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)

Overview

Suppose a machine has hundreds of on/off switches and you want to test it. A binary covering array of strength four is a table of 0s and 1s, one row per test and one column per switch, in which every choice of four switches shows all 16 on/off patterns somewhere. Fewer rows mean a cheaper test plan, and nobody knows the smallest possible number of rows at the widths in this book. The book supplies 29 such tables, with 200 to 206, 223 and 325 to 345 columns and 143 to 170 rows. Two separately written programs tested every choice of four columns, so their status is computed. At each width the table has fewer rows than the upper bound in one fixed snapshot of Colbourn's published tables, by 3 to 11 rows, and than the smallest size in Smolen's living table at one fixed commit, by 1 to 7 rows. 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). The book also asks whether a finished table can take one more column. That question is NP-complete from strength 3 up, with written proofs, and 48 related theorems are proved in Lean. The smallest possible sizes stay open. The check guide gives recipes, and the files carry fingerprints, so you can verify the tables with your own code.

CHECK IT YOURSELF

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

---

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. Try it yourself

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

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

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

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

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

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.

We hope these arrays help others find more results and keep their curiosity going. We'd love to collaborate with anyone.

Try it yourself

The figures of this chapter are the interactive checks on the tools page (they need JavaScript).

Each result on its own page

- Computation 1: Comparison with the pinned table and the living table at 29 widths
- Computation 2: The 29 binary strength-four arrays
- Lemma 3: Class lemma: extension is proper two-colouring of the row-class hypergraph (every strength)
- Lemma 4: Support criterion for strength 4
- Proposition 5: Normal form of a strength-2 prefix
- Theorem 6: Strength-2 extension reduces to N-1 duality questions
- Proposition 7: No polynomial NP-hardness reduction for strength-2 extension unless ETH fails
- Theorem 8: Strength-2 extension and monotone non-duality are equivalent
- Proposition 9: The reductions between strength-2 extension and monotone non-duality run in polynomial time
- Lemma 10: One-row antichain extension lemma
- Proposition 11: Maximality of an antichain is completeness of a list of minimal hitting sets
- Proposition 12: Local minimality is checkable; completeness is not (12 versus 14)

- Proposition 13: Fixed rank spread is tractable
- Proposition 14: Exact count of extensions
- Proposition 15: A rank-counting sufficient certificate
- Proposition 16: Structured duality: the reverse reduction is one instance
- Proposition 17: An easy region: the Lambda test
- Proposition 18: A shortcut that fails: a maximal intersecting antichain need not be self-dual
- Computation 19: Stuck arrays: small unsatisfiable cores of seven 'no more columns' cases
- Proposition 20: The strength-2 equivalence with monotone duality
- Proposition 21: Pair-class formulation
- Proposition 22: Strength-3 extension is in NP; the classes form an antichain
- Lemma 23: Padding lemma for intersecting Sperner hypergraphs
- Theorem 24: Complement of duality reduces to strength-3 extension
- Proposition 25: Decision orientation
- Proposition 26: Exact formulation through bad transversals
- Proposition 27: The bad-transversal list can be superpolynomial
- Proposition 28: Union-bound test
- Lemma 29: Lovasz Local Lemma test (Moser-Tardos)
- Proposition 30: The shared design A_r (the [8,4,4] example)
- Proposition 31: Shared-column equality stars: the part that works
- Computation 32: Worked example tables for chapters 5 to 7
- Proposition 33: The easy side of strength-3 extension
- Proposition 34: A polynomial subclass: unit-row arrays
- Lemma 35: Transfer lemma: outside columns colour every class
- Proposition 36: Private-support obstruction
- Proposition 37: A logical bridge gadget (hypergraph level only)
- Proposition 38: Off-edge label certificates and safe padding
- Proposition 39: Washing: padding and duplicate rows free the colours
- Proposition 40: Stars cannot be attached to a previous occurrence array
- Proposition 41: Complementary hubs enforce the wrong not-all-equal relation
- Proposition 42: Uniform shared flags give not-all-equal on all variables
- Theorem 43: Conditional exact-preservation theorem for supplied gadget data
- Proposition 44: Auxiliary selector encoding is not ordinary duality
- Proposition 45: Compression lower bounds for the forbidden DNF
- Computation 46: Changing both duality families does not help
- Computation 47: Interacting auxiliaries: a fitted compiler and the 167x9 escape
- Computation 48: Uniform clause constructions fail
- Computation 49: An orientation recipe
- Proposition 50: The eighteen walls
- Proposition 51: Separate star tests are not a characterisation

- Theorem 52: No fixed VC-dimension bound; two-family gap feasibility is NP-complete
- Theorem 53: Strength-3 one-column extension is NP-complete (the hub-row reduction)
- Lemma 54: Raising lemma: every fixed strength t >= 3
- Lemma 55: NP-completeness at every fixed strength t >= 3 (raising lemma composed with the hub-row reduction)
- Theorem 56: Distinct-row version stays NP-complete
- Proposition 57: Thin arrays and both properties at once
- Lemma 58: Residual-row completion rule
- Proposition 59: From array bytes to missing partial words
- Lemma 60: Block-of-columns criterion
- Lemma 61: Projection rule
- Proposition 62: Graph-colouring form of fixed-row replacement
- Proposition 63: Width ceiling for a fixed number of rows
- Computation 64: A tempting invariant fails
- Proposition 66: Repeatable growth recurrence (clone and repair)
- Proposition 68: Clone obstruction
- Proposition 69: Complementing a column needs at least eight repair rows
- Proposition 70: No universal one-row extension (parity 16x5)
- Proposition 71: A full linear parity block is too large
- Lemma 74: One-row replacement criterion
- Proposition 75: Clique limit
- Computation 76: Lower bounds on the number of rows
- Lemma 77: Support antichain lemma
- Computation 80: Strength five
- Proposition 83: Certificate analogue for the 346 search (blocks of rows)
- Computation 85: Maxima of isolated models f(n,d)
- Computation 86: The Ladder table
- Proposition 87: Critical-clause local lower bound
- Proposition 88: Constant-coefficient elimination
- Lemma 89: Isolated distance lemma
- Proposition 90: Four isolated models
- Proposition 91: Fixed-gap reduction
- Lemma 92: Critical-cover rule
- Computation 93: Restricted minimum 24
- Computation 94: The exact minimum is 24 (15 in F, 9 in G)
- Lemma 95: Partial-isolation distance lemma
- Proposition 96: From a cross-intersecting pair to an intersecting family; list selector
- Proposition 97: Distinct-row decoder failure
- Proposition 98: Witness selection

- Proposition 99: No general reduction to duality was established
- Proposition 100: New predicates and non-uniform auxiliaries
- Theorem 101: Ternary strength-2 one-column extension is NP-complete
- Computation 102: No CA(12;3,9,2) with two constant rows
- Computation 103: CA(22;2,9,4) with cyclic symmetry
- Computation 104: A non-extendible CA(12;3,6,2)
- Computation 108: Strength-2 and strength-3 arrays extending published arrays
- Computation 109: Plateaus: arrays one step behind the published table
- Computation 110: Plateaus extended by one more column
- Computation 111: Balanced extension: a negative result
- Proposition 112: A solver's 'no column fits' is never a lower bound
- Lemma 113: Table lookup rule: dropping columns preserves coverage
- Computation 114: The later collection of 1,631 strength-2 arrays
- Computation 115: The older collection of 720 arrays
- Computation 116: The two alphabet-7 arrays 163 x 917 and 169 x 1076
- Proposition 117: Two special branches of the normal form
- Proposition 118: Affine model set, with Ladder bounds 16 and 22
- Proposition 119: Tight cross bound of two isolated assignments
- Computation 120: Counts for the older collection
- Theorem 122: Lean theorem 1 of 48, class lemma
- Theorem 123: Lean theorem 2 of 48, extension iff two coloring
- Theorem 124: Lean theorem 3 of 48, support criterion
- Theorem 125: Lean theorem 4 of 48, strength mono
- Theorem 126: Lean theorem 5 of 48, strength on small set
- Theorem 127: Lean theorem 6 of 48, normalized injective
- Theorem 128: Lean theorem 7 of 48, width bound
- Theorem 129: Lean theorem 8 of 48, strength four width bound
- Theorem 130: Lean theorem 9 of 48, raised strength
- Theorem 131: Lean theorem 10 of 48, raised extends iff
- Theorem 132: Lean theorem 11 of 48, raised counts
- Theorem 133: Lean theorem 12 of 48, padding bound
- Theorem 134: Lean theorem 13 of 48, raised row bound
- Theorem 135: Lean theorem 14 of 48, minimal hit subset
- Theorem 136: Lean theorem 15 of 48, hits primal
- Theorem 137: Lean theorem 16 of 48, forward nonddual
- Theorem 138: Lean theorem 17 of 48, forward size bounds
- Theorem 139: Lean theorem 18 of 48, anchored extension iff
- Theorem 140: Lean theorem 19 of 48, anchored strength iff
- Theorem 141: Lean theorem 20 of 48, anchored forward reduction
- Theorem 142: Lean theorem 21 of 48, hits transversals iff
- Theorem 143: Lean theorem 22 of 48, reverse support iff
- Theorem 144: Lean theorem 23 of 48, sym strength

- Theorem 145: Lean theorem 24 of 48, reverse reduction
- Theorem 146: Lean theorem 25 of 48, reverse counts
- Theorem 147: Lean theorem 26 of 48, strength flip iff
- Theorem 148: Lean theorem 27 of 48, extension flip iff
- Theorem 149: Lean theorem 28 of 48, strength row equiv
- Theorem 150: Lean theorem 29 of 48, extension row equiv
- Theorem 151: Lean theorem 30 of 48, normalized array eq
- Theorem 152: Lean theorem 31 of 48, full forward reduction
- Theorem 153: Lean theorem 32 of 48, normalized reduction sizes
- Theorem 154: Lean theorem 33 of 48, minimize correct
- Theorem 155: Lean theorem 34 of 48, transversals minimize
- Theorem 156: Lean theorem 35 of 48, transversals clutter
- Theorem 157: Lean theorem 36 of 48, transversals involution
- Theorem 158: Lean theorem 37 of 48, forward clutter reduction
- Theorem 159: Lean theorem 38 of 48, forward nonempty edges
- Theorem 160: Lean theorem 39 of 48, dual implies local
- Theorem 161: Lean theorem 40 of 48, fixed yes correct
- Theorem 162: Lean theorem 41 of 48, total reverse correct
- Theorem 163: Lean theorem 42 of 48, total reverse sizes
- Theorem 164: Lean theorem 43 of 48, reverse promise sizes
- Theorem 165: Lean theorem 44 of 48, repeated row lemma
- Theorem 166: Lean theorem 45 of 48, outside column transfer
- Theorem 167: Lean theorem 46 of 48, uncolorable classes cover
- Theorem 168: Lean theorem 47 of 48, uncolorable class count
- Theorem 169: Lean theorem 48 of 48, minimal transversal local test
- Open question: CA(15;2,21,3) and CA(23;2,10,4) stay open
- Open question: Does a 346th column exist for the 170 x 345 array?
- Open question: Exact Ladder minima at 12 and 16 rows
- Open question: The smallest possible number of rows

© 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.