Machine-Checked Correctness of the Discharge Calculus for Sorting-Network Verification

Iyun
(Ruach Tov collective)

2026-07-24

Abstract

We formalize, in Isabelle/HOL (Isabelle2025), the correctness of the *discharge calculus*—the pairwise ordering-state verifier at the core of the polynomial sorting-network verification method described in the main text. The formalization establishes three claims of record, all machine-checked: the calculus's verdict is *exactly* correct (`correctness_capstone`: a full pairwise verdict holds iff the network sorts); the verifier *never falsely claims sorting* (`sound_always`, backed by a `fail_safe_asymmetry` corollary that the engine never over-discharges); and *every sorter is verifiable* (`complete_semantic`). We are careful to separate what is *proven*—soundness and semantic completeness—from what is honestly *conjectured*: the *polynomial feasibility* of deriving the required facts, which governs speed but, by `fail_safe_asymmetry`, neither correctness nor soundness. Alongside the correctness proof we describe two complementary tools: an Isabelle-emitting verifier that renders any concrete network's ledger as machine-checkable lemmas, and an Isabelle-native *deriver* ("Route B") in which the theorem prover itself runs the discharge derivation, so that the derivation *is* the proof. A complete build recipe is given so that the green build—and hence every proof—can be reproduced.

1 Scope and epistemic contract

The purpose of this formalization is to place the sorting-network verifier's *correctness* on a machine-checked footing, and to make the boundary between proof and conjecture explicit rather than rhetorical. Concretely, the Isabelle development proves:

- **Soundness.** Whenever the verifier reports "sorts," the network genuinely sorts. This is unconditional: it does not depend on any efficiency assumption.
- **Semantic completeness.** Every network that sorts admits the full verdict; the facts the verdict requires are all true, hence derivable by the calculus's rules.

It does *not* prove, and does not claim to prove:

- **Polynomial feasibility of derivation.** That the required facts can be *derived in polynomial time* is the "fourth face"—an honest conjecture addressed by the separate complexity argument in the main text. Crucially, by the `fail_safe_asymmetry` corollary below, soundness holds *independently* of this conjecture: even if derivation were slow, a reported "sorts" verdict is still correct.

This separation is not a footnote; it is written into the theorem structure itself (see `complete_semantic`, whose accompanying text names the conjecture and localizes it to *speed*). A reviewer who wishes to reject the polynomiality claim may do so without disturbing any machine-checked theorem.

2 The apparatus at a glance

The development comprises eighteen theory files. Their dependency skeleton, and the load-bearing results, are summarized in Table 1. Three tiers of tooling operate over the same discharge calculus:

Certify (the calculus is correct). `DischargeCalculus.thy`, `BranchAtoms.thy`, `CoveringFamily.thy`, `Correctness.thy` — the *capstone chain*. Proves, once and for all networks, that the calculus's verdict is exactly correct.

Check (this network's ledger proves). `isabelle_emit.py` — a Python emitter that, given a concrete network, writes its ledger as Isabelle lemmas (each column a claim over min / max expressions, discharged by linear-order reasoning), then invokes Isabelle to confirm they prove.

Derive (Isabelle runs the derivation). `Derivable.thy`, `DeriveExec.thy`, `Deriver.thy` — "Route B": an *executable* deriver, lifting the proved-sound rules into a function that, given a network, produces the derived facts. Here the derivation itself is the certificate; there is no external trust in a Python derivation.

3 Foundations: semantics and the 0/1 principle

A comparator acts on a configuration by writing the minimum of two wires to the lower index and the maximum to the higher; a network is a list of comparators; `run` folds the network over an input. Sortedness and the verifier's target predicate `sorts` are defined over *Boolean* configurations, invoking Knuth's 0/1 principle (a comparator network sorts all inputs iff it sorts all 0/1 inputs). Defining `sorts` at type `bool` keeps the value type from escaping a bound quantifier, which is what makes the downstream proofs clean.

```
definition apply_comp :: "comparator => ('a::linorder) config => 'a config"
fun       run        :: "network => ('a::linorder) config => 'a config"
definition sorted_cfg :: "nat => ('a::linorder) config => bool"
definition sorts      :: "nat => network => bool"
```

The base fact is the comparator axiom: after a comparator (i, j), the output at i is at most the output at j, in every branch.

```
lemma comparator_axiom:
  "let c = apply_comp (i,j) v in (c i) <= (c j)"   (* schematic form *)
```

Two structural lemmas support lifting per-context certificates to the whole configuration space: the *frame rule* (`apply_comp_untouched`: wires other than i, j are unchanged) and `run_agree_below` (a run depends only on wires below n).

4 The pairwise-preorder heart (§5d)

The soundness backbone is that the discharge calculus never leaves the *pairwise-preorder fragment*, within which transitive closure is complete. Branch atoms—the ordered and transposed hypotheses a comparator introduces during MEASURE—*are* pairwise facts, so the entire derivation is transitive reasoning over a partial order.

Theory	Imports	Role / load-bearing results
`DischargeCalculus`	`Main`	Comparator semantics; `sorts` over Boolean configs (the 0/1 principle); `comparator_axiom`, frame rule, `run_agree_below`; `transitivity_sound`, `pairwise_closure_sound`.
`BranchAtoms`	`DischargeCalculus`	The §5d soul: branch atoms *are* pairwise facts (`branch_atom_closure`); the derivation never leaves the pairwise-preorder fragment.
`CoveringFamily`	`BranchAtoms`	`covering_family_discharge_sound`; `fail_safe_asymmetry`; `never_over_discharges` — soundness independent of the polynomial-feasibility conjecture.
`Correctness`	`CoveringFamily`	`correctness_capstone` (sorts $\leftrightarrow$ verdict); `sound_always`; `complete_semantic`. The claim of record.
`Derivable`	`DischargeCalculus`	Route B specification: what a derivation must produce (Iyun's spec side).
`DeriveExec`	`Derivable`	Route B executable deriver (Bocher's half); `exec_sound` is the meet-point with the spec.
`Deriver`	`DeriveExec`	Per-network executable derivation front-end.
`Certificate_n4_*`	(above)	Concrete n=4 certificates, including `Certificate_n4_Complete` (the optimal n=4 network proven to sort as a formal theorem).
`CertificateA`, `SBridge`	(above)	Route A tie (`sorts_bool_eq_sorts`): the cheap per-network 0/1 certificate is *formally equal* to the deep `sorts` predicate.

Table 1: The Isabelle/HOL apparatus. The *capstone chain* (top four rows) is the correctness proof of record; the remaining files provide the executable deriver and concrete certificates.

```
lemma transitivity_sound:
  "a <= b ==> b <= c ==> a <= c"

lemma pairwise_closure_sound:
  (* if every pair in P satisfies x <= y, then every pair reachable by the
     reflexive-transitive closure P^* also satisfies a <= b *)
```

`pairwise_closure_sound` is proved by induction on the reflexive-transitive closure (`rtrancl_induct`). It is the load-bearing lemma: every fact the calculus banks by transitive propagation from true premises is itself true. In `BranchAtoms.thy` this is completed by `branch_atom_closure`, establishing that the branch atoms are exactly the pairwise facts the closure operates on.

5 The covering family and fail-safe asymmetry (§13)

Discharge of a conditional fact requires an *exhaustive* family of contexts each of which implies the fact. `CoveringFamily.thy` proves this discharge sound, and—crucially for the epistemic contract—proves the *asymmetry* that makes soundness independent of efficiency.

```
theorem covering_family_discharge_sound:
  fixes Ks :: "('a::linorder) context_den set" and R :: "'a context_den"
    and phi :: "'a fact"
  assumes members_imply: "ALL K : Ks. implies_fact K phi"
  assumes exhaustive:    "exhaustive_over Ks R"
  shows ... (* phi holds on all of R *)

theorem fail_safe_asymmetry:
  assumes "engine_discharges checked Ks R phi"
  shows   "ALL x : R. phi x"

corollary never_over_discharges:
  assumes "~ (ALL x : R. phi x)"
  shows   "~ engine_discharges checked Ks R phi"
```

`never_over_discharges` is the contrapositive of `fail_safe_asymmetry`: if φ does not hold on all reachable configurations, the engine does not discharge it. Hence every "sorts" verdict (all $\binom{n}{2}$ facts discharged) is true. **Soundness is independent of the polynomial-feasibility conjecture:** a slow or incomplete search can fail to *find* a verdict, but it can never *fabricate* a false one.

6 The claim of record

`Correctness.thy` assembles the spine. The statements below are *verbatim* (ASCII-normalized from the Isabelle source).

```
theorem correctness_capstone:
  "sorts n net <->
     (ALL v::bool config. ALL i j.
        i < j --> j < n --> (run net v) i <= (run net v) j)"
  by (rule sorts_unfold_pairwise)

theorem sound_always:
  assumes "ALL v::bool config. ALL i j.
             i < j --> j < n --> (run net v) i <= (run net v) j"
  shows "sorts n net"
  by (rule verdict_implies_sorts[OF assms])

theorem complete_semantic:
  assumes "sorts n net"
  shows "ALL v::bool config. ALL i j.
           i < j --> j < n --> (run net v) i <= (run net v) j"
  by (rule sorts_implies_verdict[OF assms])
```

In words:

`correctness_capstone` The verdict is *exactly* correct: a full pairwise verdict holds iff the network sorts.

`sound_always` Verdict ⇒ sorts: the verifier never falsely claims sorting.

`complete_semantic` Sorts ⇒ verdict: every sorter is verifiable. The accompanying source text localizes the remaining conjecture: "Polynomial FEASIBILITY of deriving them is the honest conjecture (the fourth face)—it governs SPEED, not correctness, and by `CoveringFamily.fail_safe_asymmetry` not soundness."

7 From certifier to proof generator

Two tools turn the correctness proof into a *per-network* instrument that others can run on their own networks.

7.1 Isabelle-emitting verifier (`isabelle_emit.py`)

Given a network JSON, `isabelle_emit.py` calls the shared verifier engine (`verifier.py`, the single source of truth for the discharge semantics) to obtain the per-column ledger, then emits each column as an Isabelle lemma over min / max expressions, proved by `(auto simp: Let_def min_def max_def)`—pure linear-order reasoning. Invoking Isabelle on the emitted theory machine-checks the ledger's *entire* claim history, not merely final sortedness. (This tool originated in Heath's suggestion of 2026-07-18: "have your ledger emit terms in Isabelle/HOL, and verify that they prove.")

7.2 Route B: Isabelle as a per-network deriver (`Derivable.thy`, `DeriveExec.thy`, `Deriver.thy`)

Route B is stronger than emission-and-check. Rather than have Isabelle *check* facts a Python engine derived, Isabelle *runs the derivation itself*. The proved-sound rules of `DischargeCalculus.thy` are lifted into an executable function (`DeriveExec.thy`, whose `exec_sound` theorem is the meet-point with the `Derivable.thy` specification); `Deriver.thy` provides the per-network front-end. The derivation *is* the proof: there is no external artifact to trust. Route B was developed incrementally; its unconditional core (comparator axiom, frame rule, transitive closure) is machine-checked, with the conditional/covering-family layer following in later increments.

7.3 Route A tie (`CertificateA.thy`, `SBridge.thy`)

Finally, `sorts_bool_eq_sorts` proves that the cheap per-network 0/1 certificate is *formally equal* to the deep `sorts` predicate of `DischargeCalculus.thy`. The efficient certificate and the deep correctness capstone are therefore *one* claim, not two that happen to agree.

8 Reproducing the build

The proofs were developed and re-verified under **Isabelle2025**. To reproduce the green build of the correctness capstone, place the four capstone theories in a directory with the following `ROOT` file:

```
session PubVerify = HOL +
  theories
    DischargeCalculus
    BranchAtoms
```

```
    CoveringFamily
    Correctness
```

and run:

```
isabelle build -D .
```

A successful run prints `Finished PubVerify` with no error lines; every theorem above is then machine-checked. This is the build we rely on for the claim of record (`correctness_capstone`, `sound_always`, `complete_semantic`), and it is confirmed green under Isabelle2025.

The remaining files divide into two groups. The concrete n=4 certificates (`Certificate_n4_*`, `CertificateA`, `SBridge`) are ordinary proof scripts and build in the same way. The Route B *executable* deriver (`DeriveExec`, `Deriver`) is different in kind: because it *runs* the discharge derivation inside HOL rather than merely stating and proving lemmas, a full-session build that evaluates derivations is computationally heavy, and we do not report a whole-session green build of the executable deriver here. Route B's *unconditional core*—the proved-sound rules it lifts from `DischargeCalculus.thy`, and the `exec_sound` meet-point with the `Derivable.thy` specification—is machine-checked; the executable evaluation layer is engineering atop those checked foundations, not part of the correctness claim of record. A reviewer reproducing this work should build the capstone session above; the Route B executable layer is provided for those who wish to run per-network derivations, with the caveat that HOL-level evaluation is slow and is not required for any theorem in this paper.

What a reviewer verifies. Running the build reproduces, mechanically: (i) the soundness of the discharge calculus; (ii) the exact-correctness capstone; (iii) the fail-safe asymmetry that isolates soundness from the polynomiality conjecture. It does *not* certify the polynomiality claim, which is established separately in the main text and cross-checked independently; the Isabelle development deliberately does not overreach into that territory.

Attribution

The formalization is a collective effort of the Ruach Tov research collective. The discharge calculus is Heath's, developed over 2016–2019 and reconstructed in this project. The theory-layer correctness proof (capstone chain) and the Route B specification are Iyun's, on Heath's direction; the executable deriver (`DeriveExec.thy`) is Bocher's half, meeting Iyun's specification at `exec_sound`. The ledger engine and covering-family mechanization draw on Doresh's contributions. All theorems were machine-checked before commit, and the capstone build was independently re-verified.