A Lean-Verified Machine and Encoding Bridge for OpenAI Math 378

Formalize.now Technical note, version 2

November 1, 2026

Abstract

We document a kernel-checked Lean 4.34.1 implication from OpenAI's formal theorem on deterministic bounded-width SAT algorithms to the negation of the `PositiveRateETH` predicate in OpenAI's separate ParityLifts development. The bridge proves an exact conversion between the relevant CNF encodings, a step-preserving translation of multitape machine configurations, a fixed executable simulation with polynomial overhead, and the asymptotic transfer of OpenAI's subexponential running-time bound. The resulting declaration is `OAI.Bridges.ETH.not_positiveRateETH`, whose type is ¬ `OAI.ParityWL.Complexity.PositiveRateETH`. A separate VPS build completed 11,600 jobs with exit code zero and reported only the axioms `propext`, `Classical.choice`, and `Quot.sound`. We give the scope, dependency structure, reproducibility record, and an agenda for comparison with the Anthropic ETH formalization.

1 Statement of the result

The final declaration in `OAI/Bridges/ETH/FinalImplication.lean` is:

```
theorem not_positiveRateETH :
¬ OAI.ParityWL.Complexity.PositiveRateETH :=
not_positiveRateETH_of_mainTheorem
OAI.Algorithms.SubexponentialSAT.SqrtLogSpace.original_sat_main
```

The target predicate is defined using a deterministic finite-alphabet multitape machine M, CNF instances $F : \mathrm{Formula}(N)$ whose clauses have at most three literals (repeated clauses allowed), and a worst-case execution-time relation SATTimeBound:

$$\mathrm{PositiveRateETH} := \exists \delta > 0,\ \neg\exists M, C, d\ (C > 0\ \wedge\ \mathrm{SATTimeBound}(M,\ (N, L) \mapsto C(L+1)^d 2^{\delta N})). \tag{1}$$

Here N denotes the declared variable count and L the binary encoding length. Thus its negation supplies, for each $\delta > 0$, an appropriate fixed machine and polynomial factor valid uniformly over inputs. The quantifier order is the positive-rate form associated with $s_3 > 0$ in Impagliazzo–Paturi [5]. To see why, let $A(\delta)$ assert that a uniform deterministic decider with some polynomial input-length factor meets the $2^{\delta N}$ rate. For $0 < \delta_1 \le \delta_2$, $A(\delta_1)$ implies $A(\delta_2)$, using the same machine and constants. Hence an unattainable positive rate is accompanied by unattainability at every smaller positive rate. Negating Eq. (1) gives $\forall \delta > 0,\ A(\delta)$: the machine may depend on δ, while each selected machine works on all encoded instances. The interpretation of the specific machine and encoding conventions is addressed in Section 5.

2 Source theorem and complexity bound

OpenAI's source theorem, `SqrtLogSpace.original_sat_main`, has type `SqrtLog.Statement.MainTheorem` [1]. For every fixed clause width $k \geq 3$, `Feasible k` holds. The latter asserts a primitive multitape machine deciding satisfiability of width-k CNF inputs under OpenAI's canonical encoding, on every such input, with no promise or large-input qualification, and with positive constants C, A and positive degree d that depend on k only.

The Lean-verified runtime uses

$$T(n, L) \leq C(L+1)^d 2^{A\,\rho(n)}, \qquad \rho(n) = \frac{n\,[\log\log(n+16)]^2}{\sqrt{\log(n+16)}}. \tag{2}$$

where n is the variable count of the OpenAI instance and L is the length of its encoding `Encoding.encodeCNF`. This is subexponential in n up to the polynomial input-length factor since $\rho(n)/n \to 0$. The associated manuscript discusses a sharper exponent with one rather than two powers of $\log\log n$; the bridge uses the Lean theorem in Eq. (2).

3 Construction of the bridge

3.1 Exact encoding conversion

The ParityLifts formula representation allows lists of clauses over `Fin N`, each with at most three literals. The conversion constructs a canonical CNF word and establishes both (i) satisfiability preservation and (ii) bit-for-bit equality with OpenAI's `Encoding.encodeCNF` (`canonicalWord_eq`). The executable `canonicalWordProgram` runs on the normalized bit form `minimizeList F.bitList` that ParityLifts' pipeline supplies, and `canonicalWordProgram_time_poly` bounds it by

$$\exists c, d \in \mathbb{N}\ \forall N\ \forall F : \mathrm{Formula}(N), \quad \mathrm{time}_{\mathrm{conv}}\big(\mathrm{minimizeList}(F.\mathrm{bitList})\big) \leq c(N+1)^d, \tag{3}$$

where $\mathrm{time}_{\mathrm{conv}}$ is the step count of `canonicalWordProgram`. The separate `ConverterInterface` module proves `no_polyProgram_canonicalWord`: no `PolyProgram`, whose cost is polynomial in the serialized input size alone, computes `canonicalWord`, because variable names written in binary become values whose output length grows with their magnitude. The bridge therefore uses a `TreeProgram` whose time is polynomial in the declared variable count N, which the target rate $2^{\delta N}$ absorbs. This module lies outside the final theorem's import closure.

3.2 Machine translation and executable simulation

`MachineTranslation.lean` represents source tape cells `none`, `some false`, and `some true` by three symbols, reindexes tapes, and proves a correspondence of initial configurations and individual transition steps. A source-machine halting trace within s steps produces an output trace in the target machine model (`emits_of_outputsInTime`).

`MachineRunner.lean` builds a single target program `primitiveRunner M` that probes increasing step budgets, distinguishes a completed false answer from a still-running computation, and returns the eventual Boolean answer. `MachineRunnerCost.lean` then establishes

$$\begin{aligned} \forall M\ \exists c, e \in \mathbb{N}\ \forall w, b, s, \quad & M.\mathrm{OutputsInTime}(w, b, s) \Longrightarrow \\ & \mathrm{RBEval}\big(\mathrm{primitiveRunner}(M), \mathrm{treeEncode}(w), \\ & \quad c(s + |w| + 1)^e, \mathrm{treeEncode}(b)\big). \end{aligned} \tag{4}$$

This is `primitiveRunner_poly`, stated as `PrimitiveSimulation M (primitiveRunner M) c e`. The constants c, e depend on M only. The declaration `simulationHypothesis_proved` closes the previously conditional simulation interface.

3.3 Exponential-rate transfer

The rate argument isolates two effects. The conversion and simulation introduce polynomial factors in N and the source running time. Separately, OpenAI's exponential factor $2^{A\rho(N)}$ satisfies, for every $r > 0$, a bound of the form $K(N+1)^D 2^{rN}$, by `runtimeFactor_expBound`; this also covers the e-th power introduced by the simulation, since $(2^{A\rho(N)})^e = 2^{eA\rho(N)}$. The composition and SAT decoding lemmas combine the bounds at any chosen positive rate δ, producing the negation of Eq. (1). The machine is constructed independently of the instance and its size. The verified final theorem uses the precise existential quantifiers stated in Eq. (1).

4 Verification and reproducibility

Item	Recorded value
Upstream revision	`7ef5c0cc5ca4b0e072b7a050fca09ab24bc6fa3a`
Verified C2 snapshot	`81ec97cd23d21675cee4ee76768fffba33a61e17`
Documentation C3 commit	`81708f93f` (VPS abbreviated hash)
C2 tree hash	`1637f35b64e7f9f5d8f3c4e44e28007812a74f72`
Toolchain	Lean 4.34.1; pinned Mathlib manifest
Build target	`OAI.Bridges.ETH.FinalImplication`
Independent VPS result	11,600 jobs; exit status 0
Final axiom dependencies	`propext`; `Classical.choice`; `Quot.sound`

To reproduce from the pinned project under `lean/`, run:

```
LEAN_NUM_THREADS=2 lake build OAI.Bridges.ETH.FinalImplication
```

The module includes `#print axioms OAI.Bridges.ETH.not_positiveRateETH`. Its axiom audit tests the complete proof dependencies, including the OpenAI theorem imported as a proof term. The original OpenAI sources and Comparator settings remain unchanged by the bridge. The companion reproducibility guide [2] specifies the other modules, a source-integrity scan, and optional Comparator checks. OpenAI's Comparator configuration checks `exact_count_and_sat`; `original_sat_main` is derived from it within OpenAI's sources and is checked by the Lean kernel during this build. Commit hashes differ between the authoring environment and the VPS because patches were applied with `git am`; the tree hash is the environment-independent identifier.

5 Formal scope and related work

The conclusion's meaning is determined by `PositiveRateETH` and its dependencies: `SATTimeBound`, machine configurations and transitions, `Emits`, the definition of 3-CNF formulas, and `encodeFormula`. OpenAI's `MainTheorem` enters as an already kernel-checked proof term; it is absent from the final theorem's type. In our reading of these definitions, the predicate states the $s_3 > 0$ deterministic

3-SAT formulation [5] with the same quantifier order, and each of its conventions asks at least as much of the refuting algorithm as the standard one: a deterministic finite-alphabet multitape Turing machine, worst-case step counts to an absorbing output state, clauses of width at most three, and an exponent measured in the declared variable count N with a polynomial factor in the encoding length. The monotonicity of admissible exponential rates explains the infimum-based formulation. Equivalence across machine models and specific standard input encodings remains a mathematical interpretation of the formal specifications, separate from the kernel check.

Anthropic's *formal-math/eth* [3] is a related Lean formalization with a separately defined `ETHIsFalse` target and a different machine model and toolchain. An independent cross-formalization bridge has also been discussed in Anthropic's public issue tracker [4]. These provide natural comparison points for the precise theorem types and implementation assumptions. Our result targets ParityLifts' `PositiveRateETH` within the pinned OpenAI checkout; a direct Lean-level equivalence with Anthropic's statement is an additional research task.

A further target is the separately formalized `UniformSubexponentialETH`, in which one machine with one polynomial degree must satisfy every positive-rate bound. The existing construction suggests a route to that stronger result, provided the complete decider and its polynomial degree can be fixed before the rate is selected; the theorem documented here is the checked `not_positiveRateETH` declaration.

6 Availability and next checks

The proof and C3 report are public on the branch `thinkthoughts/math:eth-bridge/component-B`. See `ETH_BRIDGE_REPORT.md`, `ETH_BRIDGE_REPRODUCE.md`, and `ETH_BRIDGE_MANIFEST.md`. The next independent reviews are: reproducibility with a second kernel/checker, semantic fidelity of the ETH predicates, and a theorem-level comparison with Anthropic's formalization.

References

[1] OpenAI, *math*, pinned revision `7ef5c0c`, https://github.com/openai/math.

[2] *ETH bridge verification and reproducibility documentation*, https://github.com/thinkthoughts/math/tree/eth-bridge/component-B/docs.

[3] Anthropic, *formal-math*, ETH development, https://github.com/anthropics/formal-math/tree/main/eth.

[4] Anthropic formal-math issue tracker, cross-formalization discussion, https://github.com/anthropics/formal-math/issues/36.

[5] R. Impagliazzo and R. Paturi, *On the Complexity of k-SAT*, Journal of Computer and System Sciences 62 (2001), 367–375.