

Mapping the SAT/UNSAT Frontier of Sen’s Liberal Paradox at $n = 2, m = 4$: An Auditable Possibility Atlas with Proof- and Witness-Carrying Evidence

Shunya Hayashida
Department of Liberal Arts, Faculty of Liberal Arts
The Open University of Japan
1720067169@campus.ouj.ac.jp

Abstract

Solver-aided mechanism design and computational social choice increasingly risk collapsing into “the solver said so”: when a result is surprising, it is often unclear whether the problem lies in the semantic specification, the encoding, the proof object, or the reported witness. This paper argues that the key first step is diagnosability. We make solver-supported impossibility exploration diagnosable by splitting obligations into Proof (Lean-kernel checking of the committed semantic theorem and committed UNSAT reference artifacts via LRAT replay), Audit (mechanical checks of CNF/manifest conformance and recorded metadata), and Witness (independent validation that a reported SAT model satisfies a concrete CNF instance).

We demonstrate this idea on `sen24`, a fixed finite instance used here as a minimal complete worked example for making the trust boundary explicit. The result is an auditable possibility atlas: a reproducible record of which axiom combinations are SAT or UNSAT, together with boundary explanations, validated SAT examples, and a narrowly scoped proof-carrying UNSAT reference.

The paper does not claim a new social-choice theorem or a new SAT proof method. Its contribution is a reproducible diagnostic workflow in which solver error, artifact drift, proof tampering, and witness misreporting become separately detectable and replayable, while the remaining end-to-end semantic encoding correctness is kept explicit as an assumption boundary rather than hidden inside solver trust.

1 Introduction

Solver-supported reasoning is becoming a practical tool for institutional design, impossibility exploration, and rule synthesis in computational social choice. But once a pipeline depends on an external solver, a familiar problem appears: when an output is implausible, one often cannot tell whether the fault lies in the semantic specification, the CNF encoding, the proof log, the reported witness, or the surrounding metadata. These failure modes are typically confounded under a single opaque verdict, namely that “the solver said so.” For a reviewer, implementer, or downstream user, this is not just a philosophical concern; it blocks debugging, narrows trust, and makes disagreement between semantic and artifact-level reasoning hard to localize. What is missing is diagnosability: a way to determine which layer is responsible when solver-supported results and mathematical expectations do not line up.

This paper proposes such diagnosability through an auditable possibility atlas equipped with an explicit Proof/Audit/Witness split, while keeping the remaining end-to-end semantic encoding correctness as an explicit assumption boundary. In our setting, Proof means Lean-kernel checking of the committed semantic theorem and committed UNSAT references via LRAT replay; Audit

means mechanical checking of DIMACS/manifest conformance, clause-family accounting, and recorded hashes; Witness means independent validation that a reported SAT model satisfies the concrete CNF instance it is claimed to solve. We develop this contract only for the narrow sen24 setting, used as a minimal complete worked example: a fixed finite case in which the trust boundary can be made explicit end to end without claiming broader theorem generality. The paper does not present a new social-choice theorem or a new SAT method. Its contribution is a reproducible diagnostic workflow that makes frontier results, failures, and supporting artifacts separable, inspectable, and auditable.

What is new here is not SAT in social choice. The closest prior line is SAT-based computational social choice, including impossibility proving by Tang and Lin [7], automated exploration and proof discovery in the Geist–Endriss line [5], and later SAT-based synthesis work by Brandt and Geist [3]. This paper does not claim a new social-choice theorem or a new SAT proof method. Its contribution is to package one fixed sen24 atlas workflow as an explicit auditable contract: what is proved in Lean, what is only audited against manifests and side conditions, what remains assumed, and which artifacts a reviewer can replay.

At a higher level, the paper makes three concrete contributions:

- We formalize a diagnosable trust contract separating Proof, Audit, and Witness obligations, while surfacing the remaining end-to-end encoding correctness as an explicit Assumption boundary.
- We instantiate the contract for sen24, including Lean kernel checking for a committed UNSAT reference and witness-validated SAT rule artifacts.
- We package replay commands, hashes, and artifacts as an auditable possibility atlas enabling detection of CNF drift, proof tampering, and witness misreporting.

The paper-level contribution ladder follows the official claims map exactly:

- **C1:** for sen24, axiom-lever enumeration yields a reproducible SAT/UNSAT frontier;
- **C2:** UNSAT boundary cases are explainable via one MUS and one small MCS candidate;
- **C3:** a committed UNSAT reference can be attached as LRAT and checked in Lean;
- **C4:** symmetry reduction and monotone pruning are guarded by explicit assumptions and runtime validators;
- **C5:** relaxed SAT cases can be filtered into a deterministic, auditable gallery with explicit SAT witness validation;
- **C6:** repair enumeration is checked against an independent optimum baseline by triangulation.

The stance of the paper is deliberately narrow. We work only at sen24, we treat auditability as the primary systems requirement, and we do not promote small-scope evidence into a claim about arbitrary n, m or a fully mechanized encoding-correctness theorem for the Python generator. The goal is to show that a killable instance can still support a meaningful auditable atlas with committed proof-carrying UNSAT references and validated SAT witnesses when the trusted computing base and guarantee boundary are made explicit.

Section 3 defines the trust model and the proof/audit/assumption split. Section 4 describes the sen24 encoding and the paper-facing artifact contract. Section 5 reports the claim-by-claim evidence accounting and re-verification status for this revision. Section 6 presents the sen24 case study, including frontier artifacts, one MUS plus one small MCS candidate, validated SAT rule cards, and the stronger C6 repair triangulation stage. Section 7 positions the paper against prior COMSOC-SAT and certificate-checking work. Section 8 closes with the scope limits that keep the paper honest.

2 Problem and Positioning

We study a fixed finite domain (`sen24`): two voters, four alternatives, and a social welfare function represented as a binary relation. Axiom toggles are treated as levers, producing a finite family of SAT instances.

Our positioning is automated-reasoning-first with COMSOC relevance:

- **Automated reasoning:** reproducible artifact generation, schema versioning, solver metadata, trusted checking, and explicit trust boundaries.
- **COMSOC:** impossibility boundary analysis, minimal contradiction causes (MUS), and minimal repair actions (MCS).

The objective is not to claim general n, m theorems, but to establish a high-integrity small-scale platform whose artifacts can be reviewed, rerun, and extended.

3 System Design

The `sen24` atlas is implemented through four separable layers. The purpose of the split is not software modularity alone; it is to keep the trusted computing base (TCB) small and explicit for the atlas artifacts that support C1–C6.

Spec. Lean definitions capture relation primitives, axioms, and theorem statements for `sen24`.

Encode. Python generators emit DIMACS CNF with manifests that record schema parameters, per-category clause counts, and hashes.

Solve. External SAT solvers produce SAT models or UNSAT proof traces.

Check. Trusted checks include: (i) CNF audit scripts against the manifest contract, (ii) LRAT verification in Lean for committed UNSAT references, and (iii) SAT witness validation against CNF.

This separation enables fast heuristic experimentation in Python while preserving a narrow proof-carrying core in the checking layer. In this paper, *proof-carrying* means that a concrete CNF artifact is accompanied by an LRAT trace that is replayed by Lean for a committed reference case. *Audited* means that artifact-level conformance is checked mechanically against the `sen24` manifest contract without claiming a full semantic encoding-correctness proof.

3.1 Trust Model and Chain of Trust

Figure 1 fixes what is trusted, what is not, and what kind of tampering is expected to be detectable.

3.2 Proof, Audit, and Assumption

We separate three kinds of checked statement and one explicit assumption boundary.

Proof. The semantic `sen24` theorem is proved in Lean, and committed UNSAT references are replayed in Lean from LRAT. These statements depend on the Lean kernel rather than on the SAT solver.

Trusted semantics/checking. Lean definitions and semantic theorems; the Lean kernel and FromLRAT; the independent sen24 auditor; committed artifact hashes.

Untrusted search/execution. The external SAT solver, the Python encoder as an implementation artifact, the OS, and the filesystem.

Chain of trust. Lean spec \rightarrow audited DIMACS + manifest \rightarrow untrusted SAT solving \rightarrow LRAT/model outputs \rightarrow Lean LRAT replay or SAT witness validation.

Detection surface. CNF or manifest drift is exposed by hash mismatches and audit failure; LRAT tampering fails kernel checking; SAT model tampering fails witness validation; metadata tampering is not itself a proof failure but is auditable by replay against recorded hashes and commands.

Figure 1: The sen24 trust model. Solvers are used for search, not as trusted decision or proof oracles.

Audited invariant. The DIMACS side is checked against a fixed sen24 schema: variable numbering, clause-family membership, manifest consistency, and hash linkage are reconstructed independently of the generator. For the rationality side, we also run a finite sen24 Python check showing that, under asymmetry on four alternatives, forbidding directed 3-cycles and 4-cycles is equivalent to acyclicity. This bridge is an audited finite-domain fact for sen24, not a Lean theorem and not an end-to-end encoding-correctness result.

Witness validation. Reported SAT rules are checked at the artifact level: the witness validator confirms that a concrete model satisfies a concrete CNF instance, but this is not a semantic proof of the rule beyond that instance.

Assumption. We do not claim a fully mechanized encoding-correctness proof from Lean semantics to Python-generated CNF. The generator is trusted to implement the audited sen24 contract, and the short-cycle/acyclicity bridge remains outside the trusted kernel as a sen24-specific finite check. Those boundaries are what reviewers can inspect and replay.

Proposition 1 (Sen24 audit contract). *Suppose that a sen24 artifact bundle contains a CNF instance, its manifest, and either an LRAT trace or a SAT witness model.*

1. *If the sen24 auditor passes on the CNF/manifest pair, then that pair satisfies the audited sen24 artifact contract: schema conformance, variable-domain reconstruction, clause-family accounting, and recorded hash linkage.*
2. *If Lean successfully replays the corresponding LRAT trace for a committed reference CNF instance, then that concrete CNF instance is UNSAT in the Lean kernel.*
3. *If the SAT witness validator succeeds on a reported model/CNF pair, then that concrete model satisfies that concrete CNF instance.*

This proposition does not prove global semantic correctness of the entire encoding pipeline, does not make the atlas as a whole proof-carrying, and still depends on the explicit assumptions and audited surfaces stated above.

Proposition 1 is the paper’s central guarantee statement. Its three clauses are deliberately different in strength: clause (1) is an audit claim about artifacts, clause (2) is a kernel-checked proof claim for committed reference UNSAT instances, and clause (3) is a witness-validation claim for reported SAT instances. None of the three clauses, by itself, proves full semantic equivalence between the Lean axioms and the Python encoding; that bridge remains the explicit assumption boundary, with the short-cycle/acyclicity step justified only by the sen24 finite check above.

Claim	Canonical command	Primary artifact
C1 frontier	<code>python3scripts/run_atlas.py--outdir/ tmp/atlas_c1--jobs4--prunenone python3scripts/summarize_atlas.py--o utdir/tmp/atlas_c1</code>	<code>/tmp/atlas_c1/atlas.json</code>
C2 one MUS + one small MCS	<code>python3scripts/run_atlas.py--outdir/ tmp/atlas_c2--jobs4--prunenone python3scripts/mus_mcs.py--outdir/tm p/atlas_c2</code>	<code>/tmp/atlas_c2/case_*/mus. json, mcs.json</code>
C3 committed kernel-checked UNSAT	<code>python3scripts/run_atlas.py--outdir/ tmp/atlas_c3--jobs1--case-masks31--e mit-proofunsat-only lakebuildSocialChoiceAtlas.Sen.Atlas .Case11111</code>	<code>Certificates/atlas/case_11 111/*</code>
C4 guarded acceleration	<code>python3scripts/run_atlas.py--outdir/ tmp/atlas_c4--jobs1--prunemonotone --prune-check--symmetryalts--symmetr y-check</code>	<code>/tmp/atlas_c4/atlas.json</code>
C5 validated SAT gallery	<code>python3scripts/run_atlas.py--outdir/ tmp/atlas_c5--jobs4--prunenone python3scripts/build_sat_gallery.p y--atlas-outdir/tmp/atlas_c5--top-k 5--min-k1</code>	<code>/tmp/atlas_c5/gallery.json</code>
C6 repair triangulation	<code>python3scripts/run_atlas.py--outdir/ tmp/atlas_c6--jobs4--prunenone python3scripts/enumerate_repairs.p y--outdir/tmp/atlas_c6 python3scripts/triangulate_repairs.p y--atlas-outdir/tmp/atlas_c6--outdi r/tmp/atlas_c6</code>	<code>/tmp/atlas_c6/repair_trian gulation.json</code>

Table 1: Body-level claim-to-artifact-to-command map for the sen24 paper claims.

3.3 Claims, Artifacts, and Commands

Table 1 is the body-level claim map extracted from `docs/paper_claims_map.md`. Its purpose is to keep the paper-level claims tied to one canonical command and one primary artifact per claim.

4 Methods

4.1 Atlas enumeration

We enumerate axiom subsets over a fixed universe and solve each case to obtain SAT/UNSAT status. Outputs include per-case CNF/manifest/summary and top-level `atlas.json`.

4.2 Boundary extraction

For UNSAT solved cases, we run a deletion-based MUS procedure and a small MCS search (single removals first, then bounded pairs). This is intentionally the narrow C2 claim: one MUS and one small MCS candidate are enough to explain the boundary case without claiming full repair enumeration.

Item	Value	Scope
Committed sen24 CNF variables	6936	invariant
Committed sen24 CNF clauses	50114	invariant
Committed LRAT proof size (bytes)	437016	invariant for the committed reference
Committed Lean replay target	available	committed reference only
Full sen24 atlas status	generated (32 cases; 30 SAT, 2 UNSAT)	current full artifact set
Tiny evidence bundle status	available	tiny reviewer path
Full atlas generation time (s)	1.449	environment-specific log value
Paper asset render time (s)	0.098	environment-specific log value
Tiny atlas generation time (s)	0.663	environment-specific log value

Table 2: Minimal quantitative verification facts for sen24. CNF counts and committed LRAT size are invariant artifact properties. Timing rows, when present, are local log values from the current generated bundles and are environment-specific measurements rather than benchmark claims.

4.3 Repair enumeration and triangulation

When we want the stronger C6 evidence, we run a separate post-processing stage with `scripts/enumerate_repairs.py` and `scripts/triangulate_repairs.py`. The paper keeps this distinct from C2 so that repair *enumeration* is not confused with boundary *explanation*.

4.4 Proof-carrying UNSAT policy

For canonical UNSAT references, we retain CNF+LRAT and verify in Lean. For atlas runs, proof metadata is recorded with hash-based linkage and replay commands.

4.5 sen24 acyclicity boundary

The CNF does not encode Lean’s `Acyclic` predicate directly. Instead, for sen24 it forbids directed 3-cycles and 4-cycles. We therefore add one explicit finite check to the method: `scripts/check_acyclicity_short_cycles.py` exhaustively enumerates the four-alternative relation space and verifies that, under asymmetry, this short-cycle encoding is equivalent to acyclicity. This is a sen24-specific audited fact, not a scalability claim.

4.6 SAT Gallery with witness validation

SAT cases are filtered by lightweight non-trivial heuristics and ranked deterministically. Each selected witness is validated against CNF using a solver-independent checker that parses DIMACS and model assignments directly.

5 Verification Status and Evidence Accounting

This section is organized as a claim-by-claim evidence checklist rather than a fresh benchmark report. The goal of the current revision is to state, for each official atlas claim C1–C6, which artifact supports it and what was actually re-verified now. The runtime harness in `scripts/eval_atlas.py` was re-run at smoke scope inside `./scripts/ci_phase2_smoke.sh`, but it is used here only as a sanity check on metadata and plotting, not as a new performance claim. The sen24 short-cycle/acyclicity bridge is tracked separately as an audited finite Python check, not as a Lean-side proof obligation under C1–C6.

Table 2 supplies the minimal quantitative context used in this paper: fixed committed artifact sizes, current full/tiny artifact status, and optional local timing values from generated logs. These rows are evidence bookkeeping, not portable benchmarks.

C1: reproducible frontier generation. Primary support comes from `scripts/run_atlas.py`, `atlas.json`, `atlas_summary.md`, and per-case `summary.json`. This revision re-verified C1 through `./scripts/ci_phase2_smoke.sh`, which re-runs a full 32-case `sen24` atlas in the repair smoke path and checks the resulting frontier artifacts.

C2: one MUS and one small MCS candidate. Primary support comes from `script s/mus_mcs.py` and the embedded `mus/mcs` fields in `atlas.json`. This revision re-verified the smoke-scope version of C2 through `./scripts/ci_phase2_smoke.sh`, which regenerates `mus.json` and `mcs.json` for the committed UNSAT boundary case `case_11111`.

C3: committed proof-carrying UNSAT. Primary support comes from the committed reference artifacts under `Certificates/atlas/case_11111/`, the Lean target `SocialChoice Atlas.Sen.Atlas.Case11111`, and proof metadata in regenerated summaries. This revision re-verified C3 through both `./scripts/ci_phase1.sh` and `./scripts/ci_phase2_smoke.sh`, which rebuild Lean proof targets and check the committed proof-carrying case.

C4: guarded symmetry reduction and monotone pruning. Primary support comes from `docs/assumptions_monotone_pruning.md`, `docs/safety_symmetry_reduction.md`, and the runtime guardrail fields in `atlas.json`. This revision re-verified C4 through `./scripts/ci_phase2_smoke.sh`, which executes both symmetry-checked and prune-checked atlas runs and validates their metadata.

C5: auditable SAT gallery. Primary support comes from `scripts/build_sat_gallery.py`, `gallery.json`, `gallery.md`, per-case `rule_card.md/rule_card.tex`, and embedded SAT witness-validation statistics. This revision re-verified C5 through `./scripts/ci_phase2_smoke.sh`, which rebuilds a small gallery, checks that selected entries are non-trivial and validated, and confirms that rule-card artifacts exist.

C6: repair triangulation against an independent optimum baseline. Primary support comes from `scripts/enumerate_repairs.py`, `scripts/triangulate_repairs.py`, `repair_triangulation.json`, and the comparison fields `size_match/set_match`. This revision re-verified C6 through `./scripts/ci_phase2_smoke.sh`, which re-runs full repair enumeration and triangulation on a solved 32-case `sen24` atlas.

6 Sen24 Case Study

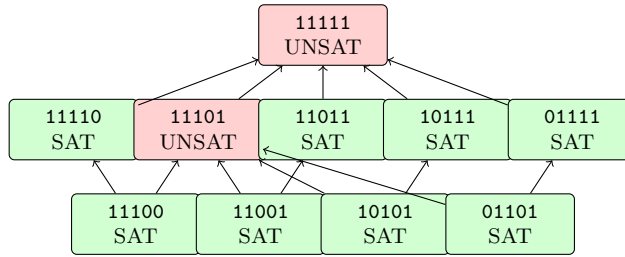
The `sen24` atlas is the fixed case study where the paper’s central argument becomes concrete. It turns the abstract claim ladder into one connected line of evidence: frontier first, then local boundary explanation, then validated SAT witnesses, then the stronger repair-triangulation stage, all under the audit contract of Proposition 1. The atlas supplies two different but complementary evidence paths:

- **Semantic path.** `SocialChoiceAtlas/Sen/BaseCase24/Theorem.lean` proves the `sen24` impossibility statement in Lean by constructing witness profiles that force 3-cycles or 4-cycles.
- **Artifact path.** `SocialChoiceAtlas/Sen/BaseCase24/SATSenCNF.lean` replays an LRAT certificate for the committed DIMACS baseline, proving UNSAT of the audited CNF instance in the Lean kernel.

00000	00001	00010	00011	00100	00101	00110	00111
SAT	SAT	SAT	SAT	SAT	SAT	SAT	SAT
01000	01001	01010	01011	01100	01101	01110	01111
SAT	SAT	SAT	SAT	SAT	SAT	SAT	SAT
10000	10001	10010	10011	10100	10101	10110	10111
SAT	SAT	SAT	SAT	SAT	SAT	SAT	SAT
11000	11001	11010	11011	11100	11101	11110	11111
SAT	SAT	SAT	SAT	SAT	UNSAT	SAT	UNSAT

Legend: SAT UNSAT PRUNED UNKNOWN

Figure 2: Full 32-case sen24 frontier generated from `atlas.json`. Each cell records one axiom-lever combination and its SAT/UNSAT status. This figure supports the official C1 frontier claim; the statuses themselves are atlas artifacts, while proof-carrying replay is attached separately only to committed UNSAT references.



Boundary slice: unsatisfiable cases and their immediate solved predecessors in the subset poset.

Figure 3: Boundary slice of the sen24 subset poset. The figure shows the two UNSAT cases together with their immediate solved predecessors, making the local frontier geometry visible without implying that the whole atlas is kernel-checked. Proof-carrying replay remains attached only to committed reference artifacts.

These paths are intentionally linked but not conflated. The semantic theorem is a statement about social-choice axioms. The LRAT replay is a statement about one concrete CNF artifact. The paper’s chain of trust is the argument that these two forms of evidence are compatible under an explicit audit contract. Proposition 1 is the formal summary of that contract.

Acyclicity bridge for four alternatives. The main sen24-specific semantic gap is that the CNF uses `NO_CYCLE3/NO_CYCLE4`, while the Lean side uses `Acyclic`. We do not leave this implicit. The repository includes `scripts/check_acyclicity_short_cycles.py`, which exhaustively checks all relations on the four-alternative domain and verifies that, under asymmetry, acyclicity is equivalent to the absence of directed 3-cycles and 4-cycles. This justifies the sen24 encoding choice without overclaiming a general theorem for arbitrary numbers of alternatives.

Frontier structure (C1). The case study cites generated assets fixed under `paper/figures/generated/` and `paper/tables/generated/`. Figure 2 shows the full sen24 SAT/UNSAT frontier, Figure 3 shows the boundary slice of the subset-poset/Hasse structure, and Table 3 supports the narrower C2 boundary explanation.

For UNSAT verification, the committed reference case uses LRAT checked by Lean as specified in Proposition 1. For SAT examples, the gallery records deterministic ranking and witness-validation statistics for each selected case.

UNSAT boundary explanation (C2). The official C2 claim remains narrow: one MUS and one small MCS candidate per boundary case. In the paper, that claim is supported by the per-case `mus.json / mcs.json` artifacts and summarized in Table 3; it is not stated as full repair enumeration.

Case bits	Status	One MUS	Small MCS candidate
11101	UNSAT	11101 (size 4)	drop <code>asymm</code> → 01101 (size 1)
11111	UNSAT	11101 (size 4)	drop <code>asymm</code> → 01111 (size 1)

Table 3: Local `sen24` boundary explanation for the official C2 claim. Each row records one UNSAT boundary case together with one MUS and one small MCS candidate extracted from the per-case artifacts.

Case	Distinct	DictMax	ConstRate	Validated	Rule Card
case_01000	219	0.7500	0.0000	true	case_01000/rule_card.tex
case_01001	219	0.7500	0.0000	true	case_01001/rule_card.tex
case_01010	219	0.7500	0.0000	true	case_01010/rule_card.tex
case_01011	219	0.7500	0.0000	true	case_01011/rule_card.tex
case_11000	219	0.7500	0.0000	true	case_11000/rule_card.tex

Table 4: Selected SAT gallery entries with validation metrics.

Validated non-trivial SAT rule cards. Each selected SAT gallery entry is accompanied by `rule_card.md` and `rule_card.tex`, including non-triviality metrics and concrete profile witnesses. SAT assignments are checked with `scripts/validate_sat_witness.py` before rule cards are emitted. Table 4 is the frozen paper-facing summary supporting the validated SAT-example part of the argument, and the selected rule-card export is fixed at `tables/generated/selected_rule_card.tex`.

Repair enumeration and triangulation (C6). The stronger C6 stage begins only after the narrower C2 boundary explanation: a fully solved atlas run enumerates `mcs_all`, `mcs_min_size`, and `mcs_min_all`, then compares those outputs against an independent optimum baseline by triangulation. The paper freezes these outputs as `repairs_table.tex` and `triangulation_table.tex`; the first reports the stronger repair enumeration output and the second reports the independent triangulation check.

Triangulation and poset structure artifacts. The optimization-side triangulation report is generated as `repair_triangulation.md` via `scripts/triangulate_repairs.py`, comparing independent optimum repair sizes against `mcs_min_size/mcs_min_all`. The frontier structure is rendered as a subset-poset/Hasse graph (`frontier_hasse.dot` and `frontier_hasse.png`) via `scripts/plot_hasse_frontier.py`.

7 Related Work and Positioning

The closest prior context is SAT-based computational social choice. That line already showed that SAT solving can be used to prove impossibility results, search for axiomatizations, and discover proofs in small finite domains; it includes impossibility proving by Tang and Lin [7], later automated exploration and proof discovery by Geist and Endriss [5], and later SAT-based synthesis work by Brandt and Geist [3]. Our paper does not claim novelty for “using SAT in social choice,” for extracting contradictions, or for small-scope automated theorem search by itself. Its narrower contribution is to package one fixed `sen24` search workflow as an explicit auditable contract whose outputs are replayable artifacts rather than solver conclusions alone.

This difference is about contract structure, not theorem content. Prior COMSOC-SAT work is the direct intellectual predecessor for boundary search and explanation, and we build squarely on that perspective. What we add is a paper-level separation between proof, audit, and assumption: semantic Lean theorems on one side, manifest-audited CNF artifacts in the middle,

Case	MUS	MCS _{min}	Count(MCS _{min})	Orbit	Solved
case_11101	4	1	4	1	true
case_11111	4	1	4	1	true

Table 5: UNSAT repair summary from atlas outputs.

Case	OptDrop	MCS _{min}	Baseline _{min}	SizeMatch	SetMatch
case_11101	1	1	–	true	true
case_11111	1	1	1	true	true

Table 6: Repair triangulation with a tiny MAXSAT-style sanity baseline (minimum repair size).

committed LRAT replay for reference UNSAT cases, and independently validated SAT witnesses for displayed feasible rules. In that sense, the paper is closer to certificate-backed automated reasoning than to a new impossibility proof technique.

Certificate checking and proof-carrying automated reasoning form the second comparison point. We differ from that tradition not by stronger proof checking, but by the way it is embedded into a social-choice workflow. The central artifact is not a bare proof trace: it is an auditable atlas with a reproducible frontier, one MUS and one small MCS candidate for boundary explanation, committed proof-carrying UNSAT references, validated SAT rule cards, stronger repair triangulation as a separate stage, and explicit commands and hashes for replay. The value of the package is that a reviewer can inspect where kernel-checked proof ends, where mechanical auditing begins, and where assumptions remain.

Constraint Hierarchies, ATMS, MAXSAT, and OMT remain relevant only as secondary contrasts [2, 4, 1, 6]. They reason about conflicting constraints, consistency under assumptions, or optimal relaxations, and they are useful reference points for the repair side of the story. But this paper does not position itself as a replacement for them or as a speed comparison against them. Their role here is narrower: optimization-style reasoning appears as the independent triangulation baseline for C6, whereas the main contribution remains auditable boundary mapping, conservative MUS/MCS explanation, and replayable SAT/UNSAT evidence artifacts.

The scope is intentionally narrow. We claim only a sen24 case study under a small-scope hypothesis, not a general account of arbitrary n, m , not completeness for multiple theorem families, and not formal Lean verification of symmetry or pruning heuristics. The point is that even under those limits, the paper makes one JAR-relevant contribution precise: a small social-choice impossibility search can be presented as an auditable reasoning workflow with a visible trust contract.

8 Limitations and Scope

This paper intentionally targets a killable instance (sen24: two voters, four alternatives). We do not claim direct scalability to arbitrary n, m , and we do not claim an end-to-end mechanized encoding-correctness proof for the Python generator.

The semantic scope is also fixed:

- the domain is sen24 only;
- Universal Domain is represented by linear orders on four alternatives;
- the NO_CYCLE3/NO_CYCLE4 encoding is justified only through the sen24 finite Python check under asymmetry, not as a general replacement for `Acyclic` and not as a Lean proof of encoding correctness.

Safe acceleration features (monotone pruning and symmetry reduction) are only valid under explicit assumptions and runtime guards. These assumptions are specified in:

- `docs/assumptions_monotone_pruning.md`
- `docs/safety_symmetry_reduction.md`

Future work should expand scale only if the same proof/audit/assumption split remains explicit.

9 Conclusion

We provide a small-scope auditable possibility atlas for computational social choice. In `sen24`, the main result is not a new impossibility theorem but a reproducible map of which axiom combinations are feasible or infeasible, together with evidence at explicitly separated trust levels. The frontier itself is recorded as replayable artifacts, local UNSAT explanation remains limited to one MUS plus one small MCS candidate, relaxed SAT cases are turned into validated rule examples, and stronger repair reasoning appears separately through full enumeration plus triangulation.

The `sen24` case study also suggests a methodological lesson. For this kind of automated impossibility search, frontier discovery, local explanation, SAT witness validation, stronger repair analysis, and kernel-checked proof replay should not be collapsed into a single undifferentiated notion of “verification.” They support different claims and belong to different parts of the trust boundary. Proposition 1 makes that split explicit: committed UNSAT reference artifacts are kernel-checked from LRAT, SAT examples are witness-validated against CNF, and the remaining encoder boundary is exposed as an audit contract rather than hidden inside solver trust.

Any later expansion of the atlas should preserve the same trust and reproducibility discipline.

A Reproducibility Appendix

A.1 Canonical repository

The canonical public repository for this artifact is: <https://github.com/SHayashida/Sen-Lean4>

A.2 Claim-to-artifact map

Claims C1–C6 are mapped to concrete commands and artifacts in `docs/paper_claims_map.md`.

- **C1**: atlas frontier generation.
- **C2**: one MUS and one small MCS candidate for boundary explanation.
- **C3**: committed proof-carrying UNSAT verification in Lean.
- **C4**: safety-guarded acceleration (symmetry/pruning).
- **C5**: SAT Gallery extraction with witness validation.
- **C6**: full repair enumeration plus triangulation against an independent optimum baseline.

Claim	Canonical command	Primary artifact
C1	<code>python3scripts/run_atlas.py--outdir/tmp/atlas_c1--jobs4--prunenone</code> <code>python3scripts/summarize_atlas.py--outdir/tmp/atlas_c1</code>	<code>/tmp/atlas_c1/atlas.json</code>
C2	<code>python3scripts/run_atlas.py--outdir/tmp/atlas_c2--jobs4--prunenone</code> <code>python3scripts/mus_mcs.py--outdir/tmp/atlas_c2</code>	<code>/tmp/atlas_c2/case_*/mus.json, mcs.json</code>
C3	<code>python3scripts/run_atlas.py--outdir/tmp/atlas_c3--jobs1--case-masks31--emit-proofunsat-only</code> <code>lakebuildSocialChoiceAtlas.Sen.Atlas.Case11111</code>	<code>Certificates/atlas/case_11111/proof.lrat</code>
C4	<code>python3scripts/run_atlas.py--outdir/tmp/atlas_c4--jobs1--prunemonotone--prune-check--symmetryalts--symmetry-check</code>	<code>/tmp/atlas_c4/atlas.json</code>
C5	<code>python3scripts/run_atlas.py--outdir/tmp/atlas_c5--jobs4--prunenone</code> <code>python3scripts/build_sat_gallery.py--atlas-outdir/tmp/atlas_c5--top-k5--min-k1</code>	<code>/tmp/atlas_c5/gallery.json</code>
C6	<code>python3scripts/run_atlas.py--outdir/tmp/atlas_c6--jobs4--prunenone</code> <code>python3scripts/enumerate_repairs.py--outdir/tmp/atlas_c6</code> <code>python3scripts/triangulate_repairs.py--atlas-outdir/tmp/atlas_c6--outdir/tmp/atlas_c6</code>	<code>/tmp/atlas_c6/repair_triangulation.json</code>

Table 7: Compact C1–C6 claim map with one canonical command per claim.

A.3 Verification status in this revision

In Table 8, “Repo-capable” means the repository contains the machinery and documented artifacts for the check. “Re-verified now” means the command was actually re-run in this revision. For the `sen24` bridge used in the paper text, the acyclicity check below is an audited finite Python check outside the trusted kernel; it is not counted as a Lean proof obligation.

For the `sen24` acyclicity bridge used in the paper text, run: `python3scripts/check_acyclicity_short_cycles.py--json-outresults/verification/sen24_short_cycles.json`.

A.4 Schema and metadata contracts

Reproducibility relies on versioned machine-readable outputs:

- `atlas_schema_version`
- `eval_schema_version`
- `solver/environment` metadata and seed policy fields

See `docs/reproducibility_appendix.md` for policy details.

A.5 Baseline protection

Protected baseline artifacts remain unchanged and must be treated as immutable references in normal development.

References

- [1] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maximum satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*,

Check / artifact	Repo-capable	Re-verified now	Command	Expected artifact(s)
sen24 acyclicity finite check	Yes	Yes	<code>python3scripts/check_acyclicity_short_cycles.py --json-outresults/verification/sen24_short_cycles.json</code>	<code>results/verification/sen24_short_cycles.json</code>
Phase1 baseline audit + Lean replay	Yes	Yes	<code>./scripts/ci_phase1.sh</code>	phase1 CNF audit outputs; Lean build for SATSenCNF
Paper PDF build	Yes	Yes	<code>make-Cpaperpdf</code>	<code>paper/build/main.pdf</code>
Phase2 smoke harness	Yes	Yes	<code>./scripts/ci_phase2_smoke.sh</code>	smoke atlas, symmetry/prune checks, repair outputs, gallery outputs, bundle outputs
Tiny evidence bundle	Yes	Yes	<code>./scripts/ci_phase2_smoke.sh</code>	<code>bundle.json</code> ; generated figures and paper tables in the tiny bundle
Full sen24 atlas (<code>--prunenone</code>)	Yes	Yes	<code>./scripts/ci_phase2_smoke.sh</code>	32-case <code>atlas.json</code> used for repair and gallery smoke checks
Full repair enumeration + triangulation	Yes	Yes	<code>./scripts/ci_phase2_smoke.sh</code>	<code>mcs_all</code> ; <code>repair_triangulation.json</code> ; <code>repair_triangulation.md</code>
SAT gallery + rule cards	Yes	Yes	<code>./scripts/ci_phase2_smoke.sh</code>	<code>gallery.json</code> ; <code>gallery.md</code> ; per-case <code>rule_card.md/.tex</code>
Committed Lean proof case <code>case_11111</code>	Yes	Yes	<code>./scripts/ci_phase2_smoke.sh</code>	committed proof-carrying case plus <code>lakebuildSocialChoiceAtlas.Sen.Atlas.Case11111</code>

Table 8: Verification accounting for this revision. Status is marked “Yes” only when the command was actually re-run now.

volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 929–991. IOS Press, 2021. doi: 10.3233/FAIA201008.

- [2] Alan Borning, Bjørn N. Freeman-Benson, and Molly Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5(3):223–270, 1992. doi: 10.1007/BF01807506.
- [3] Felix Brandt and Christian Geist. Finding strategyproof social choice functions via SAT solving. *Journal of Artificial Intelligence Research*, 55:565–602, 2016. doi: 10.1613/jair.4959.
- [4] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28(2):127–162, 1986. doi: 10.1016/0004-3702(86)90080-9.
- [5] Christian Geist and Ulle Endriss. Automated search for impossibility theorems in social choice theory: Ranking sets of objects. *Journal of Artificial Intelligence Research*, 40:143–174, 2011. doi: 10.1613/jair.3126.
- [6] Roberto Sebastiani and Silvia Tomasi. Optimization modulo theories with linear rational costs. *ACM Transactions on Computational Logic*, 16(2):12:1–12:43, 2015. doi: 10.1145/2699915.
- [7] Pingzhong Tang and Fangzhen Lin. Computer-aided proofs of Arrow’s and other impossibility theorems. *Artificial Intelligence*, 173(11):1041–1053, 2009. doi: 10.1016/j.artint.2009.02.005.