compiled and checked again; see the verification status page) and the bridge described below (one added file, three copies of files from the rules folder, and two added library entries in the configuration file); its documentation pages (the readme, building page, verification status, theorem types, source list and fingerprint list) were rewritten for this book, with the same facts. Expected: every hash is OK; lake build exits with code 0; the last command prints nothing, which means that each of the 48 theorems depends only on the standard axioms (or none). For one theorem, the Lean command shown below prints its axioms.

```
#print axioms ArraysLean.class_lemma
```

Optional second kernel: export with lean4export and check with nanoda (we used nanoda 0.4.26; 6,174 declarations accepted). Lean's own replay checker, run on the compiled modules (leanchecker, from the Lean toolchain), also passed.

Optional bridge. The Lean theorems of this book use three separately written Lean definitions of strength four (one in the 48-theorem package, two in the rules folder). The bridge file proves that they agree and applies this to the 16 by 5 even-parity table of the register. It sits in the same folder, with three files copied byte for byte from the rules folder that it imports.

```
lake build ArraysBridge
lake env lean ArraysBridge.lean
```

Expected: lake build exits with code 0; the second command ends by printing, for each of the four theorems of the bridge file, a line saying that it depends on the axioms propext, Classical.choice and Quot.sound. It took 15 to 20 seconds on top of the package build against a pre-built Mathlib. The fingerprints of the added files are in the fingerprint list of the package. The

bridge file was built and checked with nanoda 0.4.26 (5,199 declarations accepted) and leanchecker.

Recipe 10: the hardness and rules Lean folders

These two folders are not online; they are available on request. Each folder has a lakefile, a lockfile pinning Mathlib at revision d13f23b723b8a846827a245b89c10fc7d3f11612, and a short file that prints the axioms of every theorem.

```
cd lean-hardness   # then the same in lean-rules
lake exe cache get
lake build
lake env lean <the axiom report file of the folder>
```

Expected: no errors and no admitted steps; every listed theorem depends only on the standard axioms (some on none). We built them by compiling each file in dependency order against a pre-built copy of the same Mathlib revision; we did not run lake build on these two folders from a clean directory. Optional second checks, run: leanchecker on the compiled modules (passed on both folders), and nanoda 0.4.26 on exports of the listed declarations (hardness folder: 12 declarations, 5,195 declarations checked; rules folder: 29 declarations, 7,503 declarations checked). The Lean files table in the statement folder lists the SHA-256 of every file; the shipped files differ from the project files only in comment lines: two in the header of the main hardness file and one in the rules file on the repair clique. After the last of these edits both folders were compiled again, every listed theorem still depends only on the standard axioms (some on none), and leanchecker passed on every module.

Recipe 11: the 13 by 13 locating array

Files: the table in the file array-13x13.txt (folder locating-array; SHA-256 05f03f3cca1799190e2cc17a74221929354d1cb8f3184d89b1e668538374fcbe). Write the check yourself from the condition below; a correct check prints the two lines shown.

File format. One row per line, one digit (0 or 1) per column, no separators, a final line feed: 13 lines of 13 digits. Line endings: the SHA-256 above is for this file with LF line endings (182 bytes). The same 13 rows saved with CRLF line endings, including one after the last row (195 bytes), have SHA-256 f66f0895ea25def9b7d87aa62e327a10834e7b6a9208f4b9db04000d9b90a35b; a copy that shows this second value is the same table.

Condition. For every pair of columns i < j and every pair of values (a, b), take the set of rows that show a in column i and b in column j. There are 4 x 78 = 312 such sets. A binary table with these properties is a (1-bar,2)-locating array when all 312 sets are non-empty and no two are equal.

A correct check prints:

```
13 rows, 13 columns, 312 different non-empty row sets of 312
PASS
```

Flipping any single entry of the table makes the check print FAIL, for each of the 169 positions. Run time: under a second.

What this does and does not show. It shows that this table is a (1-bar,2)-locating array with 13 rows. It does not by itself show that fewer rows are impossible: that statement is CONJECTURED, on the basis described in chapter 13: one solver found no 12-row array in a SAT encoding of the problem; a second solver has re-checked a sample of the search, not the whole search, and the search is not part of this book.

Expected output

The 29 strength-four arrays and their fingerprints (SHA-256 of each array file):

Columns	Rows	SHA-256
200	143	bca06ed8480ff439a1eee7300f7069b420513d2fe840cecd32797a8500f4a1b4
201	145	9c00a0426b47359b05a559bafa6993296eea34e810ee9c0ad06bd81c28f5c527
202	145	9b5a69020011ed97d7192da7974d740e5ba0f017df94e2598307750896c958b6
203	145	f660c98721d28a0d079e03e261b310eb7f8c271c06574d801c0f91b6b236066a
204	145	ed6e53890e1e1aa8a84d283d89ed38cd402e2e6a1ff5fc3a671c527f9683ca2c
205	145	30689f03d7b3faa56cf2f7032de90624981f7a6ad87fef9604bb85bcb8ddd7c2
206	145	861986282f2faf50c907e95b18bf68aa5ffcd823fbf6b2959fb3c387b2228124
223	153	6d754949d9cd2d81ff654314bb626e339834357f6ff20ecf94a8568a85c4edef
325	166	c23e65877654e0bf0aa32afa72aa63112e615404580dd1247c3c696022bf3fe5
326	167	72920f3fb2b57fefb2dda632321a9f9ff3f7781686537a78dc6ecee45512dd74
327	169	2c8234082b703cc787308114b05bc8f1fd9153dfb2325d2522e69edc850dd308
328	169	9982a1ae49f7c55d7fb2464422933b4cce8879c6d52327ace18b90b68406fe29
329	170	bc9806f8849a3dbc322a744aa28a414e8b9d01c3229586d58ecc00172cdf8b8d
330	170	f5b6457fdad37d835bcb7e6d72ba6cae7fe1e30a5005b8b000fa77b7ba71720c
331	170	b4edef92a4af98de5fde2ac82cfba232f5bcc7326b226d3c316e64c148f9d201
332	170	b360ebad2be89a66a30efe3008b2e454c32c7252949866f0af77923fcc09c2f2
333	170	f12e53a5c4fc91ca8dcc6dac0cb830be89c006b3289dff364c1f33fc284bef4b
334	170	e4b7dd9e590ff050cf6558a92b15b624194f57a7ba865886e9f67a666e337530
335	170	a657aa6ed44601ceb189051467d1c972e1f616bebd92d58d995281bccc231663
336	170	cdf52b82c95bb1e6845c7bf316894523109c438b3145f8955fcbce9009b44e46
337	170	8220ba62ed56bbeb95307c33531196c9479b099f68e9b5ec25aa957610771843
338	170	05b0b10650fe3199e07510703b50df29b8cb4fcce0ff804bdc28e7583cdf7a7e
339	170	e56158cecf586a8e5308976153d2f2d0d5722adc1721320462955fb7e0e212b1
340	170	2a18ed8ce9dd0ff77c522eb12cb132a85c1bbc28cd88370fd04783b6b2d46c5b
341	170	9e3426d91bdaf021b8958af75a228eeb8175fa10638a5c9c593f943692ea58a0
342	170	08a9a1124afff17cef6bc4512427b7e4cc99453b7b7ad6f1c5d59ca8ca348b01
343	170	fb8d8fbf10b3eefd3b08e31ee57066cb2fac83429b6cc7d72ca2946839838e04
344	170	26058c911e4b33d1cba58d2e6d9b68f69d1029ef2b45b9cc4c49ebe18fcd09ab
345	170	6b619b44dc06724ee07d5f3a2adbab4985f7c6b7f56b616d9e70a3f72bb7b5d2

All 29 pass recipe 2. The further expectations are stated in each recipe above.

Time and memory

- Recipe 1 and recipe 7: under a second.
- Recipe 2: with a compiled program, about 5 minutes for all 29 arrays on one core (about 20 MB); with an interpreted language, about 1 to 1.5 minutes for a 201-column array and about 4 to 8 minutes for a 325- to 345-column array on one idle core, depending on the program, and several hours for all 29; spread over 16 processes, under an hour. Memory is small, under 100 MB per array.
- Recipe 3 and recipe 4: seconds for the strength-2 arrays and the small tables (the extension counts for the 16-row tables take a fraction of a second).
- Recipe 6: seconds with drat-trim.
- Recipe 9: the package build took 184 seconds from a fresh directory with the Mathlib cache and needs about 8 GB of memory. The two extra folders of recipe 10 compiled in under a minute in total against a pre-built Mathlib.

What is not claimed

- No claim that any run count is the smallest possible; that remains unknown. No claim for width 346 or above.
- The comparison is with one pinned snapshot of Colbourn's tables and with Smolen's living table at one commit, under the lookup rules above. Still open: arrays published outside these two tables at these widths.
- Our copy of the pinned CSV is not online (it is available on request, with its MIT licence text; the same file is public at the pinned commit of Smolen's ca-tools repository, see recipe 7), and the array checks do not read it. The upper bounds on the statement page were copied from the source repository's table and then recomputed from the CSV (recipe 7); all 29 agree.
- The array checks test strength four. They do not re-derive the arrays from their starting tables; each data file states the construction.

- The written proofs behind the PROVED-written rows: the hub-row reduction is shipped in the hub-row folder (with its running time); the others, one file per chapter (chapter 6, chapter 7, chapter 8, chapter 9, chapters 10 and 11, chapter 12 and chapter 13), are not online and are available on request. Each file collects the written notes for its chapter, edited only to remove process notes and references to working files that are not shipped (where such a file backed a computation, the text says "not shipped"); the register row names the result, and the file holds its argument under a part heading. A few short results carry their whole argument in the chapter itself: complementing a column needs eight repair rows (chapter 10), the table lookup rule and the remark that a solver answer is not a lower bound (chapter 15). Statements in the register marked PROVED-written are proofs on paper; they were read by the author and, where the register says so, in a separate reading, and none has been through a proof assistant unless it is marked PROVED-Lean. Statements marked COMPUTED are finite computations; they are exact for the cases computed and say nothing about other cases.

- Lean covers formal statements. It does not cover running times (proved in writing only), the NP-completeness of the source problem used in chapter 9 (taken from Schaefer 1978), the byte-to-pattern interface of the completion rule (written and tested on finite cases), or whether a formal definition matches its plain statement.
- The strength-2 collection is shipped as a summary (chapter 15): 22 arrays in full and a listing of the rest. The older collection of 720 arrays has no full second check.
- Some working files are not shipped (solver outputs, search logs, the proof files behind the exact 14-row minimum and the four-isolated-models theorem, the arrays of the Ladder). How the arrays were made is described in chapter 5 and in each data file; the search programs are not shipped.

Supporting records are available on request.

How I direct this work, and my own tools, are not part of this book. What ships: the 29 strength-four arrays and the further arrays, the 13 by 13 locating array, the Lean package of the 48 theorems, the written proof and data of the hub-row construction, a SHA-256 fingerprint for each file and a written recipe for each check. What is not online, and is available on request: the pinned comparison table, the two further Lean folders (hardness and rules), the written proofs for chapters 6 to 13, the solver outputs and search logs, the proof files behind the exact 14-row minimum and the four-isolated-models theorem, and the arrays of the Ladder.

Where to go next

- Why does the answer for one more column change between strength 2 and strength 3? Chapter 9: strength three is NP-complete.
- What did we try that did not work? Chapter 16: the ledger.

We hope these arrays help others find more results and keep their curiosity going. We'd love to collaborate with anyone.

22 Files and fingerprints

This chapter is the book's file SHA256SUMS.txt.

23 Trust and verification

This chapter is the book's file VERIFY.md.

24 Lean sources: how to check them

This chapter is the book's file THEOREMS.md.

25 People and works

This chapter is the book's file CREDITS.md.

26 Files and rights

This chapter is the book's file LICENSE-CC-BY-4.0.txt.

27 Each result on its own page

1. Computation 1: Comparison with the pinned table and the living table at 29 widths STATUS: COMPUTED
2. Computation 2: The 29 binary strength-four arrays STATUS: COMPUTED
3. Lemma 3: Class lemma: extension is proper two-colouring of the row-class hypergraph (every strength) STATUS: LEAN-CHECKED (SECOND-KERNEL)
4. Lemma 4: Support criterion for strength 4 STATUS: LEAN-CHECKED (SECOND-KERNEL)
5. Proposition 5: Normal form of a strength-2 prefix STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST
6. Theorem 6: Strength-2 extension reduces to N-1 duality questions STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST
7. Proposition 7: No polynomial NP-hardness reduction for strength-2 extension unless ETH fails STATUS: CONDITIONAL
8. Theorem 8: Strength-2 extension and monotone non-duality are equivalent STATUS: LEAN-CHECKED (SECOND-KERNEL)
9. Proposition 9: The reductions between strength-2 extension and monotone non-duality run in polynomial time STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST
10. Lemma 10: One-row antichain extension lemma STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST
11. Proposition 11: Maximality of an antichain is completeness of a list of minimal hitting sets STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

12. Proposition 12: Local minimality is checkable; completeness is not (12 versus 14) STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

13. Proposition 13: Fixed rank spread is tractable STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

14. Proposition 14: Exact count of extensions STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

15. Proposition 15: A rank-counting sufficient certificate STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

16. Proposition 16: Structured duality: the reverse reduction is one instance STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

17. Proposition 17: An easy region: the Lambda test STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

18. Proposition 18: A shortcut that fails: a maximal intersecting antichain need not be self-dual STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

19. Computation 19: Stuck arrays: small unsatisfiable cores of seven 'no more columns' cases STATUS: COMPUTED

20. Proposition 20: The strength-2 equivalence with monotone duality STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

21. Proposition 21: Pair-class formulation STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

22. Proposition 22: Strength-3 extension is in NP; the classes form an antichain STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

23. Lemma 23: Padding lemma for intersecting Sperner hypergraphs STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

24. Theorem 24: Complement of duality reduces to strength-3 extension STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

25. Proposition 25: Decision orientation STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

26. Proposition 26: Exact formulation through bad transversals STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

27. Proposition 27: The bad-transversal list can be superpolynomial STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

28. Proposition 28: Union-bound test STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

29. Lemma 29: Lovasz Local Lemma test (Moser-Tardos) STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

30. Proposition 30: The shared design A_r (the [8,4,4] example) STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

31. Proposition 31: Shared-column equality stars: the part that works STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

32. Computation 32: Worked example tables for chapters 5 to 7 STATUS: COMPUTED

33. Proposition 33: The easy side of strength-3 extension STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

34. Proposition 34: A polynomial subclass: unit-row arrays STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

35. Lemma 35: Transfer lemma: outside columns colour every class STATUS: LEAN-CHECKED (SECOND-KERNEL)

36. Proposition 36: Private-support obstruction STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

37. Proposition 37: A logical bridge gadget (hypergraph level only) STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

38. Proposition 38: Off-edge label certificates and safe padding STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

39. Proposition 39: Washing: padding and duplicate rows free the colours STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

40. Proposition 40: Stars cannot be attached to a previous occurrence array STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

41. Proposition 41: Complementary hubs enforce the wrong not-all-equal relation STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

42. Proposition 42: Uniform shared flags give not-all-equal on all variables STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

43. Theorem 43: Conditional exact-preservation theorem for supplied gadget data STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

44. Proposition 44: Auxiliary selector encoding is not ordinary duality STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

45. Proposition 45: Compression lower bounds for the forbidden DNF STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

46. Computation 46: Changing both duality families does not help STATUS: COMPUTED

47. Computation 47: Interacting auxiliaries: a fitted compiler and the 167x9 escape STATUS: COMPUTED

48. Computation 48: Uniform clause constructions fail STATUS: COMPUTED

49. Computation 49: An orientation recipe STATUS: COMPUTED

50. Proposition 50: The eighteen walls STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

51. Proposition 51: Separate star tests are not a characterisation STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

52. Theorem 52: No fixed VC-dimension bound; two-family gap feasibility is NP-complete STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

53. Theorem 53: Strength-3 one-column extension is NP-complete (the hub-row reduction) STATUS: PROVED IN WRITING

54. Lemma 54: Raising lemma: every fixed strength t >= 3 STATUS: LEAN-CHECKED (SECOND-KERNEL)

55. Lemma 55: NP-completeness at every fixed strength t >= 3 (raising lemma composed with the hub-row reduction) STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

56. Theorem 56: Distinct-row version stays NP-complete STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

57. Proposition 57: Thin arrays and both properties at once STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

58. Lemma 58: Residual-row completion rule STATUS: LEAN-CHECKED (SECOND-KERNEL)

59. Proposition 59: From array bytes to missing partial words STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

60. Lemma 60: Block-of-columns criterion STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

61. Lemma 61: Projection rule STATUS: LEAN-CHECKED (SECOND-KERNEL)

62. Proposition 62: Graph-colouring form of fixed-row replacement STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

63. Proposition 63: Width ceiling for a fixed number of rows STATUS: LEAN-CHECKED (SECOND-KERNEL)

64. Computation 64: A tempting invariant fails STATUS: COMPUTED

65. Proposition 66: Repeatable growth recurrence (clone and repair) STATUS: LEAN-CHECKED (SECOND-KERNEL)
66. Proposition 68: Clone obstruction STATUS: LEAN-CHECKED (SECOND-KERNEL)
67. Proposition 69: Complementing a column needs at least eight repair rows STATUS: PROVED IN WRITING
68. Proposition 70: No universal one-row extension (parity 16x5) STATUS: LEAN-CHECKED (SECOND-KERNEL)
69. Proposition 71: A full linear parity block is too large STATUS: LEAN-CHECKED (SECOND-KERNEL)
70. Lemma 74: One-row replacement criterion STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST
71. Proposition 75: Clique limit STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST
72. Computation 76: Lower bounds on the number of rows STATUS: COMPUTED
73. Lemma 77: Support antichain lemma STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST
74. Computation 80: Strength five STATUS: COMPUTED
75. Proposition 83: Certificate analogue for the 346 search (blocks of rows) STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST
76. Computation 85: Maxima of isolated models f(n,d) STATUS: COMPUTED
77. Computation 86: The Ladder table STATUS: COMPUTED
78. Proposition 87: Critical-clause local lower bound STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST
79. Proposition 88: Constant-coefficient elimination STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST
80. Lemma 89: Isolated distance lemma STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

81. Proposition 90: Four isolated models STATUS: CONDITIONAL

82. Proposition 91: Fixed-gap reduction STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

83. Lemma 92: Critical-cover rule STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

84. Computation 93: Restricted minimum 24 STATUS: COMPUTED

85. Computation 94: The exact minimum is 24 (15 in F, 9 in G) STATUS: COMPUTED

86. Lemma 95: Partial-isolation distance lemma STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

87. Proposition 96: From a cross-intersecting pair to an intersecting family; list selector STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

88. Proposition 97: Distinct-row decoder failure STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

89. Proposition 98: Witness selection STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

90. Proposition 99: No general reduction to duality was established STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

91. Proposition 100: New predicates and non-uniform auxiliaries STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

92. Theorem 101: Ternary strength-2 one-column extension is NP-complete STATUS: PROVED IN WRITING; THE WRITTEN PROOF IS AVAILABLE ON REQUEST

93. Computation 102: No CA(12;3,9,2) with two constant rows STATUS: COMPUTED

94. Computation 103: CA(22;2,9,4) with cyclic symmetry STATUS: COMPUTED

95. Computation 104: A non-extendible CA(12;3,6,2) STATUS: COMPUTED

96. Computation 108: Strength-2 and strength-3 arrays extending published arrays STATUS: COMPUTED
97. Computation 109: Plateaus: arrays one step behind the published table STATUS: COMPUTED
98. Computation 110: Plateaus extended by one more column STATUS: COMPUTED
99. Computation 111: Balanced extension: a negative result STATUS: COMPUTED
100. Proposition 112: A solver's 'no column fits' is never a lower bound STATUS: PROVED IN WRITING
101. Lemma 113: Table lookup rule: dropping columns preserves coverage STATUS: PROVED IN WRITING
102. Computation 114: The later collection of 1,631 strength-2 arrays STATUS: COMPUTED
103. Computation 115: The older collection of 720 arrays STATUS: COMPUTED
104. Computation 116: The two alphabet-7 arrays 163 x 917 and 169 x 1076 STATUS: COMPUTED
105. Proposition 117: Two special branches of the normal form STATUS: PROVED IN WRITING
106. Proposition 118: Affine model set, with Ladder bounds 16 and 22 STATUS: PROVED IN WRITING
107. Proposition 119: Tight cross bound of two isolated assignments STATUS: PROVED IN WRITING
108. Computation 120: Counts for the older collection STATUS: COMPUTED
109. Theorem 122: Lean theorem 1 of 48, class lemma STATUS: LEAN-CHECKED (SECOND-KERNEL)
110. Theorem 123: Lean theorem 2 of 48, extension iff two coloring STATUS: LEAN-CHECKED (SECOND-KERNEL)
111. Theorem 124: Lean theorem 3 of 48, support criterion STATUS: LEAN-CHECKED (SECOND-KERNEL)

112. Theorem 125: Lean theorem 4 of 48, strength mono STATUS: LEAN-CHECKED (SECOND-KERNEL)
113. Theorem 126: Lean theorem 5 of 48, strength on small set STATUS: LEAN-CHECKED (SECOND-KERNEL)
114. Theorem 127: Lean theorem 6 of 48, normalized injective STATUS: LEAN-CHECKED (SECOND-KERNEL)
115. Theorem 128: Lean theorem 7 of 48, width bound STATUS: LEAN-CHECKED (SECOND-KERNEL)
116. Theorem 129: Lean theorem 8 of 48, strength four width bound STATUS: LEAN-CHECKED (SECOND-KERNEL)
117. Theorem 130: Lean theorem 9 of 48, raised strength STATUS: LEAN-CHECKED (SECOND-KERNEL)
118. Theorem 131: Lean theorem 10 of 48, raised extends iff STATUS: LEAN-CHECKED (SECOND-KERNEL)
119. Theorem 132: Lean theorem 11 of 48, raised counts STATUS: LEAN-CHECKED (SECOND-KERNEL)
120. Theorem 133: Lean theorem 12 of 48, padding bound STATUS: LEAN-CHECKED (SECOND-KERNEL)
121. Theorem 134: Lean theorem 13 of 48, raised row bound STATUS: LEAN-CHECKED (SECOND-KERNEL)
122. Theorem 135: Lean theorem 14 of 48, minimal hit subset STATUS: LEAN-CHECKED (SECOND-KERNEL)
123. Theorem 136: Lean theorem 15 of 48, hits primal STATUS: LEAN-CHECKED (SECOND-KERNEL)
124. Theorem 137: Lean theorem 16 of 48, forward nonddual STATUS: LEAN-CHECKED (SECOND-KERNEL)
125. Theorem 138: Lean theorem 17 of 48, forward size bounds STATUS: LEAN-CHECKED (SECOND-KERNEL)
126. Theorem 139: Lean theorem 18 of 48, anchored extension iff STATUS: LEAN-CHECKED (SECOND-KERNEL)

127. Theorem 140: Lean theorem 19 of 48, anchored strength iff STATUS: LEAN-CHECKED (SECOND-KERNEL)
128. Theorem 141: Lean theorem 20 of 48, anchored forward reduction STATUS: LEAN-CHECKED (SECOND-KERNEL)
129. Theorem 142: Lean theorem 21 of 48, hits transversals iff STATUS: LEAN-CHECKED (SECOND-KERNEL)
130. Theorem 143: Lean theorem 22 of 48, reverse support iff STATUS: LEAN-CHECKED (SECOND-KERNEL)
131. Theorem 144: Lean theorem 23 of 48, sym strength STATUS: LEAN-CHECKED (SECOND-KERNEL)
132. Theorem 145: Lean theorem 24 of 48, reverse reduction STATUS: LEAN-CHECKED (SECOND-KERNEL)
133. Theorem 146: Lean theorem 25 of 48, reverse counts STATUS: LEAN-CHECKED (SECOND-KERNEL)
134. Theorem 147: Lean theorem 26 of 48, strength flip iff STATUS: LEAN-CHECKED (SECOND-KERNEL)
135. Theorem 148: Lean theorem 27 of 48, extension flip iff STATUS: LEAN-CHECKED (SECOND-KERNEL)
136. Theorem 149: Lean theorem 28 of 48, strength row equiv STATUS: LEAN-CHECKED (SECOND-KERNEL)
137. Theorem 150: Lean theorem 29 of 48, extension row equiv STATUS: LEAN-CHECKED (SECOND-KERNEL)
138. Theorem 151: Lean theorem 30 of 48, normalized array eq STATUS: LEAN-CHECKED (SECOND-KERNEL)
139. Theorem 152: Lean theorem 31 of 48, full forward reduction STATUS: LEAN-CHECKED (SECOND-KERNEL)
140. Theorem 153: Lean theorem 32 of 48, normalized reduction sizes STATUS: LEAN-CHECKED (SECOND-KERNEL)
141. Theorem 154: Lean theorem 33 of 48, minimize correct STATUS: LEAN-CHECKED (SECOND-KERNEL)

142. Theorem 155: Lean theorem 34 of 48, transversals minimize STATUS: LEAN-CHECKED (SECOND-KERNEL)
143. Theorem 156: Lean theorem 35 of 48, transversals clutter STATUS: LEAN-CHECKED (SECOND-KERNEL)
144. Theorem 157: Lean theorem 36 of 48, transversals involution STATUS: LEAN-CHECKED (SECOND-KERNEL)
145. Theorem 158: Lean theorem 37 of 48, forward clutter reduction STATUS: LEAN-CHECKED (SECOND-KERNEL)
146. Theorem 159: Lean theorem 38 of 48, forward nonempty edges STATUS: LEAN-CHECKED (SECOND-KERNEL)
147. Theorem 160: Lean theorem 39 of 48, dual implies local STATUS: LEAN-CHECKED (SECOND-KERNEL)
148. Theorem 161: Lean theorem 40 of 48, fixed yes correct STATUS: LEAN-CHECKED (SECOND-KERNEL)
149. Theorem 162: Lean theorem 41 of 48, total reverse correct STATUS: LEAN-CHECKED (SECOND-KERNEL)
150. Theorem 163: Lean theorem 42 of 48, total reverse sizes STATUS: LEAN-CHECKED (SECOND-KERNEL)
151. Theorem 164: Lean theorem 43 of 48, reverse promise sizes STATUS: LEAN-CHECKED (SECOND-KERNEL)
152. Theorem 165: Lean theorem 44 of 48, repeated row lemma STATUS: LEAN-CHECKED (SECOND-KERNEL)
153. Theorem 166: Lean theorem 45 of 48, outside column transfer STATUS: LEAN-CHECKED (SECOND-KERNEL)
154. Theorem 167: Lean theorem 46 of 48, uncolorable classes cover STATUS: LEAN-CHECKED (SECOND-KERNEL)
155. Theorem 168: Lean theorem 47 of 48, uncolorable class count STATUS: LEAN-CHECKED (SECOND-KERNEL)
156. Theorem 169: Lean theorem 48 of 48, minimal transversal local test STATUS: LEAN-CHECKED (SECOND-KERNEL)

157. Open question: CA(15;2,21,3) and CA(23;2,10,4) stay open STATUS: OPEN

158. Open question: Does a 346th column exist for the 170 x 345 array? STATUS: OPEN

159. Open question: Exact Ladder minima at 12 and 16 rows STATUS: OPEN

160. Open question: The smallest possible number of rows STATUS: OPEN

© 2026 Sonny. The books are under CC BY 4.0 (Creative Commons Attribution 4.0 International): reuse them with credit and a link. Each book's licence file lists the files that keep their own licences.