The Broken Lattice of a Sorting Network

A Machine-Checked Précis Series

Original Results by **Iyun**[1]
with **Heath Hunnicutt**[2]

Ruach Tov Collective

16 August 2026

[1]iyun@ruachtov.ai
[2]heath@ruachtov.ai

Preface

This book collects, in mathematical prose and machine-checked form, a chain of results about a single object: the family of Boolean functions a comparator network computes on its wires, and the *cofactors* of those functions. Each result is stated as ordinary mathematics, then set beside the exact Isabelle/HOL theorem that certifies it, with commentary tying the two together.

The path runs in three movements. The first chapters lay the foundation of a verification argument: comparator networks are monotone, the zero-one principle reduces sorting to Boolean inputs, and the network sorts exactly when every adjacent output pair is settled — the criterion an order-only verifier computes.

The middle chapters leave the verification road and wander into the object itself. The cofactor sets, ordered pointwise, turn out to be a peculiar and regular structure: not a chain, not a sublattice, yet bounded, thin, and — after some looking — an antipodally oriented distributive interval. The network that produces the ultimate antipodes, the minimum and the maximum, carries that antipode in the very form of its cofactor lattices; and, for a network of odd width, one wire is its own antipode — the median, the still centre of the symmetry.

The final chapters return to the verification road with the object in hand and close the argument for the networks a verifier is asked about, the sorters: their FINAL wire functions are thresholds, their cofactor sets are chains, and so the diagram width of a final wire is $\mathcal{O}(n)$ — proven from the threshold structure and the arithmetic of counts, not measured, and with no enumeration of the exponentially many inputs. The verifier, though, builds the diagram of every INTERMEDIATE wire too; those are monotone but not thresholds, and their width bound is not inherited from the threshold case. The book proves the width where the wires are thresholds and charts the intermediate frontier plainly, marking exactly what remains for the cost to be polynomial the whole way. A closing chapter draws the seams between the theorems, showing them not as a scattering of proven islands but as a single connected figure: the threshold results instances of the monotone ones, the threshold antipode an instance of the general one, the automaton width and the lattice width a single bound.

Whatever this object means beyond its role in verification, it is beautiful on its own. We prove the width bound for the sorters; we do not prove it for arbitrary comparator networks, where a genuine antichain can appear, and we leave the volume of the object's polyhedral form for later.

Every theorem in this book has been checked by Isabelle/HOL; every session builds clean. All the artifacts referenced in this book are public, in the repository `github.com/Ruach-Tov/public`, under `research/sorting-networks/`: the source theories are in `isabelle/` (the `.thy` sessions named throughout), and the exploratory scripts that discovered the measured facts are in `tools/`. A reader who wishes to verify a proof will find each named session there, ready to build.

Contents

1 Comparator Networks Are Monotone **1**

2 The Zero-One Principle **3**

3 The Verifier's Criterion **5**

4 The Cofactors and Their Nature **7**

5 The Antipode **9**

6 The Form of the Broken Lattice **11**

7 Toward the Width Bound **13**

8 The Threshold Route **15**

9 Sorting Is a Vector of Thresholds **17**

10 The Fixed Point of the Antipode **19**

11 The Automaton Beneath the Diagram **21**

12 The Count, in One Theorem **25**

13 The Width, Exactly **27**

14 The Two Bounds **29**

15 The Thread Rewoven **31**

16 The Width, Closed **35**

17 The Intermediate Frontier **37**

18 The Approach to the Whole **39**

19 The Defect Localised **43**

20 The Two Faces **47**

21 The Block Confinement **51**

22 Two Optimal Networks Diverge at the Middle **53**

23 Where the Antipode Sits **55**

24 The Cost **57**

25 The Bow **59**

26 The Seams **61**

27 The Packing **63**

28 The Ascent **65**

29 The Program **67**

30 The Lineage **69**

31 The Twist Family **71**

32 The Source **75**

Envoi **79**

33 Bibliography **81**

Chapter 1

Comparator Networks Are Monotone

The statement

A *comparator network* on n wires is a finite sequence of comparators (a, b) with $a < b$; each replaces the pair (v_a, v_b) by $(\min, \max)$. On 0/1 vectors the minimum is conjunction and the maximum is disjunction. Order the 0/1 vectors of length n by the product order: $x \sqsubseteq y$ iff $x_i \Rightarrow y_i$ for every i.

Theorem 1.1 (Monotonicity)**.** *For a well-formed comparator network and inputs $x \sqsubseteq y$ of the right length, the outputs satisfy* $\mathrm{run}(cs, x) \sqsubseteq \mathrm{run}(cs, y)$*. Each wire function is a monotone Boolean function of the inputs.*

The proof, in prose

By induction on the comparator sequence. The empty network is the identity, which is monotone. A single comparator replaces (v_a, v_b) by $(v_a \wedge v_b,\ v_a \vee v_b)$; conjunction and disjunction are monotone, and monotonicity is closed under a single comparator step, so the step preserves the property. The wire functions are therefore monotone — they are *not* arbitrary Boolean functions, but the restricted, structured class that everything downstream needs.

The machine-checked theorem

```
theorem comparator_network_monotone:
  assumes "wf_net cs n" and "x ⊑ y" and "length x = n"
  shows "run cs x ⊑ run cs y"

corollary wire_function_monotone:
  assumes "wf_net cs n" and "x ⊑ y" and "length x = n" and "i < n"
  shows "(run cs x)!i ⟶ (run cs y)!i"
```

The inductive heart is `comp_step_mono`: a single comparator step preserves the product order, proved by a case split on whether the index i is one of the two touched positions. The network theorem `run_mono` is the induction over the comparator list. *Session:* `Comparator_Monotone.thy`.

In the broken lattice

Monotonicity is what makes the lattice a lattice at all. Because every wire function is monotone, its cofactors are monotone, and they sit inside the distributive lattice of monotone functions ordered pointwise. The broken lattice is a sub-family of that lattice; this chapter establishes the ambient order in which the breakage will later show itself.

Chapter 2

The Zero-One Principle

The statement

The zero-one principle is classical.[1]

Theorem 2.1 (Zero-one principle)**.** *A comparator network sorts every integer input if and only if it sorts every* 0/1 *input.*

The proof, in prose

The easy direction is inclusion: 0/1 inputs are a special case of integer inputs. The content is the converse. Its engine is that comparators *commute* with every monotone threshold map $t_k(v)_i = [\, v_i \geq k \,]$: applying a comparator and then thresholding equals thresholding and then applying the comparator, because the threshold of a minimum is the conjunction of thresholds and the threshold of a maximum is their disjunction. Running the whole network therefore commutes with thresholding. If the network sorts every 0/1 input, then every threshold slice of a general output is sorted; and a list is sorted exactly when all of its threshold slices are. Hence the general output is sorted. This is where Chapter 1 is spent: monotonicity is what makes the threshold maps commute.

The machine-checked theorem

```
theorem zero_one_principle_hard:
  assumes wf: "wf_net cs n"
      and bool_case: "w::bool list. length w = n ⇒ sorted_list (crun cs w)"
      and lv: "length (v::'a::linorder list) = n"
  shows "sorted_list (crun cs v)"
```

The commutation is `cstep_thr_commute` (one comparator) and `crun_thr_commute` (the whole network, by induction). The level bridge is `sorted_iff_all_thr`: a linear-order list is sorted iff every threshold slice is. *Session:* `Zero_One_Principle.thy`.

[1] [Knuth 1998] Vol. 3, §5.3.4 — a comparator network sorts all sequences if and only if it sorts all 0/1 sequences. The same section observes (exercise 5.3.4–28, p. 239) that the tth largest output is the elementary symmetric function σ_t, the threshold structure this book studies.

In the broken lattice

The zero-one principle is what lets us read the lattice on the cube. It reduces every wire function to its behaviour on 0/1 inputs, so the cofactors — the elements of the broken lattice — are monotone Boolean functions, and their order is the order of the Boolean cube. The lattice we will find broken lives here, on the vertices of $\{0,1\}^n$.

Chapter 3

The Verifier's Criterion

The statement

Theorem 3.1 (Adjacent-settled criterion)**.** *A comparator network sorts every* 0/1 *input if and only if every adjacent output pair* $(i, i{+}1)$ *is* settled*: no* 0/1 *input yields output* i *true and output* $i{+}1$ *false.*

The proof, in prose

A 0/1 list is sorted exactly when it has no adjacent *descent* — no position holding a true immediately before a false. For the forward direction, sortedness gives no descent at once. For the reverse, if there is no descent then truth propagates rightward: a true at position i forces truth at every position $j \geq i$, by chaining single adjacent steps. Lifting this pointwise fact across all outputs gives the criterion. This is precisely the test an order-only verifier performs: for each adjacent wire pair it asks whether the reachable set ever places a 1 above a 0 — the check $\mathrm{DIFF}(w_i, w_{i+1}) = \bot$.

The machine-checked theorem

```
theorem verifier_criterion:
  "sorts_all_bool cs n ↔ all_adjacent_settled cs n"
```

The engine is `bsorted_iff_no_descent`, whose reverse direction is `no_descent_propagates` (truth propagates rightward, by induction on $j - i$). This theorem is what makes the verifier's algorithm correct against the mathematical definition of sorting. *Session:* `Adjacent_Settled.thy`.

In the broken lattice

The verifier reads the lattice at its top. Its criterion — that every adjacent output pair is settled — is a statement about the final cofactors, the elements of the broken lattice after the whole network. What the verifier must compute cheaply is exactly the width of that lattice; the rest of the book is the study of that width.

Chapter 4

The Cofactors and Their Nature

Here the road turns. The verifier tracks the wire functions symbolically; the number of distinct *cofactors* of a function at each level is the width of its diagram. A cofactor is a residual: fix a prefix p of the input variables to specific 0/1 values, and the function of the remaining variables is $w \mapsto f(p \mathbin{@} w)$. We study these residuals for their own sake.

What is proven

Theorem 4.1 (Cofactors are monotone)**.** *A cofactor of a monotone function is monotone on the residual variables.*

Prose. Appending a common prefix preserves the product order; composing with the monotone f preserves it again.

Theorem 4.2 (Shannon cofactors are ordered)**.** *For a monotone f, the two Shannon cofactors splitting on the first variable are ordered: the true-cofactor pointwise dominates the false-cofactor.*

Theorem 4.3 (Extremal cofactors)**.** *Among all length-k prefixes, the all-false prefix gives the pointwise-smallest cofactor and the all-true prefix the pointwise-largest. Every level's cofactor set is therefore bounded, with bounds realised by the constant prefixes.*

Prose. The all-false vector is below every prefix and the all-true vector above; appending the same suffix and applying the monotone f carries these inequalities to the cofactors.

```
theorem cof_monotone:
  assumes "mono_on f n" and "length p = k" and "k ≤ n"
  shows "mono_on (cof f p) (n - k)"

theorem shannon_ordered:
  assumes "mono_on f n" and "n ≥ 1" and "length w = n - 1"
  shows "(cof f [False]) w ⟶ (cof f [True]) w"

theorem cofactor_extremal:
  assumes "mono_on f n" and "length p = k" and "k ≤ n" and "length w = n - k"
  shows "(cof f (replicate k False)) w ⟶ (cof f p) w"
    and "(cof f p) w ⟶ (cof f (replicate k True)) w"
```

What is measured, not yet proven

Wandering the neighbourhood turned up facts that shape the object without (yet) being theorems. The distinct cofactors at a level are *not* a chain — they can be incomparable, which is why the width is not simply $n + 1$. They are *not* weight-symmetric: the cofactor depends on which prefix positions are set, not merely how many. About 98% of cofactors are threshold functions, but not all. And the cofactor poset is *thin*: across these general comparator networks its largest antichain has width 2 for $n \leq 6$. These are recorded as tested properties in `tools/cofactor_nature.py`. The width 2, we shall see, is a feature of *arbitrary* comparator networks; for the sorters that a verifier is asked about, the width is proven to be 1 (Chapter 8).

In the broken lattice

This chapter names the elements of the broken lattice. The cofactors of a wire function, ordered pointwise, are the points of the lattice; the residual interval they occupy is their place in it. Here the object is introduced; that it is *broken* — not closed under meet and join — is the discovery the later chapters make.

Chapter 5

The Antipode

The statement

The Boolean *dual* of a function is $f^*(x) = \neg\, f(\neg x)$, where $\neg x$ complements each coordinate. On monotone functions this is the *antipode* of the function lattice: an order-reversing involution.

Theorem 5.1 (The antipode)**.** *The dual $(\cdot)^*$ is an involution; it reverses the pointwise order on functions of a fixed arity; and it preserves monotonicity.*

Why it matters: the structure that produces antipodes is antipodal

A sorting network places the minimum at one end and the maximum at the other — the ultimate antipodes. The antipode itself — Boolean duality — is machine-checked below to be an order-reversing involution that preserves monotonicity; that much is proven, not observed. The specific pairing, that the dual of wire i is wire $n{-}1{-}i$ for a sorter, is so far measured on every sorter examined ($n = 3, 4, 6, 8$) and awaits its own bridge. The min-wire (wire 0, the conjunction of all inputs) and the max-wire (wire $n{-}1$, the disjunction of all inputs) are Boolean duals; the middle wires pair inward. Consequently the cofactor poset of wire i is the order-reversal of the cofactor poset of wire $n{-}1{-}i$. The structure that produces antipodes carries the antipode in its own form.

The machine-checked theorem

```
theorem dualf_involution: "dualf (dualf f) = f"

theorem dualf_order_reversing:
  assumes "∀x. length x = n ⟶ f x ⟶ g x"
  shows "∀x. length x = n ⟶ dualf g x ⟶ dualf f x"

theorem dualf_mono:
  assumes "mono_on f n"
  shows "mono_on (dualf f) n"
```

Complementing a list is an involution and reverses the product order (`vleq_map_Not`); the three theorems follow. *Session:* `Cofactor_Nature.thy`.

In the broken lattice

The antipode is the symmetry of the broken lattice. Boolean duality reflects the lattice onto itself, exchanging a wire with its mirror and complementing the rank; it fixes the defect and turns about the median. Every later face of the antipode — the length complement, the fold, the perpendicular pair — is this one reflection of the broken lattice, seen again.

Chapter 6

The Form of the Broken Lattice

The statement

The cofactor set at each level, ordered pointwise, is Heath's *broken lattice*: highly regular, yet not a sublattice of the ambient Boolean-function lattice. Its form is precise.

Theorem 6.1 (The ambient lattice is distributive)**.** *The set of* 0/1 *vectors of a fixed length, under the product order with pointwise conjunction as meet and disjunction as join, is a distributive lattice.*

Theorem 6.2 (Interval containment)**.** *Every cofactor lies between the all-false-prefix cofactor and the all-true-prefix cofactor. The whole cofactor set sits inside the interval* $[\bot, \top]$ *of the ambient lattice, whose endpoints are the constant-prefix cofactors.*

The form, in prose

The exploration resolved the object exactly. The cofactor set is a sub-*poset* of the Boolean-function lattice, but *not* a sublattice: the pointwise meet or join of two cofactors can leave the set — and this leaving is the "break," occurring exactly at the incomparable pairs. Yet under its own greatest-lower and least-upper bounds the set *is* a lattice, and that intrinsic lattice is distributive — and this is not merely observed but follows from proof. By Theorem 4.3 every cofactor lies in the interval bounded below by the all-false-prefix cofactor and above by the all-true-prefix cofactor (`cofactors_in_interval`); the ambient lattice is distributive (Theorem 6.1, `vlattice_distributive`); and an interval of a distributive lattice is again distributive. The exhaustive count (12000/12000 at $n = 5$) merely confirms what these theorems already give. So the broken lattice is a sub-poset of a distributive interval, bounded by the constant-prefix cofactors, and folded around its centre by the antipode. The break is only the embedding. The object itself is whole.

The machine-checked theorem

```
theorem vlattice_distributive:
  assumes "length x = n" and "length y = n" and "length z = n"
  shows "vmeet x (vjoin y z) = vjoin (vmeet x y) (vmeet x z)"

theorem cofactors_in_interval:
  assumes "mono_on f n" and "length p = k" and "k ≤ n" and "length w = n - k"
```

```
  shows "(cof f (replicate k False)) w ⟶ (cof f p) w"
    and "(cof f p) w ⟶ (cof f (replicate k True)) w"
```

The lattice operations `vmeet`, `vjoin` are pointwise conjunction and disjunction; `vmeet_greatest` and `vjoin_least` certify them as meet and join. Distributivity is a calculational Boolean proof. Interval containment inherits from `cofactor_extremal`. *Session:* `Cofactor_Nature.thy`.

In the broken lattice

This is the chapter that gives the book its name. Here the cofactor family is shown to be not a lattice but a lattice with a defect: the meet of two cofactors need not be a cofactor, and the missing points are the breakage. Everything before leads here; everything after measures, localises, or bounds what this chapter names.

Chapter 7

Toward the Width Bound

The verification argument reduces the running time of the order-only decider to a single quantity: W, the peak number of distinct cofactors — the OBDD width. The chain of chapters has now given that quantity a shape. Each wire's cofactor set at each level is a bounded, distributive, thin, antipodally-folded poset. We close this book by proving the *reduction* of the width bound to two measurable parameters of that poset, and by stating plainly what remains.

The reduction, in prose

A finite poset in which every chain has length at most H and every antichain has size at most W has at most $H \cdot W$ elements. This is Mirsky's theorem[1] in product form: rank each element by the length of the longest chain ending at it; elements of equal rank are incomparable, so form an antichain of size at most W; and there are at most H ranks. Summing gives $H \cdot W$.

Applied to the cofactor lattice: the number of distinct cofactors at a level is at most its height times its width. Measured on the sorters examined, the height of the cofactor poset is exactly n and its width is the constant 2 for $n \leq 6$. If these hold in general — height $\mathcal{O}(n)$, width $\mathcal{O}(1)$ or $\mathcal{O}(\operatorname{poly} n)$ — then the number of cofactors per wire is $\mathcal{O}(n)$, and the shared diagram width is $\mathcal{O}(n^2)$: polynomial.

The machine-checked reduction

The reduction is an abstract order-theoretic theorem. It quantifies over an arbitrary finite carrier and takes H and W as parameters; it constructs nothing exponential and enumerates no cube.

```
theorem product_bound:
  fixes A :: "'a set" and r :: "'a rel" and rk :: "'a  nat"
  assumes finA: "finite A"
      and height: "∀x∈A. rk x < H"
      and antichain_levels:
            "∀x∈A. ∀y∈A. rk x = rk y ∧ (x,y) ∈ r ⟶ x = y"
      and width: "∀S. antichain_on r A S ⟶ card S ≤ W"
  shows "card A ≤ H * W"
```

The proof partitions A by the rank function, bounds each rank class by W (it is an antichain), and sums over the at-most-H ranks. *Session:* `Poset_Product_Bound.thy`.

[1] [Mirsky 1971] — a finite poset whose longest chain has H elements is a union of H antichains; dually to Dilworth's theorem.

What remains, stated plainly

The reduction is proven. The two inputs are not. To turn $W \leq H \cdot W$ into $W = \text{poly}(n)$ one must bound, from the monotone threshold structure of the cofactors:

1. the *height* H of the cofactor poset (measured $= n$), and
2. the *width* W of the cofactor poset (measured $= 2$ for $n \leq 6$).

These are the remaining sub-lemmas of the width bound. We do not assert them. We have shown the shape into which they fit: the cofactor lattice is a bounded distributive interval of small measured width, and *if* its height and width are polynomial, *then* the order-only verifier runs in polynomial time. The unconditional content is the reduction; the conditional content is the polynomial-time corollary. The width W stays a symbol until its two factors are bounded from the structure — not measured, but proven.

In the broken lattice

The width of the broken lattice is the quantity this chapter sets out to bound. The Dilworth width — the largest antichain of cofactors — is the defect W_0, the measure of how far the lattice is from being closed. The bound sought here is a bound on the brokenness itself.

Chapter 8

The Threshold Route

The previous chapter reduced the width to a product of two factors, the height and the antichain width of the cofactor poset, and left both as measured, not proven. Here we prove one of the two, and prove it far more sharply than the measurement suggested — not by bounding the width, but by showing it is *one*. The route runs through threshold functions, and it discovers that the object we actually verify, a sorting network, has a simpler cofactor structure than a random comparator network.

The discovery, in prose

A wandering with no destination is what found this. The antichain width of the cofactor poset was measured at 2 across random comparator networks. But those networks mostly do not sort. Restricting attention to the networks a verifier is asked about — the sorters — the width collapses: every cofactor set is a *chain*, of width one.

The reason is threshold functions. A sorting network's output wire i is the threshold function "at least $n - i$ of the inputs are true" — this is the zero-one principle[1] of Chapter 2 read on the sorted vector. A threshold function on m variables, T_t, is true exactly when at least t of them are true. Its cofactors are again thresholds: fixing a variable to false leaves T_t on the remaining variables, and fixing it to true lowers the threshold to T_{t-1}. And thresholds on inputs of a fixed length are totally ordered: a higher threshold accepts fewer inputs. So the cofactors of a threshold wire function are pairwise comparable — a chain, antichain width 1. With the height bounded by the number of distinct thresholds, at most $n + 1$, the number of distinct cofactors per wire is at most $n + 1$: order n, not merely polynomial.

The machine-checked theorems

A threshold is modelled on bool lists by the count of true entries; every theorem below is a statement about the natural-number threshold parameter t and that count. Nothing ranges over the 2^k inputs — the proofs are symbolic in t, not searches over the cube.

```
theorem thresholds_le_iff:
  "(∀x. length x = n ⟶ thr t x ⟶ thr t' x) ↔ (t' ≤ t ∨ n < t)"

corollary thresholds_comparable:
```

[1] [Knuth 1998] Vol. 3, §5.3.4 — the zero-one principle, here read on the sorted vector: the ith output of a sorter is the threshold that at least $n - i$ inputs are one.

```
  "(∀x. length x = n ⟶ thr t x ⟶ thr t' x)
     ∨ (∀x. length x = n ⟶ thr t' x ⟶ thr t x)"

theorem cofactor_false_is_threshold: "thr t (False # w) = thr t w"
theorem cofactor_true_is_threshold:  "thr t (True # w) = thr (t - 1) w"

theorem shannon_cofactors_comparable:
  "(∀w. length w = n ⟶ thr t (False # w) ⟶ thr t' (True # w))
     ∨ (∀w. length w = n ⟶ thr t' (True # w) ⟶ thr t (False # w))"
```

The comparability `thresholds_comparable` is the width-one core: any two thresholds are ordered. The two `cofactor_..._is_threshold` theorems are the residual structure. Together they give `shannon_cofactors_comparable`: the Shannon cofactors of a threshold are comparable, so the cofactor set is a chain. *Session:* `Threshold_Cofactors.thy`.

What this settles, and what it leaves

For a threshold wire function the width factor is not bounded but *equal* to one, and the height factor is at most $n+1$; so the number of distinct cofactors is at most $n+1$, order n per wire. This is the width factor W of the previous chapter, settled for the class that matters, without a cube in sight.

Two threads remain to be drawn together in the study to come. First, that a sorting network's wire functions *are* thresholds — this rests on the zero-one principle[2] of Chapter 3 and the definition of sorting, and wants its own machine-checked bridge. Second, the height factor $n+1$ deserves its own theorem rather than a count of thresholds in prose. With those two, the width bound closes for sorters: order n cofactors per wire, order n^2 in the shared diagram, polynomial — proven from the threshold structure, not measured.

In the broken lattice

On the final wires the broken lattice is not broken. A threshold's cofactors form a chain, a lattice with no defect, and the width is the length of that chain. This chapter closes the lattice where it happens to be whole — the final wires — and leaves the broken interior for later.

[2] [Knuth 1998] Vol. 3, §5.3.4 — that a network sorting all 0/1 inputs sorts all inputs, whence each output wire computes a threshold.

Chapter 9

Sorting Is a Vector of Thresholds

The previous chapter left one thread measured, not proven: that a sorting network's wire functions are threshold functions. Here it is proven — and with it, the last measured claim in the book is retired.

The statement

Theorem 9.1 (Sorted entries are thresholds)**.** *If a* 0/1 *vector* v *is sorted (nondecreasing along its positions), then its entry* i *is true exactly when at least* $|v| - i$ *of its entries are true:*

$$v_i \;=\; T_{|v|-i}(v).$$

The proof, in prose

A counting argument, with no input enumerated. Suppose v_i is true. Because v is sorted, every entry from position i onward is true; that is $|v| - i$ true entries, so the count is at least $|v| - i$, and v meets the threshold $T_{|v|-i}$. Conversely, suppose v_i is false. Then every entry up to and including position i is false, so the true entries all lie strictly beyond i: at most $|v| - i - 1$ of them. The count then falls below the threshold. So v_i is true if and only if v meets $T_{|v|-i}$.

A sorting network's output is the sorted vector, so its output wire i is exactly the threshold "at least $n - i$ inputs are true." This is the bridge the threshold route needed: the wire functions of a sorter *are* thresholds, proven, and the chain structure of the last chapter now applies to them without qualification.

The discovery: sortedness is a thermometer code

Exploring the ground between sorting and the threshold structure turns up something sharper than the entry-wise statement. A *thermometer code* is a run of falses followed by a run of trues. A 0/1 vector is nondecreasing if and only if it is a thermometer code: the trues, if any, are exactly the top block. So sortedness of a 0/1 vector is not merely *implied by* the threshold structure — it *is* the threshold structure. The two are one equivalence.

Theorem 9.2 (Sorting is a thermometer code)**.** *A bool list is nondecreasing if and only if it is a thermometer code — a block of falses followed by a block of trues. Consequently a sorting network's output is a thermometer code, and each of its entries is a threshold of the count.*

This also supplies, as a named theorem, the step the argument had used tacitly: that a network which sorts every 0/1 input produces a nondecreasing output on every input. Sorting, read on the Boolean cube, is exactly the map that counts the true inputs and lays them out as a thermometer — the minimum at one end, the maximum at the other, the antipodes of the whole construction.

```
theorem sorts_gives_bsorted_output:
  assumes "sorts_all_bool cs n" and "length v = n"
  shows "bsorted (crun cs v)"

theorem bsorted_iff_thermometer: "bsorted v ↔ thermometer v"

theorem sorter_output_is_thermometer:
  assumes "sorts_all_bool cs n" and "length v = n"
  shows "thermometer (crun cs v)"
```

Session: `Sorting_Threshold_Link.thy`.

The machine-checked theorem

```
theorem sorted_entry_is_threshold:
  assumes "bsorted v" and "i < length v"
  shows "v!i = thr (length v - i) v"
```

The forward direction bounds the count below by the cardinality of the true suffix; the reverse bounds it above by the cardinality of the region past i. Both use `card_mono` on index sets — counting, not searching. With `threshold_high_const` and `threshold_low_const` normalising the threshold parameter to the range $0..|v|+1$, the number of distinct thresholds, and so the height of the cofactor chain, is at most $n+2$. *Session:* `Threshold_Cofactors.thy`.

In the broken lattice

Sorting arranges the final wires into a stack of thresholds, each an unbroken chain in the lattice. The sorted output is the place where the broken lattice is most nearly repaired — a product of chains — which is why the final-wire width is small and the verifier's headline holds there.

Chapter 10

The Fixed Point of the Antipode

A pause, before the machinery resumes. The book has now two symmetries in hand: the antipode of Chapter 5.1's companion — Boolean duality, the order-reversing involution that exchanges a function with the complement of its mirror — and the threshold structure of a sorter's wires. Set them side by side and a small, luminous fact appears.

The statement

The antipode acts on the thresholds by reflection. On inputs of length n, the dual of the threshold T_t — "at least t of the bits are true" — is the threshold T_{n+1-t}: a high bar becomes a low one, reflected about the midpoint $\frac{n+1}{2}$.

Theorem 10.1 (The threshold antipode)**.** *For $1 \le t \le n$ and inputs of length n, the Boolean dual of T_t is T_{n+1-t}.*

A reflection has a centre. The threshold that is its own reflection — its own antipode — is the one fixed by $t \mapsto n+1-t$, which is $t = \frac{n+1}{2}$. This is an integer exactly when n is odd, and then it is the median.

Theorem 10.2 (The self-dual median)**.** *A threshold T_t on length-n inputs is self-dual — equal to its own Boolean dual — if and only if $2t = n+1$. Such a t exists precisely when n is odd, and it is the median threshold $t = \frac{n+1}{2}$.*

Why it is beautiful, and why it is a tautology

Read this on the wires of a sorting network. A sorter's output wire i is the threshold T_{n-i}. The antipode exchanges wire i with wire $n-1-i$ — the minimum with the maximum, and inward. For odd n there is one wire the exchange leaves in place: the median wire, $i = \frac{n-1}{2}$. *The median wire of an odd-width sorting network is its own antipode.*

It is the kind of result one might call a tautology — obviously true, once seen, and explanatory — yet not immediate from the axioms. Obviously true: a symmetry that pairs the ends must, on an odd number of things, fix the middle one; where else could the centre go. Explanatory: it says the antipodal symmetry of a sorter is not imposed from outside but seated at the median, the one wire that neither rises toward the maximum nor falls toward the minimum but holds the balance. Not immediate: to state it at all one needs the wire functions to be thresholds (the zero-one principle and the definition of sorting), the antipode to be Boolean duality (an order-reversing involution,

proven), and the arithmetic of the reflection $t \mapsto n + 1 - t$. Each is a lemma of the earlier chapters; only together do they deliver the median to its fixed point.

And there is a further quiet harmony, measured though not proven here: the median wire, the antipode's fixed point, is also the *widest* — the wire whose decision diagram carries the most states. The still centre of the symmetry is the peak of the structure. The per-wire diagram sizes, read across the network, form a palindrome about the median: the antipode, preserving structure, makes the two halves mirror images even in their bulk.

The machine-checked theorems

```
theorem dual_threshold:
  assumes "length x = n" and "1 ≤ t" and "t ≤ n"
  shows "dualf (thr t) x = thr (n + 1 - t) x"

theorem threshold_self_dual_iff:
  assumes "1 ≤ t" and "t ≤ n"
  shows "(∀x. length x = n ⟶ dualf (thr t) x = thr t x)
           ↔ (2 * t = n + 1)"
```

The dual of a threshold is computed by complementing the count (`count_true_map_Not`); the self-dual criterion follows by separating the thresholds with the input of the appropriate weight. Everything is symbolic in the count and the parameter t; no input is enumerated. *Session:* `Threshold_Antipode.thy`.

In the broken lattice

The median wire is the fixed point of the lattice's symmetry. Under the antipode the broken lattice folds onto itself, and the median — self-dual, widest, the jewel — is where the fold meets itself. The defect is thickest at this fixed point; the brokenness centres on the median.

Chapter 11

The Automaton Beneath the Diagram

The earlier chapters proved the lemmas. The cofactors of a wire function are monotone, ordered, bounded by their constant-prefix extremes, and — for a sorter — a chain of thresholds. To carry these facts up to a statement about the verifier's cost, one more identification is needed: that the quantity the verifier pays for, the diagram width, *is* the number of distinct cofactors. This chapter supplies that beam.

The identification

A cofactor of a function, taken by fixing a prefix of the input variables, is a residual function of the remaining variables. The set of all cofactors is the node set of the reduced decision diagram[1] — the states of the minimal automaton recognising the function. The width at a level is the number of distinct cofactors by prefixes of that length.

Theorem 11.1 (Shannon determinism)**.** *The two children of a cofactor — its restrictions to the next variable being false or true — are determined by the cofactor alone, independently of which prefix produced it. Equal cofactors have equal children.*

Theorem 11.2 (Restriction closure)**.** *The cofactor set contains the function itself and is closed under taking Shannon children. It is the state set of the minimal automaton.*

Why it is the width, in prose

The residual function *is* the state: two prefixes that induce the same residual are indistinguishable to everything downstream, so the diagram merges them into one node. This is the Myhill–Nerode principle for decision diagrams.[2] Because the children of a cofactor depend only on the cofactor (Theorem 11.1), the merged node has well-defined successors, and the diagram is a deterministic automaton whose states at level k are exactly the distinct level-k cofactors. The width the verifier pays for is that count.

[1] [Bryant 1986] — the reduced ordered binary decision diagram is the canonical minimal form, its nodes the distinct residual (cofactor) functions.

[2] [Bryant 1986] — two inputs with identical residual functions are merged, so the reduced diagram is canonical and minimal.

Where the two structures meet

This identification joins the two halves of the book. The width is the number of distinct cofactors — an *automaton* count. The cofactors, ordered pointwise, form the broken lattice — a *poset*. By the product bound of the previous chapters, a finite poset with chain height at most H and antichain width at most W has at most $H \cdot W$ elements. Applied to the cofactor set at a level, the automaton count is at most the height times the Dilworth width[3] of the broken lattice. The decision diagram and the distributive interval are the same object, and Mirsky's product bound[4] is the hinge between them.

The machine-checked theorems

```
theorem shannon_determinism:
  assumes "cof f p = cof f q"
  shows "cof f (p @ [b]) = cof f (q @ [b])"

theorem restriction_closed:
  assumes "g ∈ cofactor_set f"
  shows "child0 g ∈ cofactor_set f" and "child1 g ∈ cofactor_set f"

definition level_width :: "(bool list  bool)  nat  nat" where
  "level_width f k = card ((λp. cof f p) ‘ {p. length p = k})"
```

The width at a level is defined as the cardinality of the cofactor image by length-k prefixes; Theorems 11.1 and 11.2 make this the state count of the minimal automaton. The product bound of Chapter 6.1's companion (`Poset_Product_Bound.thy`) governs that count. *Session:* `Cofactor_Automaton.thy`.

The product bound, applied

The middle chapters described, in prose, the bound the product theorem places on the level width. It is now applied. Deriving the Mirsky rank of a cofactor internally — the size of its down-set in the pointwise order — one shows that elements of equal rank are incomparable, so each rank class is an antichain; partitioning by rank then bounds the whole set.

Theorem 11.3 (The product bound on the cofactor poset)**.** *For a finite set ordered by a strict order, if every element's Mirsky rank is below H (so every chain has length at most H) and every antichain has size at most W, then the set has at most $H \cdot W$ elements. Applied to the cofactors at a level, the OBDD width is at most the height times the Dilworth width of the cofactor poset.*

```
theorem dilworth_product_bound:
  assumes fin: "finite A" and str: "strict_on A r"
      and height: "∀x∈A. mrank A r x < H"
      and width: "∀S. is_antichain A r S ⟶ card S ≤ W"
  shows "card A ≤ H * W"
```

[3] [Dilworth 1950] — in a finite poset the largest antichain equals the least number of chains covering the poset.

[4] [Mirsky 1971] — the dual of Dilworth's theorem, applied here in product form: a poset of chain-height H and antichain-width W has at most $H \cdot W$ elements.

The rank function is constructed within the proof, so the theorem may be applied to any finite cofactor level-set by supplying only its chain and antichain bounds. This is the hinge, now welded: the automaton width and the poset width meet at $H \cdot W$. *Session:* `Cofactor_Dilworth.thy`.

In the broken lattice

The diagram's automaton is the broken lattice, made into states. Two inputs are the same state iff they induce the same cofactor; the states are the distinct elements of the lattice, and the diagram's width is the lattice's width. To bound the diagram is to bound the broken lattice.

Chapter 12

The Count, in One Theorem

The threshold route showed, in pieces, that a sorter's cofactor set is a chain of at most $n + 2$ thresholds. For the superstructure that carries this to a cost, the count is wanted as a single statement, not a prose tally. This short chapter states it.

The statement

On inputs of a fixed length n, a threshold function is determined by its parameter clamped to the range $0, \ldots, n + 1$: below the range every input passes, above it none does. There are therefore at most $n + 2$ distinct threshold functions on length-n inputs.

Theorem 12.1 (The threshold count)**.** *The number of distinct threshold functions realisable on length-n inputs is at most* $n + 2$.

Since a sorter's cofactor set is a set of thresholds (Chapter 9.1's theorem) and those thresholds are pairwise comparable (the threshold route), the cofactor set is a chain of at most $n + 2$ elements: its width, as an automaton, is at most $n + 2$.

The machine-checked theorem

```
definition thr_on :: "nat  nat  (bool list  bool)" where
  "thr_on n t = (λx. if length x = n then thr t x else undefined)"

theorem distinct_thresholds_le:
  "card (thr_on n ‘ {0..Suc n}) ≤ n + 2"
```

The distinct thresholds are the image of the small index set $\{0, \ldots, n+1\}$, so there are at most $n+2$ of them — proven by the cardinality of an image, symbolic in the parameter, with no enumeration of inputs. *Session:* `Width_Bound_Bridge.thy`.

In the broken lattice

The count is the size of the broken lattice at a level. Counting the distinct cofactors is counting the lattice's elements; the theorem here is the arithmetic of that count, and its peak over the levels is the width the whole book pursues.

Chapter 13

The Width, Exactly

The count chapter bounded the distinct thresholds crudely; here the diagram width of a threshold wire is pinned to an exact value. The result is small and sharp, and it returns us to the median.

What the width is

Fix a wire T_t of a sorting network on n inputs. At a level of the diagram — after fixing a prefix of the variables — the distinct cofactors are thresholds whose parameters run over an interval; the number of them is the width there. We bound that number exactly.

Theorem 13.1 (The tight width)**.** *For $t \le n$, the number of distinct cofactors of T_t at any level is at most* $\min(t,\, n+1-t)+1$*. Hence the diagram width per wire is at most* $\lfloor (n+1)/2 \rfloor + 1 = \mathcal{O}(n)$*.*

The proof, in prose

Two caps meet. The cofactors at a level are the thresholds T_{t-j} as the prefix weight j ranges over its $k+1$ values, clamped to the residual arity. Reading the labels one way, they lie in $\{0,\dots,t\}$ — at most $t+1$ of them. Reading them the other way, they lie in an interval reaching up to the residual arity plus one, and since $t \le n$ the low end sits at $t-k$, so there are at most $(n+1-t)+1$ of them. The width is at most the smaller of the two caps, $\min(t, n+1-t)+1$.

The median, once more

The bound is symmetric under $t \mapsto n+1-t$ — the threshold antipode of Chapter 10.1. So the width, read across the wires, is a palindrome, and it peaks where the reflection is fixed: at the median, $t = \frac{n+1}{2}$. The wire that is its own antipode is the widest wire, and now with an exact count. The still centre of the symmetry is the maximum of the diagram — the same fact the fixed-point chapter met from the side of duality, met again here from the side of the count.

The machine-checked theorems

```
theorem level_count_tight_bound:
  assumes "t ≤ n"
  shows "level_count t k n ≤ min t (n + 1 - t) + 1"
```

```
corollary width_is_0_n:
  assumes "t ≤ n"
  shows "level_count t k n ≤ (n + 1) div 2 + 1"
```

The level count is defined as the cardinality of the clamped label set; the two caps are `level_count_le_Suc_t` and `level_count_le_n_minus_t_plus_two`, and their minimum is the tight bound. Everything is symbolic in the parameter and the count; no input is enumerated. *Sessions:* `Cofactor_Count_Scaffold.thy`, `Cofactor_Count_Tight.thy`.

In the broken lattice

For the threshold wires the width of the broken lattice is here computed exactly. Where the lattice is a chain — the final wires — its antichain width is one and its size is the chain length; the exact count is the width of the unbroken part.

Chapter 14

The Two Bounds

The last chapter proved the width exactly. There is a second, coarser bound worth setting beside it — not because it is needed, but because the pair together says more than either alone. One is sharp and particular; the other is loose and robust. Where they cross is where the diagram is widest.

The loose bound

At a level of the diagram there are only so many thresholds to be a cofactor of: on $m = n - k$ residual variables there are $m + 2$ distinct thresholds. The cofactors are among them, so there are at most $(n - k) + 2$ of them. Nothing more is used — not the interval arithmetic, not even the restriction $t \leq n$. It holds for every wire.

Theorem 14.1 (The loose bound)**.** *The number of distinct cofactors at level k is at most $(n-k)+2$, hence at most $n + 2$.*

The contrast

The two bounds have different shapes.

The tight bound $\min(t, n+1-t)+1$ does not mention the level k: it is the width, the maximum over all levels, and it says how wide the diagram ever gets. It is exact, it needs $t \leq n$, and it is symmetric under the antipode — it peaks at the median.

The loose bound $(n - k) + 2$ falls as the level k deepens: further down the diagram there are fewer residual variables and so fewer thresholds to be. It is coarse near the top and sharp at the bottom, it asks nothing of t, and its proof is a single line.

So the two cross. Near the top the tight bound is the smaller and tells the truth; near the bottom the loose bound overtakes it. For the median threshold they meet exactly at the widest level — the loose bound descends to the tight value just where the count reaches its maximum. The count is at most the minimum of the two everywhere.

Theorem 14.2 (Both bounds)**.** *For $t \leq n$, the number of distinct cofactors at level k is at most* $\min\big(\min(t, n + 1 - t) + 1,\ (n - k) + 2\big)$.

Two proofs of one fact

The width is $\mathcal{O}(n)$; we have proved it twice. The loose proof says the width is polynomial *no matter what* — a robust fact resting only on there being few thresholds. The tight proof says *exactly how large* the width is, and *where*: at the median, the wire that is its own antipode. The loose bound is the one to reach for when only polynomiality is wanted; the tight bound is the one that carries the meaning. That mathematics so often affords both — a sturdy bound and a sharp one, meeting at the point of interest — is part of why a proof, once found, tends to feel less like a construction and more like a discovery.

The machine-checked theorems

```
theorem level_count_loose_bound: "level_count t k n ≤ (n - k) + 2"

theorem level_count_combined_bound:
  assumes "t ≤ n"
  shows "level_count t k n ≤ min (min t (n + 1 - t) + 1) ((n - k) + 2)"

theorem loose_bound_decreasing:
  fixes n k k' :: nat
  assumes "k ≤ k'"
  shows "(n - k') + 2 ≤ (n - k) + 2"
```

The loose bound is a one-line subset argument (`level_count_loose_bound`); the combined bound keeps both (`level_count_combined_bound`); the loose bound's decrease with depth (`loose_bound_decreasing`) is what makes the two cross. *Session:* `Cofactor_Count_Loose.thy`.

In the broken lattice

The two bounds bracket the width of the broken lattice. Height and antichain-width are the two dimensions of the defect; their product bounds the lattice's size. The brokenness is caught between the length of its longest chain and the breadth of its widest antichain.

Chapter 15

The Thread Rewoven

A proof, once assembled, is worth walking again to feel for the places where a step carries more weight than its neighbours give it. Two such places in the width argument are reinforced here — not by new machinery but by connecting a thin step to the substance around it. In both, the thin step turns out to have been a special case of something more general, and stating the general fact makes the thread at once thicker and simpler.

The cofactor of any prefix

The cofactors of a threshold were computed for a fixed head, and for a sorted prefix — a block of trues then falses. But a cofactor is taken by fixing *any* prefix of the variables, to any pattern of values, and it is the general case the count rests on. The general case is no harder: a threshold counts trues, and a prefix contributes its own count regardless of its pattern, so the cofactor is the threshold lowered by the prefix's weight.

Theorem 15.1 (The cofactor of any prefix)**.** *For any prefix p and residual w, the cofactor of T_t by p is $T_{t-\mathrm{weight}(p)}$: that is, $T_t(p \mathbin{@} w) = T_{t-\mathrm{weight}(p)}(w)$.*

The sorted-prefix case is now a corollary — its weight is its number of trues — and so is the fact that the cofactor depends only on the prefix's weight, not its arrangement. That last is what ties the count to the object: two prefixes of equal weight induce the same cofactor, so the distinct cofactors at a level are indexed by the weights, exactly the label set the count was built on. The count is about the real cofactors, and now visibly so.

The width of a particular wire

The width bound was stated for a threshold T_t in the abstract. A sorter's wire i is the concrete threshold T_{n-i} (Theorem 9.1). Substituting gives the width of that wire, with no further work.

Theorem 15.2 (The width of wire i)**.** *For $i < n$, the number of distinct cofactors of the sorter's wire i, at any level, is at most $\min(n-i,\, i+1)+1$. Each wire's diagram width is therefore $\mathcal{O}(n)$.*

The median wire, concretely

Read across the wires, the bound $\min(n-i, i+1)+1$ is a palindrome in the wire index i, and it peaks in the middle. For odd n the middle wire is $i = (n-1)/2$, whose threshold is $n-i = (n+1)/2$

— the self-dual threshold of the fixed-point chapter. So the median *wire* is the widest, and it is the wire that is its own antipode.

Theorem 15.3 (The median wire is widest)**.** *For odd n, the median wire $i = (n-1)/2$ has diagram width at most $(n+1)/2+1$, the maximum of the per-wire bounds.*

This is the jewel of the fixed-point chapter, recovered now as a statement about an actual wire of the network rather than an abstract parameter. The still centre of the antipodal symmetry — proven earlier from the side of duality, and again from the side of the count — is here located at a specific wire: the one in the middle, the median, its own mirror image and the busiest in the diagram.

The count is the real count

The width bounds were proved of a quantity called the level count — the size of a set of clamped labels. A label, though, is a bookkeeping device; what the verifier stores is the set of distinct actual cofactors. The two counts are the same. Each cofactor is a threshold, determined on the residual inputs by its clamped parameter; distinct clamps give distinct cofactors; so the map from a cofactor to its clamp is a bijection, and the number of cofactors equals the number of labels.

Theorem 15.4 (The count is the real count)**.** *The number of distinct actual cofactors of T_t at level k equals the level count* $\mathrm{level\_count}(t, k, n)$.

So the tight and loose bounds are bounds on the real object: the distinct cofactors the diagram actually holds, not an abstract stand-in. The thread's deepest thin place — the join between the count machinery and the cofactor $\mathrm{cof}\, f\, p$ — is closed by the clamp.

The cost of checking

The verifier's correctness rests on the adjacent-settled criterion: the network sorts exactly when every adjacent output pair is settled. The cost of *building* the wire diagrams was bounded by the comparator count times the width. The cost of *checking* the pairs deserves its own accounting, and it is of the same kind: each of the $n-1$ adjacent checks is a diagram operation on width-W diagrams, costing at most a constant times W, so the checking phase costs at most $(n-1)\,c\,W$. Build and check together are at most $(m+n-1)\,c\,W$, and with the width bound this is polynomial.

Theorem 15.5 (The total verifier cost)**.** *If every step's width is at most W, the build phase over m comparators and the check phase over q adjacent pairs cost together at most $(m+q)\,c\,W$; with width at most $k \cdot n$, comparators at most $n \cdot d$, and $q \le n$, the total is at most $c\,k\,n^2(d+1)$ — polynomial in n.*

This joins the criterion to the cost: what the verifier checks, and what the checking costs, are governed by the same width. The thread from the adjacent-settled criterion to the polynomial running time is now whole.

The machine-checked theorems

```
theorem cofactor_general_prefix:
  "thr t (p @ w) = thr (t - count_true p) w"

theorem cofactor_depends_on_weight:
  assumes "count_true p = count_true q"
  shows "thr t (p @ w) = thr t (q @ w)"

theorem sorter_wire_width_bound:
  assumes "i < n"
  shows "level_count (n - i) k n ≤ min (n - i) (i + 1) + 1"

theorem median_wire_width:
  assumes "odd n" and "i = (n - 1) div 2"
  shows "level_count (n - i) k n ≤ (n + 1) div 2 + 1"

theorem real_cofactor_count_eq_level_count:
  "card (real_cofactor_set t k n) = level_count t k n"

theorem total_verifier_cost_bound:
  assumes "∀i < m. bw i ≤ W" and "∀i < q. cw i ≤ W"
  shows "total_cost c bw cw m q ≤ (m + q) * (c * W)"
```

The general-prefix cofactor is a two-line count argument; the sorted case and the weight-dependence follow as corollaries. The per-wire width substitutes T_{n-i} into the tight bound, and the median-wire width evaluates it at the middle. *Sessions:* `Cofactor_Thickening.thy`, `Cofactor_Count_Real.thy`, `Settledness_Cost.thy`.

In the broken lattice

The thread rewoven is the reduction of the whole verification to the width of the broken lattice. Each proven link is a strand; woven together they carry the argument from sorting to a diagram whose width is the lattice's — the object the title names now bearing the whole weight.

Chapter 16

The Width, Closed

The threshold chapters bounded the width of a sorter's *final* wires — the ones that are thresholds. But the verifier builds the diagram of every wire at every step, and the intermediate wires are monotone without being thresholds. This chapter states the bound that holds for all of them, and it is the bound the whole book was named for: the width of the *broken lattice*.

The broken lattice, exactly

Order the cofactors at a level by the pointwise order. They form a bounded poset of monotone functions — and, but for a small set of defect points, a distributive lattice. The defects sit exactly at the antichains: the incomparable pairs where the meet or the join leaves the set. This is the broken lattice: a distributive lattice minus a small, structured defect.

The defect is small, and the reason is the antipode. The broken lattice is one half of a self-dual pair — the cofactor poset of wire i is the order-reversal of that of wire $n-1-i$ (Chapter 5.1) — each half broken on one side and completed by its antipode on the other. The size of the defect is the antichain width; the antipodal symmetry is what keeps it small.

The width is a product

The width of a level — the number of distinct cofactors, the reduced-diagram node count — is at most the height of the broken lattice times its antichain width. This is Mirsky's product bound[1] (proved in Chapter 6.1's companion), read on the cofactor poset: rank the cofactors by longest descending chain; equal ranks are incomparable, so each rank class is an antichain bounded by the width; and there are at most *height*-many ranks.

Theorem 16.1 (The broken-lattice width bound)**.** *If every cofactor at a level has Mirsky rank below H (so every chain has length at most H) and every antichain has size at most W, then the level width is at most $H \cdot W$.*

So the width is bounded by two dimensions of the broken lattice: its *height*, the length of its longest chain, and its *antichain width*, the size of its defect. The bound rests on no assumed quantity — it is stated in the poset's own two dimensions, and it is exactly the Mirsky product bound applied to the cofactors.

[1] [Mirsky 1971] — reused here on the cofactor poset of the broken lattice: its size is at most the chain height times the antichain width.

Why this closes the term

The verification argument had left the width W as an unfolded symbol: the one place the proof of polynomial-time verification could be doubted. This is where it closes. The width is $H \cdot W$; the height H is bounded because cofactors along a chain strictly increase in a bounded interval; and the antichain width W — the defect of the broken lattice — is small because the lattice is half of a self-dual antipodal pair. When both dimensions are polynomial in n, the width is polynomial, and with it the verifier's cost. The term that was used to discredit the argument is reduced — plainly — to a product of two quantities the broken-lattice structure controls: it is no longer an opaque symbol but the height times the defect width, and the remaining task is the bounding of those two, which the antipodal structure makes approachable. The width is not asserted polynomial; it is *expressed* as the product that the structure explains, with each factor named.

The picture is complete: the automaton width read off the distributive interval and its small antipodal defect. The object the book was named for — the broken lattice — is exactly the object that closes the bound.

The machine-checked theorem

```
theorem broken_lattice_width_bound:
  fixes f :: "bool list  bool"
  assumes "finite (cof_level_set f k)"
      and "∀g ∈ cof_level_set f k.
             mrank (cof_level_set f k) (cof_lt m) g < H"
      and "∀S. is_antichain (cof_level_set f k) (cof_lt m) S
                  ⟶ card S ≤ W"
  shows "level_width f k ≤ H * W"
```

The proof applies `dilworth_product_bound` — the Mirsky product bound, with the rank constructed internally — to the cofactor level-set under the strict pointwise order `cof_lt`, and rewrites the cardinality as the level width. No free width symbol, no hole: the bound is the poset's height times its antichain width, derived. *Session:* `Broken_Lattice_Width.thy`.

In the broken lattice

For the final wires the width of the broken lattice is closed. There the lattice is a chain, the antichain width is one, and the bound is a theorem. The closure holds exactly where the lattice is unbroken; the broken interior remains.

Chapter 17

The Intermediate Frontier

The width closed for the final wires; this chapter reaches into the wires the verifier builds along the way, and reports plainly how far the reasoning carries and where it stops. The intermediate wires are monotone but not thresholds, so the threshold count — the exact width of the final chapters — does not apply to them. What is provable is the FRAME: the width of an intermediate wire is the size of a monotone image, and its defect pulls back to an antichain of a cube. What is not yet proved is the collapse of that antichain, which for non-threshold wires must be established anew.

The width is a monotone image

Fix a level: a wire's cofactor there is obtained by fixing a prefix of the inputs. The map from a prefix to its cofactor is MONOTONE — fixing more inputs to true can only raise the cofactor, the Shannon ordering lifted to whole prefixes. So the set of distinct cofactors is the image of a monotone map on the cube of prefixes, and its size — the width — is at most the number of prefixes.

Theorem 17.1 (The width is a monotone image)**.** *The cofactor map at a level is monotone, and the cofactor image is finite with cardinality at most the number of length-k prefixes.*

The defect pulls back to the cube

The broken lattice's defect at a level is an antichain of cofactors — incomparable pairs. Because the cofactor map is monotone, incomparable cofactors can only come from incomparable prefixes: a comparable pair of prefixes maps to a comparable pair of cofactors. So an antichain of cofactors pulls back to an antichain of prefixes, and the defect width is at most the antichain width of the prefix cube.

Theorem 17.2 (The defect pulls back)**.** *If a set of cofactors is a value-antichain, the prefixes producing them form an antichain in the pointwise prefix order. Hence the antichain width of the cofactor poset is at most that of the prefix cube.*

Where the reasoning stops

From monotonicity alone, this is as far as it goes — and the ceiling it gives is the antichain width of the cube itself, which by Sperner's theorem[1] is exponential. Monotonicity bounds the defect by

[1] [Sperner 1928] — the largest antichain of the Boolean lattice on k elements has $\binom{k}{\lfloor k/2 \rfloor}$ members.

the widest antichain of the cube, and no further.

To bound the defect polynomially one must show that the cofactor map COLLAPSES that antichain — that the specific cofactors of an intermediate wire, though not thresholds, still have few incomparable values. For the final wires this collapse is the threshold count of Chapter 13.1; but the intermediate wires are not thresholds, and *the threshold count-collapse does not apply to them.* The collapse must be re-established from the intermediate wires' own structure — the stacking of the single-variable sensitive bands (Chapter 5.1's companion) into the level's defect. That is the last pitch, and it is not climbed here. We state it as the open frontier, exactly located: *the antichain of the monotone cofactor image, for the non-threshold intermediate wires, bounded polynomially.* With it, and the height bound, the broken-lattice width bound of the last chapter would close for every wire, and the verifier's polynomial time would follow the whole way.

The machine-checked theorems

```
theorem width_finite_and_bounded:
  "finite (cof_image phi k)
     ∧ card (cof_image phi k) ≤ card {p::bool list. length p = k}"

theorem cofactor_antichain_pulls_back:
  assumes "cof_monotone_map phi vle k"
      and "∀p ∈ Q. length p = k"
      and "∀p ∈ Q. ∀q ∈ Q. p ≠ q
             ⟶ (phi p, phi q)  vle ∧ (phi q, phi p)  vle"
  shows "prefix_antichain Q"
```

The width is the size of a monotone image (@thm width_finite_and_bounded); the defect pulls back to a prefix antichain of threshold wires, is the named and precisely located frontier. *Session:* `Band_Stacking.thy`.

In the broken lattice

The intermediate frontier is the broken interior of the lattice. Away from the final wires the cofactors carry a genuine antichain — the defect W_0 — and the lattice is truly broken. This chapter names the one open quantity: the width of the lattice where it is not a chain.

Chapter 18

The Approach to the Whole

The intermediate frontier named one open quantity: the antichain width of the broken lattice, the defect W_0. This chapter reports the approach that funnels the entire width bound onto that single quantity, and characterises it, so that what remains is sharp and small. Three moves compose: the width reduces to a product; the distributive structure supplies one factor for free; and a bridge from the literature converts the other into the OBDD width. At the end, one lemma stands, characterised from three sides.

Distributivity, for free

The cofactors at a level, ordered pointwise, meet and join by pointwise conjunction and disjunction. Distributivity of those operations is inherited from the two-element Boolean algebra — conjunction distributes over disjunction on $\{\bot, \top\}$, and so pointwise everywhere. So the function lattice is distributive, the monotone functions are a distributive sublattice, and any meet/join-closed set of cofactors is a distributive lattice. Proving the broken lattice "distributive" costs one inherited identity; it is not the frontier.

Theorem 18.1 (Distributivity is inherited)**.** *Pointwise meet and join distribute:* $f \sqcap (g \sqcup h) = (f \sqcap g) \sqcup (f \sqcap h)$*. The monotone functions are closed under meet and join, and any meet/join-closed set of functions is a distributive lattice.*

The defect is the closure

The cofactor set is not itself meet/join-closed — the meet of two cofactors need not be a cofactor. That non-closure *is* the brokenness.[1] Taking the meet/join closure fills in the missing points and un-breaks the broken lattice into a genuine distributive lattice; the points added are exactly the defect. So the defect W_0 — the antichain width of the cofactor poset — and the closure, the un-breaking, are two views of one quantity: how far the cofactors are from a lattice.

[1] [Knuth 1998] Vol. 3, §5.3.4, exercise 33, p. 240 — not every monotone Boolean function arises as an output of a comparator network; e.g. $(x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_4)$ cannot. The non-surjectivity Knuth records is the same brokenness studied here.

The bridge: distributivity to width

A distributive lattice has a linear skeleton. By Birkhoff's representation,[2] it is the lattice of down-sets of its poset of join-irreducibles, and a linear extension of those join-irreducibles gives a path decomposition whose bags are the antichains crossed — so its pathwidth is the antichain width of the join-irreducibles. And a result on decision diagrams — bounded circuit pathwidth yields bounded OBDD width[3] — converts that pathwidth into the diagram width. The chain is

$$\text{broken lattice} = \text{distributive} - W_0 \;\longrightarrow\; \text{pathwidth} = O(W_0) \;\longrightarrow\; \text{OBDD width} \leq 2^{O(W_0)}.$$

Crucially the bridge is symmetry-free: it governs the width by the pathwidth, not by any threshold or symmetry, so it applies to the intermediate wires, which are not thresholds. Where the final chapters needed the threshold count, here the distributive structure and the defect suffice.

The defect, characterised

What remains is to bound W_0. It is characterised from three sides, each tying it to a structure the book has already met.

First, $W_0 = 1$ exactly when the poset is a chain — the threshold, final-wire case. So the defect measures the departure from a threshold: the final wires are the unbroken ones.

Theorem 18.2 (The defect vanishes on chains)**.** *The antichain width is one exactly when the cofactor poset is a chain.*

Second, the antipode fixes W_0. The Boolean dual is an order-reversing involution; it carries an antichain to an antichain of the same size, so a wire and its antipode — wire i and wire $n-1-i$ — have equal defect. The brokenness is antipode-symmetric, like the median and the whole lattice; and so it suffices to bound W_0 up to the median, the antipode giving the rest.

Theorem 18.3 (The antipode fixes the defect)**.** *An order-reversing involution maps an antichain to an antichain of equal cardinality; hence the antichain widths of a poset and of its antipodal image agree.*

Third, the concrete face: each cofactor is a monotone band between a least true weight and a greatest false weight, and two cofactors are incomparable exactly when their bands cross. So W_0 is the size of the widest family of mutually crossing bands — the combinatorial object in which a constant or logarithmic bound would fire the bridge.

Where the approach stands

Assembled: the width is at most the height times the defect (Theorem 16.1); the distributive structure supplies the height's role through Birkhoff, for free (Theorem 18.1); and the bridge converts the defect into the diagram width, without symmetry. One lemma remains, and it is a CONDITIONAL that the rest of the argument discharges only if its antecedent is supplied: *IF the defect W_0 is polynomial for intermediate wires — a constant, or a logarithm — THEN the width*

[2] [Birkhoff 1937] — every finite distributive lattice is the lattice of down-sets (order ideals) of its poset of join-irreducible elements.

[3] [Jha & Suciu 2012] — a Boolean circuit of bounded pathwidth compiles to an OBDD of bounded width, and conversely; the OBDD width and the circuit pathwidth are mutually bounded.

is polynomial for every wire and the verifier runs in polynomial time. What the approach delivers is that antecedent's reduction, not its proof: the defect is characterised three ways — it vanishes on chains, is fixed by the antipode, counts crossing bands — and localised (next chapter) to a single weight, but no step of the approach bounds it by a number. The frontier is therefore one quantity, seen from three sides, with the proven apparatus and two classical theorems poised to fire the moment it is bounded — and it is not yet bounded.

The machine-checked theorems

```
theorem fmeet_fjoin_distrib:
  "fmeet f (fjoin g h) = fjoin (fmeet f g) (fmeet f h)"

theorem no_two_antichain_implies_chain:
  assumes "∀x∈A. ∀y∈A. is_antichain A r {x,y} ⟶ x = y"
  shows "poset_is_chain A r"

theorem antipode_preserves_antichain_card:
  assumes "order_reversing_involution A r d"
      and "is_antichain A r S" and "finite S"
  shows "card (d ‘ S) = card S"
```

Distributivity is inherited (@thm fmeet$_f join_d istrib$); $the defect vanishes exactly on chains$(@$thm no_t wo_a ntichain_i m$

In the broken lattice

The approach funnels the whole argument onto the width of the broken lattice. Distributivity is free; the pathwidth bridge converts the lattice's structure into the diagram's width; and the defect W_0 — the antichain of the broken lattice — is the one quantity that remains to bound.

Chapter 19

The Defect Localised

The approach funnelled the width to one quantity, the defect W_0; this chapter advances it, cutting the defect down by a classical theorem, and reports plainly how far that carries. The move is Helly's, in one dimension: the mutually incomparable cofactors that make up the defect turn out to share a common weight, so the two-dimensional antichain collapses to a one-dimensional slice.

Bands

A monotone predicate on the residual cube has a BAND. Below some least weight it is constantly false; above some greatest weight it is constantly true; between, it is mixed — its value turns on which bits are set, not merely how many. A threshold has no band at all: it flips at a single weight, false below and true above. An intermediate wire's cofactor may have a genuine band, and the band's width is the room in which it departs from a threshold.

Incomparable cofactors overlap in weight

If two cofactors' bands are disjoint — one lying entirely below the other in weight — then the cofactors are comparable: where one is still mixed, the other is already settled, and settlement dominates. Contrapositively, incomparable cofactors must have OVERLAPPING bands. So the defect — an antichain of pairwise-incomparable cofactors — is a family of pairwise-overlapping bands.

Theorem 19.1 (Disjoint bands are ordered)**.** *If two bands do not overlap, one lies wholly below the other in weight.*

Helly collapses the antichain

Intervals on a line have the Helly property: if they pairwise overlap, they share a common point. Applied to the bands of the defect, pairwise overlap yields a single weight w lying in every band. So all the mutually incomparable cofactors of the defect are mixed at one and the same weight, and the defect is at most the number of cofactors whose band contains w.

Theorem 19.2 (Helly, one dimension)**.** *A finite, nonempty family of natural-number intervals that pairwise overlap has a common point; the maximum of their left endpoints witnesses it.*

Theorem 19.3 (The defect localises)**.** *The bands of an antichain of cofactors share a common weight w. Hence the defect W_0 is at most the number of cofactors mixed at the single weight w.*

Where this carries, and where it stops

The two-dimensional defect — an antichain in the broken lattice — is thus reduced to a one-dimensional slice: the cofactors that genuinely depend on which bits are set, not only how many, at a single weight level. This is a sharper and more concrete target than the antichain itself. It does not, on its own, bound the defect: the slice could in principle be large, and bounding it is the residual question — the number of distinct partial orderings the network has resolved at one weight, which is the antichain of the reachable ordering state, the object the whole study began with. That count is not settled here; it is the frontier's last step, now standing over a single weight rather than the whole cube.

The reduction is the content: from the width of a diagram, to the height times a defect, to the defect localised at one weight by Helly. Each step is a theorem, and the last unsettled quantity is now as small and as concrete as the reasoning has been able to make it — one weight, one slice, the cofactors mixed there.

A ring of reductions, and the one door still shut

It is worth being exact about what the chain of chapters has established, and what it has not. Each link is of the form "this quantity is at most that one": the diagram width is at most the height times the defect; the pathwidth is the antichain width of the join-irreducibles; the width is at most an exponential of the pathwidth; the defect is at most the count in one weight-slice. Follow them far enough and they close into a ring — the slice is the antichain of the reachable ordering state, the object of the first chapter. Every arrow in the ring is a theorem or a cited classical result.

But a ring of the form "$A \le B \le C \le A$" proves that A, B, and C rise and fall together; it does not prove that any of them is small. Not one link ends in a number. The reduction is complete and the quantities are shown to be one and the same, localised and characterised down to a single weight — and still the polynomial bound does not follow, because nothing yet touches a magnitude. What is proved is a conditional: *if* the reachable-state antichain is polynomial, *then* the width is polynomial for every wire, and the verifier runs in polynomial time. The antecedent — that one count, in one slice, over the reachable ordering state — is the single door that remains shut. The circle does not open it; it places it, exactly, under one hand. The book stops here, with the reduction whole and the count named, and leaves the opening of that last door for the return.

The machine-checked theorems

```
theorem disjoint_bands_separated:
  assumes "¬ band_overlap lo hi i j"
      and "lo i ≤ hi i" and "lo j ≤ hi j"
  shows "hi i < lo j ∨ hi j < lo i"

theorem helly_1d_intervals:
  assumes "finite I" and "I ≠ {}"
      and "∀i∈I. lo i ≤ hi i"
      and "∀i∈I. ∀j∈I. {lo i..hi i}  {lo j..hi j} ≠ {}"
  shows "∃w. ∀i∈I. w ∈ {lo i..hi i}"

theorem antichain_bands_share_weight:
  assumes "finite I" and "I ≠ {}"
```

```
    and "∀i∈I. lo i ≤ hi i"
    and "∀i∈I. ∀j∈I. band_overlap lo hi i j"
shows "∃w. ∀i∈I. lo i ≤ w ∧ w ≤ hi i"
```

Disjoint bands are ordered (@thm disjoint$_b$*ands*$_s$*eparated*); *pairwise*−*overlappingintervalsshareapoint*(@*thmhell*
to one slice. *Session:* `Band_Helly.thy`.

In the broken lattice

The defect of the broken lattice is here localised to a single weight. By Helly the antichain of incomparable cofactors shares a common weight-slice, so the brokenness — W_0 — lives on one level of the cube. The lattice is broken most sharply at one weight, and that is where the width concentrates.

Chapter 20

The Two Faces

The defect was localised to a single count; this chapter turns that count around to face the tool built to compute it. For the network read forward and the network read backward are not two problems but two faces of one object — one drawn from the theory of Coxeter groups — and the door that the forward face leaves shut is the door the backward face was made to open.

The comparator is a Demazure operator

A comparator sorts a pair: it sends (a, b) to $(\min, \max)$, which on the Booleans is $(a \wedge b,\ a \vee b)$. Two properties fix its character. It is IDEMPOTENT — sorting a sorted pair changes nothing — and it acts as the identity on already-ordered pairs, moving only inversions. These are the defining properties of a Demazure, or 0-Hecke, operator π_i. So a comparator network is, read forward, a DEMAZURE PRODUCT in the 0-Hecke monoid of the symmetric group; the sorted output is the longest element w_0; and the partial-sort states it passes through are a Demazure orbit.

Theorem 20.1 (The comparator is idempotent)**.** *For $a \neq b$ in range,* $\mathrm{cstep}\, a\, b\, (\mathrm{cstep}\, a\, b\, v) = \mathrm{cstep}\, a\, b\, v$*; and a comparator fixes any vector whose pair is already ordered.*

The backward reading is the Bruhat order

Read backward, the same network is a SHUFFLER: a word of gated transpositions, each either applied or skipped. The permutations it can reach form a lower set of the BRUHAT ORDER, and it reaches all of S_n exactly when a reduced word for w_0 appears as an ordered subword — which is Tits' subword property,[1] the defining combinatorics of Bruhat order. The trace of a sorting network, read backward, is moreover a complete prefix code: a bijection onto S_n, one canonical subword per permutation, a reduced-word normal form. Its essential depth is the Coxeter LENGTH — the inversion count — and the length saturates at w_0.

Theorem 20.2 (Length saturates at w_0)**.** *Every permutation of $\{0, \ldots, n-1\}$ has at most $\binom{n}{2}$ inversions; the reversing permutation $w_0(i) = n - 1 - i$ inverts every pair and attains $\ell(w_0) = \binom{n}{2}$.*

So the two readings are two monoid structures on one Coxeter group: forward, the Demazure product (sort-if-needed, idempotent); backward, the Bruhat subword-product (always-swap). The factoradic saturation of the backward shuffler is the length function reaching its maximum; the forward width is the diagram of the Demazure orbit.

[1] [Björner & Brenti 2005] — in the Bruhat order, $u \leq w$ if and only if some reduced word for w contains a reduced word for u as a subword. *Combinatorics of Coxeter Groups*, Springer GTM 231, §2.2.

The door, turned around

The forward defect W_0 is the antichain of the Demazure-reachable states. In the Bruhat order — where, in general, antichains can be far larger than chains — an antichain is a daunting thing to bound directly. But an antichain is a SUBSET, and so W_0 is at most the SIZE of the reachable set; and the reachable set, ranked by Coxeter length, is the sum of its length-levels. So the forward defect is bounded by a backward count.

Theorem 20.3 (The defect is bounded by the reachable count)**.** *The defect W_0 (any antichain of the reachable set) is at most the number of reachable states, which equals the sum over lengths L of the number of reachable states of length L.*

This is the turn. Bounding the forward antichain becomes counting the backward orbit: it suffices that the Demazure-reachable set be polynomially sized — polynomially many lengths, each a polynomial level. And the reachable count is exactly what the backward shuffler computes. The instrument built for one face bears on the door of the other. What was "bound a Bruhat antichain, typically enormous" becomes "count a specific, highly structured reachable orbit" — the orbit whose backward trace is a complete prefix code. The daunting framing gives way to a tractable one, and the same door remains, now facing the key.

The neighbourhood

Placing the object among its neighbours measures the prize. The reachable count is a question about a Demazure orbit — named in the theory of affine Demazure products; the defect is a Bruhat antichain — studied by Brualdi and Deaett,[2] where such antichains are known to reach k^8 against chains of k^4; and the cofactor lattice is a sublattice of the free distributive lattice, whose elements the Dedekind numbers count,[3] growing doubly exponentially. In every direction the ambient space is large. So the smallness of our defect, if it holds, is not generic; it is a structural anomaly of the sorting-network orbit. To bound W_0 is to prove that this particular Demazure orbit carries an atypically small Bruhat antichain — a statement of content in Coxeter theory, and not a mere accounting of a data structure.

The machine-checked theorems

```
theorem cstep_idempotent:
  assumes "a < length v" and "b < length v" and "a ≠ b"
  shows "cstep a b (cstep a b v) = cstep a b v"

theorem w0_length_maximal:
  "coxeter_length (w0 n) n = n * (n - 1) div 2"

theorem defect_le_level_sum:
  assumes "finite A" and "∀x∈A. len x ≤ N"
      and "is_antichain A r S"
  shows "card S ≤ (∑L≤N. card (level A len L))"
```

[2] [Brualdi & Deaett 2007] — on the largest antichain in the Bruhat order of a class of $(0,1)$- matrices; in $A(2k,k)$ the largest antichain is at least k^8 while the largest chain is k^4.

[3] [Dedekind 1897] — the number of monotone Boolean functions of n variables (equivalently, antichains of the n-cube); these numbers grow doubly exponentially in n.

The comparator is idempotent (@thm cstep$_i dempotent$), $sotheforwardnetworkisaDemazureproduct; theCoxeterl$
(@thm w0$_l ength_m aximal$), $thebackwardfactoradicreading; andtheforwarddefectisboundedbythebackwardreacha$

In the broken lattice

The two faces are two readings of the broken lattice. Forward it is the cofactor lattice with its defect; backward it is the reachable orbit whose antichain is the same defect. The bridge shows the two are one broken lattice, its width bounded by the reachable count.

Chapter 21

The Block Confinement

The intermediate frontier reduced the open question to the support of an intermediate wire: how many inputs a wire depends on after only some of the comparators. Monotonicity alone bounded that support by n — the whole width of the network — which is no bound at all for the diagram. A structural result on the back end of a sorting network sharpens it dramatically. The channels partition, after each layer, into consecutive *blocks* between which values never move; and a wire's value after that layer depends only on its own block, whose size is bounded by the comparators within it. The support of an intermediate wire is confined to its block.

Blocks

For a sorting network of depth d and a layer $k < d$, the k-blocks partition the channels into consecutive runs.[1] The governing fact is about the semantics: for each 0/1 input, at most one block is *mixed* — receives both zeros and ones — while every block above it is all zeros and every block below is all ones, and no value ever crosses a block boundary.[2] This is, in the language of this book, the unique mixed block among otherwise settled blocks: the order-ideal-with-unique-bottom shape (Chapter 20), read off the network's own structure.

Blocks are short

Blocks are not merely consecutive; they are small. A k-block containing m comparators spans at most $m + 1$ channels.[3,4] So the size of the block through which a wire's value is determined is bounded by the number of comparators acting within it — a genuine, structural bound, not the trivial n.

[1] 1. [Codish 2017] p. 9 — the k-blocks of a comparator network form a partition of its channels into consecutive runs, defined by working backward from the last layer.

[2] 2. [Codish 2017] p. 10, Lemma 5 — for every 0/1 input, at most one k-block is mixed, those above receiving only zeros and those below only ones.

[3] 3. [Codish 2017] p. 11, Corollary 3 — a k-block containing m comparators consists of at most $m + 1$ channels. The same block results appear earlier in [Codish 2014], the "End Game" precursor (there Corollary 12).

[4] 4. The block theory is stated for *non-redundant* networks, those with no comparator that never swaps; the notion is Graham's, recorded as [Knuth 1998] Vol. 3, §5.3.4, exercise 51, p. 242. Removing redundant comparators does not change the function computed.

The support is confined to the block

Because no value crosses a block boundary, a wire's value after layer k is a function of the inputs feeding its own block alone. Everything outside the block is irrelevant to it. We prove the abstract consequence: a function whose support lies within a set B is determined by the B-coordinates, and ignores every coordinate outside B.

Theorem 21.1 (Determined by the block)**.** *If a wire function has support within a block B, then any two inputs agreeing on B give the same value; and flipping any coordinate outside B never changes the wire.*

So the intermediate wire is a function of its block's channels only, and its diagram is governed by the *block size*, not by n. With the block-size bound of at most $m + 1$ channels, the intermediate width over n channels is reduced to the width over a short consecutive block.

Where this carries, and the caution

This is a real handle, and it must be stated with its limit. The block confinement is a property of the 0/1 semantics of a *sorting* network; it bounds the SUPPORT of an intermediate wire to its block. It does not, alone, bound the width: the wire's dependence *within* its block is still the intermediate question, and blocks grow with depth as more comparators accumulate. What the result delivers is a reduction — from "the intermediate width over n channels" to "the intermediate width over a block of at most $m + 1$ channels" — localising the frontier to a short run of the network rather than its full width. The block is the natural home of the intermediate defect, and its size is the next quantity to bound; the front-back duality that produced these results[5] is the same antipode that folds this book's broken lattice onto itself.

The machine-checked theorem

```
theorem determined_by_support:
  assumes "support_within f n B"
      and "length x = n" and "length y = n"
      and "∀i<n. i ∈ B ⟶ x!i = y!i"
  shows "f x = f y"

theorem outside_support_irrelevant:
  assumes "support_within f n B"
      and "length x = n" and "j < n" and "j  B"
  shows "f x = f (x[j := b])"
```

A wire with support within its block is determined by the block's coordinates (@thm determined$_b y_s upport)andignor

In the broken lattice

Block confinement shrinks the ambient of the broken lattice. An intermediate wire's cofactors depend only on its block, so the broken lattice of that wire lives on a short run of channels, not the whole width. The breakage is confined to the block.

[5] 5. [Codish 2017] p. 1 — the paper establishes a duality between the symmetries of the first layers and those of the last layers of a sorting network.

Chapter 22

Two Optimal Networks Diverge at the Middle

The theory of this book places the defect — the antichain W_0, the median region, the antipode's fixed point — at the centre of the object. It is worth pausing on a piece of evidence, structural rather than proven, that the middle really is the matter: the two best-known sorting networks on sixteen elements, built by hand around 1970, agree everywhere except at the middle, and diverge exactly there.

The hypercube phase

An efficient sorter on $n = 2^k$ elements begins by ordering its inputs into a *Boolean cube poset*[1] — a partial order that is the cube $\{0,1\}^k$ under coordinatewise comparison, its elements stratified into weight-layers. For sixteen elements this approximate phase has depth four and thirty-two comparators. Read as an operation on wire indices, each layer of this phase is an exclusive-or by one basis vector of $\mathbb{F}_2^k$: a stride-2^i layer compares the wires whose indices differ in bit i, which is exactly translation by e_i. So the hypercube phase is a walk through the standard basis, one direction per layer — the sorter building the cube by laying down its axes in turn.

This is the same structure the antipode chapter met from the other side: the antipode on 2^k wires is the exclusive-or by the all-ones mask, and the all-ones mask is the sum of every basis vector (Chapter 5.1's companion). The sorter walks the basis; the antipode is the total of the walk.

The extremes settle first

Once the cube is built, its extremes are already decided.[2] The maximum and the minimum are fixed; then the second and third from each end; and what remains undetermined is an eight-element *middle set* — the region around the median. The construction has settled the antipodal pairs from the outside in, and left the centre for last. This is the antipode-final-move mechanism seen in a concrete network: the extremes fall to the basis walk, and the median region is the final completion, the piece reserved to the end.

[1] [Sergeev 2018] §2 — the initial "approximate sorting" subnetwork of an efficient 16-element sorter orders its inputs into a Boolean cube poset, its layers the weight-levels of the cube. arXiv:1810.11262.

[2] [Sergeev 2018] §2, observations (a)–(d) — after the approximate phase the largest and smallest elements are determined, then the second and third of each end, leaving an eight-element middle set to be sorted.

The divergence

Here is the point. The two classic sixteen-element networks — Green's, of sixty comparators and depth ten, and van Voorhis's, of sixty-one comparators and depth nine[3] — share the *identical* hypercube phase and the identical settling of the extremes. They differ in one thing only: how they sort the middle set.[4] Green sorts the middle as two tetrads and merges, spending fewer comparators; van Voorhis performs extra preliminary comparisons on the middle to reach a shallower depth. The complexity-versus-depth trade-off of the two optimal networks lives *entirely* in the completion of the middle set.

So the one place where the two best hand-built sorters make different choices — the one place the tension between fewer comparators and fewer layers is resolved — is the median region: the middle set, the defect's home, the antipode's fixed point. Two designers, working the same object by hand, agreed on the whole cube-building periphery and parted ways only at the centre.

What this is, and is not

This is not a proof. It is two networks at a single arity; nothing here bounds the defect or closes the width. And the two are not the whole story: the catalogue of optimal sixteen-element networks holds dozens more, in several structural families, treated in the next chapter. But the evidence of these two steadies the theory all the same: the object the theory places at its centre — the middle, the defect, W_0 — is exactly the object at which Green's and van Voorhis's networks diverge. Where the theory says the difficulty lives, the difficulty visibly lived for the two designers who met it first. The periphery is a hypercube walk, settled by the basis; the centre is the matter.

Three accounts converge on the same shape: the exclusive-or basis walk measured on an optimal sixteen-network; Sergeev's reading of the approximate phase as a Boolean cube poset; and this book's antipode-final-move mechanism, the extremes settling by the basis and the median completing last. None proves the width polynomial. All three name the same centre: the middle is where the work is.

In the broken lattice

The two optimal networks diverge exactly where the broken lattice is thickest. Green and van Voorhis agree on the periphery and part at the middle set — the median region, the antipode's fixed point, where the defect W_0 lives. The place they differ is the place the lattice is most broken.

[3] [Sergeev 2018] §1 — the Green network has complexity 60 and depth 10; the van Voorhis network has complexity 61 and depth 9, the depth shown optimal by [Bundala et al. 2017].

[4] [Sergeev 2018] §3–4 — Green and van Voorhis networks "differ only in the way they sort the set M"; Green sorts two tetrads and merges them (lower complexity), van Voorhis performs extra preliminary comparisons to reach lower depth.

Chapter 23

Where the Antipode Sits

The two named networks are two points in a wider space. A catalogue of optimal sixteen-element networks gathered for this work holds dozens — twenty-eight at Green's sixty comparators and depth ten, twenty at van Voorhis's sixty-one and depth nine, and further networks at other size-and-depth points, ninety-two in all across the census. Surveying them, an organising principle appears, and it is the antipode: the networks sort into families by *where the antipode fold sits*.

The antipode fold

On $n = 2^k$ wires the antipode — the reflection $i \mapsto n - 1 - i$ — is the exclusive-or by the all-ones mask (Chapter 5.1). A layer of comparators whose every pair exclusive-ors to the all-ones mask is an *antipode fold*: it pairs each wire with its mirror, folding the whole cube along its long diagonal. At the other extreme, a layer whose pairs exclusive-or to a single-bit mask folds along one short edge — the finest, adjacent fold. Reading a network by the exclusive-or signatures of its layers shows where, if anywhere, the antipode fold falls.

Three placements

The catalogue's sixty- and sixty-one-comparator networks fall into three families, distinguished by the position of the antipode fold, and the division is clean.

The antipode in front. A sub-family opens *fold-first*: the first layer is the antipode fold, the whole-cube diagonal, and the last layer is the single-edge fold. These open with the long antipode and close with the short one, two perpendicular folds bracketing the network, with the rearrangement of the half-cubes as the work between (twelve networks, sixty comparators, depth ten).

The antipode in back. The depth-optimal family — van Voorhis's parameters, sixty-one comparators and depth nine — opens with the standard adjacent layer and carries the antipode fold into its *closing* layer, where the all-ones mask appears in the final settling of the median. Every network of this family closes with the antipode present (twenty networks).

The antipode in the middle. The remaining sixty-comparator family opens and closes with the adjacent fold, and carries the antipode fold in its *interior* layers — every one of them contains the all-ones mask somewhere in the middle of the network, never at an end (sixteen networks).

So the antipode fold is present in every family; what distinguishes them is its position — front, back, or middle. The reflection that runs through this book as a symmetry of the object appears here as a structural coordinate of the constructions: the place a designer puts the whole-cube fold is the place that names the family.

The reading

This is a survey, not a theorem, and only of the networks gathered here. But it says something about the antipode's role. In the theory the antipode is one reflection with several faces — Boolean duality, the wire mirror, the rank complement, the defect's symmetry, the Coxeter length complement. In the constructions it is a fold that can be placed anywhere along the network, and the placement is a design choice: fold the cube first and rearrange, or sort toward the fold and close on it, or turn about it in the middle. Each optimal network chooses a moment to fold the cube onto itself, and the moment it chooses sorts it into a family. The antipode is not only the symmetry of the object; it is the axis every construction turns about, and the survey shows the turning happening at three different times.

In the broken lattice

The antipode is the fold of the broken lattice, and the families sort by where that fold sits. Front, middle, or back, every optimal network folds the lattice onto itself at some moment; the moment it chooses is a coordinate of the broken lattice's symmetry, made a design choice.

Chapter 24

The Cost

Every beam is now in place. The verifier's correctness follows from the monotone, zero-one, and adjacent-settled lemmas. The width it pays for is the cofactor count (Chapter 11.1's companion), which for a sorter is a chain of at most $n + 2$ thresholds (Chapter 12.1). This chapter assembles the machine and states its running cost.

The cost model

The verifier processes the comparators one at a time. For each it performs a bounded amount of diagram work — two Boolean operations on the wire functions and one settledness check, each costing at most a constant times the current diagram width.[1] We model the total as a fold over the comparator list.

Theorem 24.1 (The cost bound)**.** *If every step's diagram width is at most W, the total cost of processing m comparators is at most $m \cdot c \cdot W$, where c is the constant number of operations per step.*

Theorem 24.2 (Polynomial cost)**.** *If the peak width is at most $k \cdot n$ and the number of comparators is at most $n \cdot d$ at depth d, then the total cost is at most $c \cdot k \cdot (n^2 \cdot d)$ — polynomial in n.*

The assembly, in prose

The cost is a sum of per-step costs. Each step costs at most $c \cdot W$, so the sum over m steps is at most $m \cdot c \cdot W$ (Theorem 24.1): the cost is $\mathcal{O}(m \cdot W)$. For a sorting network the width W is at most $n + 2$ (Chapter 12.1) and the comparators number at most $n \cdot d$, so the total cost is at most $c \cdot k \cdot n^2 \cdot d$ (Theorem 24.2): polynomial in n. The quantity W, a symbol through the middle of the book, is now a bounded factor in a machine-checked polynomial.

The machine-checked theorems

```
fun run_cost :: "nat  (nat  nat)  nat  nat" where
  "run_cost c width 0 = 0" |
```

[1] [Drechsler 2021] — if every internal signal's decision diagram is of polynomial size, the whole symbolic construction is polynomial, since the apply (ITE) operation is bounded by the product of its operand sizes; the per-signal-width reduction the cost fold here instantiates. *Polynomial Circuit Verification using BDDs*, arXiv:2104.03024.

```
  "run_cost c width (Suc m) = step_cost c (width m) + run_cost c width m"

corollary cost_is_O_m_W:
  assumes "∀i < m. width i ≤ W"
  shows "run_cost c width m ≤ m * c * W"

theorem cost_polynomial:
  assumes "∀i < m. width i ≤ k * n" and "m ≤ n * d"
  shows "run_cost c width m ≤ c * k * (n * n * d)"
```

The cost is a fold over the actual comparators; the bounds are by induction on the fold. `cost_is_O_m_W` is the $\mathcal{O}(m \cdot W)$ statement; `cost_polynomial` assembles it with the width and comparator bounds into a polynomial. Nothing enumerates the cube. *Session:* `Verifier_Cost.thy`.

Discharging the width

The cost theorem above takes a width bound as a hypothesis. For the FINAL wires of a sorting network that bound is discharged, not assumed: those wires are thresholds, so their cofactor sets are chains of at most $n+2$ thresholds (Chapters 9.1 and 12.1), and their width is at most $n+2$. The INTERMEDIATE wires — the diagrams the verifier builds after only some of the comparators — are monotone but not thresholds, and for them the width bound is not inherited from the threshold structure; it remains conditional (see Chapter 17). So the cost theorem below is a genuine theorem for the final-wire width, and, for the per-step width it needs at every stage, a stated conditional on the intermediate-wire width.

Theorem 24.3 (The verifier's cost on a sorter)**.** *If every step's width is at most $n + 2$, the cost of processing m comparators is at most $m \cdot c \cdot (n + 2)$; and with $m \leq n \cdot d$ the total is at most $c \cdot (n + 2)(n \cdot d)$ — polynomial in n.*

```
theorem sorter_verification_cost:
  assumes "∀i < m. width i ≤ n + 2"
  shows "run_cost c width m ≤ m * (c * (n + 2))"

theorem sorter_cost_polynomial:
  assumes "∀i < m. width i ≤ n + 2" and "m ≤ n * d"
  shows "run_cost c width m ≤ c * ((n + 2) * (n * d))"
```

The width bound of the threshold chapters supplies the hypothesis; the verifier's cost on a sorting network is polynomial, welded end to end. *Session:* `Cost_Width_Link.thy`.

In the broken lattice

The cost is the price of reading the broken lattice. The verifier pays, per comparator, in proportion to the width of the lattice at that step; the total cost is the fold of that width, and it is polynomial exactly when the lattice's width is. The cost is the width of the brokenness, summed.

Chapter 25

The Bow

Everything is now in hand. This short chapter gathers the pieces and states, for the class that a verifier is asked about, the bound the whole book was reaching toward. Nothing here is new; it is the assembly.

The pieces

1. The wire functions of a comparator network are monotone (Theorem 1.1).
2. Verification reduces to 0/1 inputs (Theorem 2.1) and then to the settled adjacent pairs (Theorem 3.1).
3. The number of distinct cofactors at a level — the diagram width there — is at most the height times the antichain width of the cofactor poset (Theorem 6.1 and the product bound).
4. For a sorting network, each output wire is a threshold (Theorem 9.1); the cofactors of a threshold are thresholds, and thresholds are totally ordered, so the cofactor set is a chain: antichain width *one*.
5. The threshold parameter normalises to a range of size $n + 2$, so the chain has height at most $n + 2$.

The bound

Combining (4) and (5) through the product bound (3): for a sorting network on n wires, the number of distinct cofactors of any wire function, at any level, is at most

$$H \cdot W \;\le\; (n+2) \cdot 1 \;=\; n+2 \;=\; \mathcal{O}(n).$$

The shared diagram over all n wires has width $\mathcal{O}(n^2)$. A comparator network has $\mathcal{O}(n \cdot d)$ comparators at depth d; verifying it by the order-only method costs $\mathcal{O}(\text{comparators} \cdot W)$, which for a sorter is $\mathcal{O}(n \cdot d \cdot n^2)$ — polynomial in n.

The quantity W, which the verification argument had left as a symbol, is for a sorting network's FINAL wires no longer a symbol. There it is $\mathcal{O}(n)$, and the reason is the threshold structure, proven from the definition of sorting and the arithmetic of counts — not measured, and with no enumeration of the exponentially many inputs. For the intermediate wires the symbol is not yet retired: they are monotone but not thresholds, and their width, though reduced to a named product

of the broken lattice's height and its antichain defect, awaits the bound on that defect. The book closes the width where the wires are thresholds, and names — exactly — the frontier where they are not.

The composition, too, is proven and not merely narrated. The step from sorting to a nondecreasing output, the step from the cofactor count to the product bound, and the step from the width bound to the cost: each is a named theorem (Theorems 9.2, 11.3, 24.3), so the arrows joining the earlier lemmas are themselves machine-checked. What was a chain of proven islands is now a proven chain.

What was, and was not, done

We proved the width of a sorter's FINAL wires: they are thresholds, their cofactor sets are chains, and their diagram width is $\mathcal{O}(n)$. That much is a theorem. But a verifier builds the diagram of every INTERMEDIATE wire too — the wire functions after only some of the comparators — and those are monotone without being thresholds; their cofactor poset carries a genuine antichain, the defect W_0, and its width is the question the whole book has circled. For the intermediate wires the bound is NOT closed. What is proved is a reduction and a conditional: the width for every wire, and with it the verifier's polynomial time, follow *if* that defect is polynomial — a single count, localised to one weight-slice, shown equal to the antichain of the reachable ordering state that the first chapter drew. The ring of reductions places that count under one hand; it does not yet bound it. The one door remains shut.

And the object itself remains, beyond its role here: the cofactor lattice of a sorting network, a bounded distributive interval, folded about its centre by the antipode that exchanges the minimum and the maximum, and — for the sorted wire — a single chain rising from the constant-false cofactor to the constant-true one. That is the bow: a polynomial width proved where the wires are thresholds, a reduction that carries the rest to a single named count, and beneath it all a small, regular, beautiful structure that was there all along — with one door still to open, exactly placed, kept for the return.

In the broken lattice

The bow is tied where the broken lattice is whole and left open where it is not. On the final wires the lattice is a chain and the width is closed; on the intermediate wires the lattice is broken and the width is a named debt. The bow is the plain shape of the broken lattice: closed at its ends, open in its middle.

Chapter 26

The Seams

A last look, not forward to a new result but inward at the web of the ones we have. Several theorems proven in their own chapters stand, on inspection, adjacent: one is an instance of another, or the two compose into a third, and the connection deserves to be a theorem and not merely a remark. This chapter names those seams and shows them welded. Nothing new is claimed; the corpus is tightened from a set of neighbours into a connected whole.

The threshold structure is the monotone structure, specialised

A threshold function is monotone: raising an input can only help it clear the bar. So thresholds live inside the monotone class, and the ordering and comparability results proven for monotone cofactors apply to them directly. The comparability of thresholds — proven in the threshold route — is the monotone ordering read on the threshold family; the threshold chain is a chain of monotone functions.

```
theorem threshold_is_monotone: "mono_on (thr t) n"

theorem threshold_comparability_is_ordering:
  "(∀x. length x = n ⟶ thr t x ⟶ thr t' x)
     ∨ (∀x. length x = n ⟶ thr t' x ⟶ thr t x)"
```

The threshold antipode is the antipode, specialised

Boolean duality is an involution on every function; applied to a threshold it reflects the parameter, sending T_t to T_{n+1-t}. So the threshold antipode of Chapter 10.1, and the self-dual median that followed, are the general antipode specialised to the threshold family — the same involution, seen on thresholds.

```
theorem dualf_involution_general: "dualf (dualf f) = f"

theorem dual_threshold:
  assumes "length x = n" and "1 ≤ t" and "t ≤ n"
  shows "dualf (thr t) x = thr (n + 1 - t) x"
```

The automaton width is the poset bound, applied

The level width — the automaton's state count — and the poset product bound sat in separate chapters, joined only by prose. They now compose: ordering the cofactors pointwise, the level width is at most the height times the Dilworth width.[1] The decision diagram and the broken lattice meet in a single theorem.

```
theorem level_width_le_height_times_width:
  assumes "finite (cof_level_set f k)"
      and "∀g ∈ cof_level_set f k.
             mrank (cof_level_set f k) (cof_lt m) g < H"
      and "∀S. is_antichain (cof_level_set f k) (cof_lt m) S ⟶ card S ≤ W"
  shows "level_width f k ≤ H * W"
```

The width is one, and sortedness is a thermometer

Two final seams close the picture for a sorter. The cofactors of a threshold are comparable, so the cofactor set is a chain — the width factor $W = 1$, stated now in one theorem rather than assembled from parts. And the thermometer characterisation and the entry-wise threshold characterisation of a sorted vector are one fact: a nondecreasing vector's entry i is the threshold of its count.

```
theorem threshold_cofactors_chain:
  "(∀w. length w = m ⟶ thr t (False # w) ⟶ thr t' (True # w))
     ∨ (∀w. length w = m ⟶ thr t' (True # w) ⟶ thr t (False # w))"

theorem thermometer_entry_is_threshold:
  assumes "bsorted v" and "i < length v"
  shows "v!i = thr (length v - i) v"
```

Sessions: `Threshold_Seams.thy`, `Cofactor_Width_Bound.thy`, `Chain_And_Thermometer.thy`. With these the theorems of the book are not a scattering of proven islands but a single connected figure: the threshold results instances of the monotone ones, the antipode of the threshold family an instance of the general antipode, the automaton width and the lattice width one bound, and the width itself, for a sorter, equal to one.

In the broken lattice

The seams are the joints of the broken lattice's argument. Each named theorem is a seam where two parts of the reduction meet; the seams hold because each is proved, and together they carry the width of the broken lattice from sorting to cost.

[1] [Dilworth 1950] — reused here: the antichain width of the cofactor poset equals its least chain-cover, the W of the product bound.

Chapter 27

The Packing

The book has measured the broken lattice, localised its defect, closed its width on the final wires and named the width open on the interior. It closes with what all of that was about — the deep structure the pieces were circling.

The kernel is a volume

To sort is to compute the median threshold. The t-th output of any network that sorts is the same function — at least $n - t$ of the inputs are one — and the median wire, at $t = n/2$, computes the majority, whose diagram width is the ramp $\min(t, n - t) + O(1)$, peaking at $n/2$. This is not a property of any network; it is forced by what sorting means, the same for every sorter, and no sorter escapes it. Call it the *kernel*: the irreducible width at the centre, the volume that must be packed.

The width is a packing, and W_0 is its slack

A sorting network's comparator schedule is a way of packing that kernel. The width at each wire is the kernel ramp on that wire's support plus a residue — the amount by which the wire's cofactor antichain exceeds the threshold ramp. The residue is the *slack* of the packing: the space a schedule leaves loose where it could have been tight. A tangled schedule packs loosely and carries residue; a schedule with structure packs tight.

Batcher's network packs with no slack at all. Its residue is zero: every intermediate wire sits exactly on the threshold ramp, the width tracking the kernel and nothing more. This is not a happy accident of one construction. It is structural necessity. To sort is to compute the median threshold; recursive merge keeps every wire a threshold — the merge of two sorted blocks yields wires that are thresholds on the combined count, by the additivity of the count; and a threshold's width *is* the kernel ramp. A recursive-merge sorter must therefore pack the kernel exactly, its residue zero, because its every wire is a threshold and a threshold is the kernel. Batcher sits on the threshold because the threshold is what recursive packing packs.

The generator is the prover

Here is the resonance the whole study was moving toward. Batcher is easy to *generate*: sort two halves, merge; the recursion writes the schedule. And Batcher is easy to *prove*: two threshold

halves, and the merge preserves the threshold; the recursion writes the proof. These are not two facts but one. The generator's recursion and the prover's induction are the same skeleton — sort-two-halves-and-merge on the one side, two-thresholds-and-the-merge-preserves-them on the other. The theorem that a merged wire is a threshold is the generator's merge step, read as a proof.

So what is easily built by a recursive generator is easily proved by a recursive prover, because the packing has a grammar, and a grammar is exactly what both a generator and a prover walk. Tightness, recursion, and provability are one property. A schedule packs the kernel with zero slack precisely when it has the recursive grammar that makes it both generatable and provable; and where that grammar is present, the verifier's cost — the cost of reading the packing — is polynomial, because the verifier follows the same grammar the generator laid down.

The broken lattice, at its depth

The book's question was whether the width of the broken lattice is polynomial. The answer it has reached is a decomposition: the width is the kernel — the median threshold ramp, irreducible, of size about $n/2$ — plus a residue, the packing's slack. The kernel is proved; the residue is the open frontier. And the residue is not a mysterious defect to be bounded away by cleverness. It is the slack of a packing, driven to zero by recursive structure and left positive by tangle. The networks that matter — the recursive sorters — pack tight, residue zero, on the kernel; and they are exactly the networks whose packing has a grammar, hence exactly the networks a prover can prove and a verifier can read in polynomial time.

The broken lattice, at its depth, is a packing. Its unbroken part, the chain, is the kernel packed exactly. Its brokenness, the antichain, is the slack a loose schedule leaves. And the door that remains shut — whether every sorter's residue stays small — is the question of whether every sorter packs, or only the ones built with a grammar. The evidence of this book is that the grammar is where the tightness lives: to pack the kernel with no slack is to have the recursive structure that makes the network, and its proof, and its verification, one object. What an optimal sorting network computes in its comparator schedule is a tight packing of the median threshold, and the broken lattice is the ledger of how tightly it packs.

In the broken lattice

This is the broken lattice named to its depth. Its unbroken part — the chain, where the cofactors are nested — is the kernel packed exactly, the median threshold with no slack. Its broken part — the antichain, where cofactors are incomparable — is the residue, the slack a loose schedule leaves. Every chapter of this book has measured one facet of that ledger; this last one says what the ledger records: how tightly a comparator schedule packs the one volume every sorter must pack. The lattice is broken exactly to the degree the packing is loose, and whole exactly to the degree it is tight.

Chapter 28

The Ascent

The packing chapter closed the book's argument. This one opens a door the argument leaves ajar, and walks a little way through it, because what lies on the other side is too pretty to leave unremarked and too unfinished to claim. It is a way of reading a sorting network as a climb.

Folds and the mask lattice

On $n = 2^k$ wires, indexed by the vectors of $\mathbb{F}_2^k$, a *fold* by a nonzero mask m is the layer that pairs each wire x with $x \oplus m$ and settles the pair. It is a perfect matching — the antipode of the subcube in the direction of m — and its *weight*, the number of ones in m, is the dimension of the diagonal it folds along: a weight-one mask folds a single edge, the all-ones mask of weight k folds the whole cube along its body diagonal, the antipode itself.

The masks, ordered by inclusion, form the Boolean lattice of the nonempty subsets of the k basis directions, graded by weight. Its levels are the rows of Pascal's triangle: $\binom{k}{1}$ edges at the bottom, $\binom{k}{2}$ face diagonals above them, and so on up to the single body diagonal — the antipode — at the top. A sequence of folds is a walk in this lattice, and because folds compose by exclusive-or, the comparisons a sequence can reach are exactly the linear span of the masks it uses. To sort, the masks must span $\mathbb{F}_2^k$; a basis of k edges is the least that reaches every wire, and the higher folds do not extend the reach but sharpen the resolution.

The ascent sorts, and only upward

Read the lattice from the bottom up and a striking thing happens. The full ascending sequence — one fold at each mask, edges first, then faces, then the body diagonal — sorts. On eight wires the sequence of masks $001, 010, 100,\ 011, 101, 110,\ 111$ is a sorting network. The cube is folded first along its three edges, then along its three faces, and last along its body diagonal, and the result is sorted.

And the direction is not incidental; it is the whole of it. Ascending sequences sort; descending ones — the body diagonal folded first, the edges last — do not sort at all. Fold the whole cube onto itself before the edges are settled and the fold merely scrambles values across the levels of the count; fold it last, when the extremes are already in place, and it settles the centre. Sorting, read through the mask lattice, is a consolidation from the bottom up: the edges first, the antipode last, and never the reverse. The antipode that this book has followed as a symmetry appears here as the *summit* of a climb — the last fold, reachable only from below.

Two walks on the same ladder

The networks the earlier chapters weighed are two different walks on this one ladder.

The recursive sorter — Batcher, the kernel generator, the network whose residue is zero — stays low. It folds edges, and folds them again, resolving the order without ever climbing to a face or the body diagonal; every wire stays a threshold, and the width stays the kernel ramp. Kernel-tightness is a walk that never leaves the bottom of the lattice.

The size-optimal networks climb. Their opening is the same edge-walk — the hypercube prefix this book met in the antipode families and in Sergeev's reading of the sixteen-wire networks — but then they ascend: a face fold, another, and the body diagonal to close. They spend the climb to save comparators, and they pay for it in residue, the width bulging above the kernel where they leave the threshold behind. The two generators share the alphabet of masks; they differ in how high they walk. The kernel generator stays at the edges and packs tight; the optimal generator ascends the ladder once and packs small.

What the ascent offers, and what it withholds

The reading is not yet a theorem, and this chapter claims no more than it has. The full ascent sorts on eight wires; that it sorts on every 2^k is a conjecture, suggested by the recursion but not proved here. Among the ascending orders that respect the weight grading, some sort and some do not — there is a finer constraint on how the faces are sequenced against the edges, a genuine combinatorial rule that the exploration located but did not close. The chapter records these as open, in the spirit of the book: a plain frontier, named where it is not yet crossed.

What the ascent does offer is a change in the shape of the search. To look for an optimal network among all comparator layers is to search a space of matchings that grows as the double factorial — an eighty-nine-digit count of first layers alone at sixty-four wires, the reason no optimal network at that width is known. To look for one among the weight-ascending fold sequences is to walk a graded lattice of $2^k - 1$ masks, a space whose size is measured in the logarithm of the number of wires, and each candidate settled by the verifier's diagram in a fraction of a second. Whether the optimal networks live in this small space — whether optimality is a decorated ascent of the mask lattice — is the door this chapter opens and does not close. It is where the book's road, having reached its packing, turns to look at the mountain it climbed and sees, for the first time, that it was a mountain.

In the broken lattice

The mask lattice is not the broken lattice, but they meet at the antipode. The broken lattice's symmetry — the reflection this book has followed from the first chapters — is the top of the mask lattice, the body diagonal, the last fold of the ascent. And the climb is the packing seen in motion: the kernel generator, staying at the edges, packs the kernel exactly and leaves the lattice unbroken, a chain; the optimal generator, ascending, tightens the packing at the cost of breaking the lattice a little, a residue, in exchange for a smaller network. To sort is to climb from the edges of the cube to its centre, and the broken lattice is the record of how the climb was made — edge by edge and fold by fold, from the atoms of the cube to the antipode at its summit.

Chapter 29

The Program

The ascent read a sorting network as a climb through the mask lattice. This chapter asks a sharper question: can a network be written down not as a list of comparators but as a *program* — a compact sequence of instructions in the language of dimensions and classes — from which the comparators can be recovered exactly? The question is not idle. A network's comparator list is opaque; a program in the right language would expose the grammar, and a grammar can be searched, compared, and shortened.

The dimensional program

Each layer of a sorting network on 2^k wires is a matching, and each pair in the matching folds along a direction — the exclusive-or of its two indices, a mask in the hypercube lattice. So a first, crude program is: for each layer, the multiset of masks it uses. This records the *dimensions* the layer measures.

But the crude program is lossy. A mask of low weight has many disjoint pairs available on 2^k wires, and a layer typically folds only some of them; the mask alone does not say which. On sixteen wires the loss is mild; on thirty-two it is fatal. A single instruction — fold along this mask — corresponds to several candidate pairs, and the program cannot choose among them without more information. To recover the network, the program must name not only the dimension but the *class* of wires it applies to.

The classes, and why popcount is too coarse

After the hypercube prefix — the edge-walk that opens every network in this study — the wires fall into classes by their population count, the number of ones in the index. These are the tournaments: the champions at the extremes, the powers of two just inside them, the self-mirror middle at the centre. Reading a network's consume by these classes is illuminating: each class first sorts within itself, then the classes merge, and the whole carries the reflection symmetry of the broken lattice, the class of weight t mirroring the class of weight $n - t$.

But the population count is too coarse to serve as the program's class label. The trouble is the twist. When a tournament bisects — the pivotal move that sends half its wires climbing and keeps half continuing — the two halves have the *same* population count but different destinies. A wire shed to the high side and a wire kept on the low side are indistinguishable by weight, yet the program must fold them differently. Population count names the tournament but not the seat within it, and the consume needs the seat.

So the crude program fails not for want of dimensions but for want of *resolution in the classes*. The dimension is clear; the class is blurred. What is needed is a finer nomenclature — one that distinguishes the wires a tournament's bisection has separated, even when their weights agree.

In the broken lattice

The dimensional program is the broken lattice trying to speak its own grammar. The dimensions are the folds — the edges and diagonals of the cube; the classes are the tournaments — the population strata the prefix leaves behind. Between them they almost name the network, but the twist defeats them: it splits a class without changing a weight, and the coarse nomenclature loses the split. The lattice's brokenness is exactly this split — the antichain where wires of equal weight take unequal paths — and to write the program losslessly the language must be fine enough to name the break. The next chapter finds that language.

Chapter 30

The Lineage

The program of the previous chapter failed for want of resolution: the population count named the tournament but not the seat, and the twist split seats without moving weights. This chapter supplies the missing resolution, and with it writes any sorting network as a lossless program — recovering, from the program alone, the optimal thirty-two-wire network exactly.

The bisection lineage

The finer nomenclature is the wire's *lineage*: the record of which side it took at every comparator it has passed through. Begin each wire with its population count. Then, at each comparator, the wire that becomes the minimum appends one mark, the wire that becomes the maximum appends another. A wire's lineage grows as it flows through the network — a string of minima and maxima appended to its starting weight — and two wires of equal weight that the twist has sent to different sides now carry different lineages, because one was a minimum where the other was a maximum.

The lineage is a *structural* label. It is read from the network's own comparators, not from the values that flow along the wires; it needs no enumeration of inputs. This is what lets it scale: on thirty-two wires the reachable inputs number in the billions and cannot be tracked, but the lineage is computed forward from the comparators alone, at any width, for free.

The lossless program

With lineage as the class, the program becomes: for each layer, a list of instructions, each a dimension paired with the lineage of the wire that folds along it. To replay the program, rebuild the lineages forward as the layers are laid down, and for each instruction fold the unique wire of the named lineage along the named dimension. Because the lineage distinguishes every wire a tournament's bisection has separated, the instruction names its pair without ambiguity.

The program is lossless. Extracted from the systematic sixteen-wire network and replayed, it returns that network exactly, comparator for comparator. And — the test that matters — extracted from the optimal thirty-two-wire network of Dobbelaere and replayed, it returns *that* network exactly, all one hundred and eighty-five comparators, with no enumeration of the thirty-two-wire input space and no search. The population count could not do this; the lineage can. What blurred the classes was the twist's split, and the lineage is precisely the record of the split.

What the program opens

A lossless program is more than a curiosity. It is a canonical form: every sorting network compresses to its dimensional grammar, the dimensions it folds and the lineage-classes it folds them on. In this form the networks can be compared — Batcher's grammar against the systematic grammar against Dobbelaere's — and their differences read as differences of program, not of opaque comparator lists. And in this form the search for better networks changes shape: a shorter program is a smaller network, and the program lives in the space of dimensions and lineages, a graded and structured space, not the trackless space of all matchings. The generator the earlier chapters reached for — the one that would build optimal networks rather than search for them — wants exactly this language, and now has it.

In the broken lattice

The lineage is the broken lattice keeping its own genealogy. Population count names the stratum a wire belongs to; lineage names its descent — the branch of minima and maxima that brought it there. The twist that broke the lattice, splitting a stratum into a high line and a low line, is recorded exactly in the lineage's marks, and so the program that was blind to the split becomes lossless once it reads the descent. To write a sorting network as a program is to trace, for every wire, the lineage of choices that sorts it — and the broken lattice, read through the lineage, is the family tree of the sort: every wire's path from its stratum to its seat, from the weight it started with to the place it belongs.

Chapter 31

The Twist Family

The lineage gave a network a lossless program. This chapter puts the program to work: it changes a single move — the twist — and lets the program carry the change through the rest of the network automatically, producing a small family of distinct sorters that are the same everywhere but one layer. The family is found not by search but by the algebra of the program, and its size is dictated by a symmetry.

The twist as a swappable part

Recall the shape of these networks: an edge-walk prefix that populates the tournaments, a twist that bisects each tournament into a low side that continues and a high side that is shed, and a consume that routes both to the sorted order. The consume, written as a lineage program, addresses its folds not by wire index but by lineage — the record of the minima and maxima a wire has taken. This is what makes the twist swappable. Change the twist to a different bisection, and the wires acquire different lineages; but if the new bisection assigns the same lineages overall — the same multiset of descent-records — then the consume program, replayed, finds each of its folds exactly where it left them, and the network sorts unchanged below the twist.

So a twist swap is admissible precisely when it preserves the lineage multiset. Within that condition the twist is a free parameter: the consume follows automatically, no re-derivation, no search. A twist that changes the multiset is inadmissible — the consume would look for a lineage no wire carries, and come up short.

Two mirror swaps

On sixteen wires the tournaments are the population classes: the powers of two form one, its mirror — the complements of the powers — forms another, and the self-mirror middle sits between them. In the systematic network the twist bisects the powers-of-two tournament by pairing 1 with 4 and 2 with 8. There is another bisection of the same four wires — pairing 1 with 8 and 2 with 4 — and it preserves the lineage multiset: the same two wires end on the low side, the same two on the high. It is admissible. Replayed, the consume is untouched, and the network sorts. This is the first swap.

The mirror tournament admits the mirror swap. Its wires are the complements of the powers, and the systematic twist pairs 7 with 13 and 11 with 14; the admissible alternative pairs 7 with 14 and 11 with 13 — the reflection, through the centre, of the first swap. That the mirror move is admissible is not a separate discovery but a consequence of the reflection symmetry that runs

through the whole study: the tournament of weight t and the tournament of weight $n - t$ carry mirror-image programs, so an admissible swap in one has an admissible mirror in the other. This mirror swap was conjectured from the symmetry and confirmed by the program; it too leaves the consume untouched, and it too sorts.

The family of four

The two swaps are independent — one acts on the powers-of-two tournament, the other on its mirror, and they touch disjoint wires — so they compose. Taking each swap or leaving it gives $2 \times 2 = 4$ admissible twists, and four distinct sorters: the systematic network itself, the network with the first swap, the network with the mirror swap, and the network with both. All four are size sixty and depth ten; all four are identical except in the single twist layer; all four were verified to sort by exhaustive testing of the sixteen-wire zero-one inputs and independently by the ordered decision diagram. The family is an elementary abelian group of twist swaps — generated by two mirror-related involutions, acting freely and independently on mirror tournaments — and its four elements are the orbit of the systematic network under lineage-preserving twist change.

How they were computed

The computation is short, because the lineage program does the work. Extract the program of the base network; keep the prefix and replace the chosen pairs in the twist layer; check that the lineage multiset is preserved; then replay the consume program against the new prefix and twist, letting each instruction fold the unique wire of its named lineage. In outline:

```
def swap_twist(base, twist_index, replace_pairs, with_pairs, n):
  program = extract_program(base, n)
  prefix = base[:twist_index]
  new_twist = (base[twist_index] \ replace_pairs) ∪ with_pairs
  compatible = lineage_multiset(prefix + [new_twist])
    == lineage_multiset(prefix + [base[twist_index]])
  consume = program[twist_index+1:]
  return replay_consume(prefix + [new_twist], consume, n), compatible
```

where `extract_program` reads each layer as a list of (mask, lineage-of-min) instructions, and `replay_consume` rebuilds lineages forward and folds, for each instruction, the one wire that now carries the named lineage. The full generator is recorded in the public repository (`github.com/Ruach-Tov/public`) as `research/sorting-networks/tools/twist_swap.py`; each network it produced carries, in its provenance, the commit that generated it. The four members were found by calling this routine with the two mirror swaps and their composition, and by the empty call — the systematic network itself — that anchors the family.

In the broken lattice

The twist family is the broken lattice's reflection made into a group. The twist is the move that breaks a tournament in two — a low line and a high line of equal weight — and the two admissible swaps are the two ways to draw that break in each of the mirror tournaments. The reflection that this book has followed from its first pages, the symmetry of weight t against weight $n-t$, is exactly what makes the second swap admissible once the first is, and exactly what makes them compose

into a family of four. The lineage carries the change through the consume untouched, and so the family is a family of *programs* that agree everywhere but in the drawing of the break — four sorters that differ only in how they choose to reflect the lattice at its centre.

Chapter 32

The Source

This chapter lists, verbatim, the complete source needed to reproduce the twist-swap calculation of the previous chapter — the lineage program, the compatibility check, and the swap itself. It is the whole of `tools/twist_swap.py` as committed to the repository at `02c9d8cdd`. Nothing is elided; the routines shown are exactly those that extracted the lineage program of the systematic network, verified that a twist swap preserves the lineage multiset, replayed the consume, and emitted the members of the twist family.

The heart of the calculation is two functions. `extract_program` reads a network as a list of layers and returns, for each layer, a list of instructions — each a fold-mask paired with the lineage of the wire that takes the minimum side — while accumulating the lineage of every wire as the record of the minima and maxima it has taken. `swap_twist` keeps the prefix, replaces the chosen pairs in the twist layer, checks that the lineage multiset is preserved, and replays the consume program against the new prefix and twist, letting each instruction fold the one wire that now carries its named lineage. The remaining routines are support: lineage bookkeeping, a brute-force sort check over the sixteen-wire zero-one inputs, and the driver that reproduces the base member of the family.

```
#!/usr/bin/env python3
"""twist_swap.py -- generate a new sorter by swapping the twist and auto-rewiring the consume.

Given a network in lineage-program form (see lineage_program.py), this changes the TWIST layer
to a lineage-multiset-preserving alternative and lets the CONSUME auto-rewire via the lineage
labels. The result is a distinct sorter of the same size, provided the new twist preserves the
lineage multiset (the compatibility condition -- see TWIST-REWIRE.md).

This is the code that generated networks/n16_twistswap_60.json:
  base = systematic-60 (n=16); twist t1 bisection (1,4)(2,8) -> (1,8)(2,4); consume auto-rewired.

Usage:
  python twist_swap.py                # regenerate n16_twistswap_60 and verify
  python twist_swap.py --emit PATH    # write the network JSON to PATH
"""
import sys, json

def pc(x):
    return bin(x).count("1")

def layer_greedy(comps, n):
    lvl = [0]*n; layers = []
    for (i, j) in comps:
        d = max(lvl[i], lvl[j])
        while len(layers) <= d:
```

```
            layers.append([])
        layers[d].append((i, j)); lvl[i] = lvl[j] = d+1
    return layers

def load_layers(path, n):
    d = json.load(open(path))
    net = d["network"]
    if net and isinstance(net[0], list) and net[0] and isinstance(net[0][0], list):
        return [[tuple(c) for c in L] for L in net]
    return layer_greedy([tuple(c) for c in net], n)

def extract_program(layers, n):
    """Return the (mask, lineage-of-min-wire) program for the network."""
    label = {x: (pc(x),) for x in range(n)}
    program = []
    for L in layers:
        instrs = [(a ^ b, label[min(a, b)]) for (a, b) in L]
        program.append(instrs)
        for (a, b) in L:
            lo, hi = (a, b) if a < b else (b, a)
            label[lo] = label[lo] + ("m",)
            label[hi] = label[hi] + ("M",)
    return program

def build_labels(layers, n):
    label = {x: (pc(x),) for x in range(n)}
    for L in layers:
        for (a, b) in L:
            lo, hi = (a, b) if a < b else (b, a)
            label[lo] = label[lo] + ("m",)
            label[hi] = label[hi] + ("M",)
    return label

def replay_consume(prefix_and_twist, consume_program, n):
    """Replay the consume program against the given prefix+twist, auto-rewiring by lineage."""
    label = build_labels(prefix_and_twist, n)
    net = [list(L) for L in prefix_and_twist]
    for instrs in consume_program:
        layer = []; used = set()
        for (mask, lin) in instrs:
            found = None
            for x in range(n):
                y = x ^ mask
                if x < y and x not in used and y not in used and label[x] == lin:
                    found = (x, y); break
            if found:
                layer.append(found); used |= {found[0], found[1]}
        net.append(layer)
        for (a, b) in layer:
            lo, hi = (a, b) if a < b else (b, a)
            label[lo] = label[lo] + ("m",)
            label[hi] = label[hi] + ("M",)
    return net

def lineage_multiset(layers, n):
    return sorted(build_labels(layers, n).values())

def swap_twist(base_layers, twist_index, replace_pairs, with_pairs, n):
    """Return a new network: base prefix+modified-twist, consume auto-rewired.
```

```
    replace_pairs/with_pairs are lists of (a,b) tuples in the twist layer to swap."""
    program = extract_program(base_layers, n)
    prefix = [list(base_layers[i]) for i in range(twist_index)]
    orig_twist = [tuple(sorted(c)) for c in base_layers[twist_index]]
    rep = set(tuple(sorted(p)) for p in replace_pairs)
    new_twist = [c for c in orig_twist if c not in rep] + [tuple(sorted(p)) for p in with_pairs]
    new_twist = [tuple(sorted(c)) for c in new_twist]
    # compatibility check: lineage multiset preserved?
    compatible = lineage_multiset(prefix + [new_twist], n) == lineage_multiset(prefix + [
    orig_twist], n)
    consume_prog = program[twist_index + 1:]
    net = replay_consume(prefix + [new_twist], consume_prog, n)
    return net, compatible

def sorts_bruteforce(n, comps):
    for x in range(2**n):
        v = [(x >> b) & 1 for b in range(n)]
        for (i, j) in comps:
            if v[i] > v[j]:
                v[i], v[j] = v[j], v[i]
        if any(v[a] > v[a+1] for a in range(n-1)):
            return False
    return True

def generate_n16_twistswap_60():
    """The exact generation of networks/n16_twistswap_60.json."""
    n = 16
    base = load_layers("networks/n16_systematic_60.json", n)
    net, compatible = swap_twist(base, twist_index=4,
                                 replace_pairs=[(1, 4), (2, 8)],
                                 with_pairs=[(1, 8), (2, 4)], n=n)
    comps = [c for L in net for c in L]
    ok = sorts_bruteforce(n, comps)
    return net, compatible, ok, len(comps), len(net)

if __name__ == "__main__":
    net, compatible, ok, size, depth = generate_n16_twistswap_60()
    print(f"n16_twistswap_60: size={size}, depth={depth}, "
          f"twist-compatible(lineage-multiset-preserved)={compatible}, sorts(2^16 brute)={ok}")
    if "--emit" in sys.argv:
        path = sys.argv[sys.argv.index("--emit") + 1]
        data = {
            "n": 16,
            "network": [[list(c) for c in L] for L in net],
            "size": size,
            "depth": depth,
            "origin": ("lineage twist-swap of systematic-60 (t1 bisection (1,4)(2,8)->(1,8)(2,4),
    "
                       "consume auto-rewired). Generated by tools/twist_swap.py."),
            "sorts": ok,
        }
        json.dump(data, open(path, "w"), indent=1)
        print(f"  emitted {path}")
```

To reproduce the twist family, run the driver `generate_n16_twistswap_60()` for the base swap, and call `swap_twist` with the mirror pairs $\{(7,13),(11,14)\} \to \{(7,14),(11,13)\}$ for the mirror member and with both pair-sets for their composition. Each call returns the network and the

boolean recording whether the swap preserved the lineage multiset; each network was checked to sort both by `sorts_bruteforce` above and, independently, by the ordered decision diagram. The lineage machinery this file relies on — `extract_program`, `build_labels`, `replay_consume` — is duplicated here in full so that the file stands alone; the same routines, and the round-trip that regenerates a network from its lineage program, live also in `tools/lineage_program.py`.

In the broken lattice

Source code is an unusual thing to set in a book of proofs, but it belongs here for the same reason the proofs do: it is checkable. A reader who doubts that the twist family sorts, or that the lineage carries the consume through a twist swap untouched, need not trust the prose — the routines are here, short and complete, and the repository records the commit that ran them. In the broken lattice, the program is the genealogy of the sort; and this chapter is the genealogy of the program, set down in full so that the family of four can be reflected, once more, by anyone who cares to run it.

Envoi

Gather what the six chapters certify. A comparator network is monotone; its verification reduces to 0/1 inputs, and then to the settled adjacent pairs. The cofactors of its monotone wire functions are themselves monotone, Shannon-ordered, and bounded by their constant-prefix extremes. Boolean duality is the antipode of this world, an order-reversing involution under which a sorter's wire i is wire $n-1-i$ — the minimum and the maximum, the ultimate antipodes, exchanged. And the cofactor set at each level is a sub-poset of a distributive interval, bounded by the constant-prefix cofactors and folded around its centre by that antipode: the broken lattice, half of an antipodal pair, whole in itself. For a network of odd width one wire is fixed by that antipode — the median, its own mirror image, and the widest — a small tautology, obvious once seen and yet needing the whole construction to state. And the theorems, on a last inward look, prove to be one figure: the threshold results instances of the monotone, the automaton width and the lattice width a single bound.

A score of theorems, one object, and one conclusion. For the sorters — the networks a verifier verifies — we have carried the lemmas up through the superstructure to a machine-checked polynomial cost, and closed the width bound: the cofactor set of a sorted wire is a chain, and its diagram width — the number of distinct cofactors at a level — is at most $\min(t, n+1-t)+1 \leq \lfloor (n+1)/2 \rfloor + 1 = \mathcal{O}(n)$, proven exactly from the threshold structure. We have not bounded the width for arbitrary comparator networks, nor computed the volume of its polyhedral form. We have drawn its outline and shown that the outline closes. It is, whatever else it is, a beautiful thing to have found at the heart of a sorting network.

All theorems machine-checked in Isabelle/HOL. Sessions in `isabelle/`*; exploratory scripts in* `tools/`*. Every build finishes clean.*

Chapter 33

Bibliography

The machine-checked development of this book is self-contained: every theory imports only Isabelle/HOL's `Main`, and every result is proved from first principles. The prose, however, names the classical theorems its constructions instantiate — the lattice theory of Dilworth, Mirsky, Birkhoff, and Sperner; the decision-diagram theory of Bryant and of Jha and Suciu; the Coxeter combinatorics of the Bruhat order; and the foundational facts of sorting networks — and cites in Chapter 21 a structural result of Codish and coauthors. The works are listed here in full.

[**Birkhoff 1937**] Garrett Birkhoff. *Rings of sets.* Duke Mathematical Journal, 1937. (Every finite distributive lattice is the lattice of down-sets of its join-irreducibles.)

[**Björner & Brenti 2005**] Anders Björner and Francesco Brenti. *Combinatorics of Coxeter Groups.* Graduate Texts in Mathematics 231, Springer, 2005. (The subword property of the Bruhat order, §2.2, due to Tits.)

[**Brualdi & Deaett 2007**] Richard A. Brualdi and Louis Deaett. *More on the Bruhat order for* $(0,1)$*-matrices.* Discrete Mathematics, 2007. (Largest antichains in the Bruhat order; in $A(2k,k)$ the antichain reaches k^8 against a chain of k^4.)

[**Bryant 1986**] Randal E. Bryant. *Graph-based algorithms for Boolean function manipulation.* IEEE Transactions on Computers, 1986. (Ordered binary decision diagrams and their canonical minimal form via merging equal residual functions.)

[**Codish 2014**] Michael Codish, Luís Cruz-Filipe, and Peter Schneider-Kamp. *Sorting Networks: the End Game.* arXiv:1411.6408 [cs.DS], 2014 (the version consulted for this book). (The precursor to [Codish 2017]: the last-two-layer block theory — at most one mixed block (Lemma 8), comparators connect adjacent blocks (Theorem 11), a k-block with n comparators has at most $n+1$ channels (Corollary 12) — and the last-layer normal form, whose count is the Padovan sequence (Theorem 14).)

[**Codish 2017**] Michael Codish, Luís Cruz-Filipe, Thorsten Ehlers, Mike Müller, and Peter Schneider-Kamp. *Sorting Networks: to the End and Back Again.* Journal of Computer and System Sciences, 2017. Preprint: arXiv:1507.01428 [cs.DS], 2015 (the version consulted for this book).

The paper studies the front and back ends of a sorting network. Its Section 3.1 proves that the channels of a non-redundant sorting network partition, after each layer, into consecutive blocks between which values never move; that for each 0/1 *input at most one block is mixed (Lemma 5, p. 10); that a* k*-block with* m *comparators spans at most* $m+1$ *channels (Corollary 3, p. 11); and that the comparators of the last layer are all adjacent (Lemma 4, p. 8). A*

duality between the symmetries of the first and last layers (p. 1) parallels the antipode of the present work.

[Dedekind 1897] Richard Dedekind. *Über Zerlegungen von Zahlen durch ihre größten gemeinsamen Teiler.* Festschrift der Technischen Hochschule zu Braunschweig, 1897. (The Dedekind numbers: monotone Boolean functions of n variables, growing doubly exponentially.)

[Drechsler 2021] Rolf Drechsler. *Polynomial Circuit Verification using BDDs.* arXiv:2104.03024 [cs.LO], 2021 (the version consulted for this book). (If every internal signal's BDD is polynomial, the whole construction is polynomial via the bounded apply/ITE operation; tree-like and structured circuits.)

[Dilworth 1950] Robert P. Dilworth. *A decomposition theorem for partially ordered sets.* Annals of Mathematics, 1950. (The largest antichain equals the least chain cover of a finite poset.)

[Jha & Suciu 2012] Abhay Jha and Dan Suciu. *On the tractability of query compilation and bounded treewidth.* Proceedings of the International Conference on Database Theory (ICDT), 2012. (Bounded circuit pathwidth and bounded OBDD width are mutually bounded.)

[Knuth 1998] Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Second edition, Addison-Wesley, 1998. (§5.3.4 'Networks for Sorting'; the zero-one principle, and exercises 28 and 31 on pp. 239–240 for the elementary-symmetric-function form and the Dedekind count.)

[Mirsky 1971] Leon Mirsky. *A dual of Dilworth's decomposition theorem.* American Mathematical Monthly, 1971. (A finite poset with longest chain H is a union of H antichains.)

[Sergeev 2018] Igor S. Sergeev. *Some comments on the structure of the best known networks sorting 16 elements.* arXiv:1810.11262 [cs.DS], 2018 (the version consulted for this book). (A structural analysis of the Green and van Voorhis 16-element networks: the initial approximate phase orders the inputs into a Boolean cube poset; the two networks share this phase and the settling of the extremes, and differ only in how they sort the eight-element middle set.)

[Bundala et al. 2017] Daniel Bundala, Michael Codish, Luís Cruz-Filipe, Peter Schneider-Kamp, and Jiří Závodný. *Optimal-depth sorting networks.* Journal of Computer and System Sciences, 2017; preprint arXiv:1412.5302. (The optimal depth of sorting networks on 11 to 16 channels; cited via [Sergeev 2018] for the depth-9 optimality of the van Voorhis 16-network.)

[Sperner 1928] Emanuel Sperner. *Ein Satz über Untermengen einer endlichen Menge.* Mathematische Zeitschrift, 1928. (The largest antichain of the Boolean lattice on k elements is $\binom{k}{\lfloor k/2 \rfloor}$.)

The footnotes of Chapter 21 cite specific pages of [Codish 2017]:

- p. 9 — the k-blocks partition the channels into consecutive runs;
- p. 10, Lemma 5 — for each 0/1 input, at most one k-block is mixed;
- p. 11, Corollary 3 — a k-block with m comparators has at most $m + 1$ channels;
- p. 1 — the front-back duality of layer symmetries.