PNP Formal Reconstruction Report

Compiled Lean theorem inventory and current claim boundary

Coordinate `PNP-FORMAL-RECONSTRUCTION-STATUS-2026-08-12-133`

The repository does not currently establish $P = NP$.

Public theorem emission is disabled. A direct finite raw-machine verifier now proves concrete CNF-SAT membership in NP with an explicit polynomial bound. It does not prove CNF-SAT in P, NP-hardness, NP-completeness, or $P = NP$. The general charged pipeline is linked to the raw machine kernel, and the Cook–Levin formula has an exact externally sized answer-independent slot schedule plus direct fuelled bit and token specification cursors. A literal finite builder emits `FormulaWidth` copies of `T` followed by `F`, `Sep`, and the complete positive clause on variables zero, one, and two. It then executes the first padding transition and the entire remaining first-clause padding run, then emits the complete second clause `Sep F F F T F Finish`, traverses the entire remaining clause-two padding block, and then emits the complete third clause `Sep F F F T T F Finish` on variables zero and two, traverses the entire remaining clause-three padding block, and emits the complete fourth clause `Sep F T F F T T F Finish` on variables one and two, traverses the remainder of the first scheduled constraint, emits the second constraint's first positive literal through its width-selected successor, and consumes the next seven width-dependent opportunities. At width one those opportunities are padding and emit nothing; at wider widths they emit the first four unary `T` tokens and terminating `F` of the second literal, followed by the next literal's opening `T` and first unary-index `T`. Every traversed padding slot and emitted populated slot has a direct schedule proof, the emitted bits are the exact canonical prefix, and an external polynomial bounds the compiled run. The following padding-or-terminating-`F` opportunity, a general dynamic formula cursor, arbitrary raw slot decoder, and the remaining formula body are not implemented, so there is no complete raw polynomial-time Cook–Levin formula builder. A separate fixed all-input machine now translates strict canonical CNF into well-formed topological NAND, preserves satisfiability exactly, supplies a polynomial-time function and raw refinement, and packages the direct and locked-NAND-composed polynomial reductions. The report-facing theorem now publishes their all-bitstring concrete locked-NAND reduction. It does not decide CNF-SAT, put the locked target in P, or discharge residual-band/ZeroSlack/PCCMin. The reviewed activation fingerprints are unset, no root theorem exists, and the abstract string-handle bridge is never an activation source.

Compiled declarations	27794
Theorem-kind declarations	14454
Assumption-free theorem-kind declarations	7347
Project-specific axioms	4
Concrete publication gate passed	**false**

Inventory SHA-256: `696c76220a092e5a84e7caa804fd1c57889f193968d1285b520c408f8237f5c1`

Generated from the pinned compiled Lean environment, not from source-text parsing.

Contents

1 Current authority and claim boundary

This report is the current repository report surface. Its factual theorem inventory comes from the compiled `PNP` environment under `leanprover/lean4:v4.31.0`. The exporter enumerates `Environment.constants`, resolves each originating module, and calls `Lean.collectAxioms` for every exported project declaration. The committed inventory is `status/LEAN_THEOREM_INVENTORY.json`; the public mirror is byte-identical.

The target remains $P = NP$, but target wording—including the new inactive definition `PNP.Main.ConcretePEqualsNP`—is not theorem evidence. JavaScript acceptance, Boolean fields, JSON strings, historical release records, report prose, and external review cannot activate a theorem. Only the separately derived concrete publication gate may control theorem emission, and that gate is currently false.

The completed direct CNF verifier consumes the literal canonical pair of an encoded formula and assignment certificate. Lean proves universal acceptance equivalence, rejection equivalence, and absence of timeout at its explicit polynomial fuel bound, and constructs a proof-bearing `PolynomialTimeVerifier CNFSAT`. Thus `PNP.Concrete.CNFSAT` is in the concrete NP class. This is a verifier theorem, not a deterministic polynomial-time SAT algorithm.

The concrete Cook–Levin development proves that the generated CNF formula is semantically equivalent to ordinary raw verifier execution in both input modes. It now also bounds the actual canonical unary-indexed formula encoding by an explicit fixed-verifier polynomial evaluated only at external source-input length. That output-size theorem does not execute the formula builder as a raw finite machine, prove its construction runtime, or package a concrete polynomial reduction.

The concrete CNF-to-NAND development traverses any decoded formula structurally and constructs an intrinsically topological NAND circuit without querying satisfiability. Lean proves exact valuation and satisfiability semantics, strict canonical decoder inversion, well-formed output bytes, the exact gate count, a quadratic serialized-output bound in external input length, and fail-closed equivalence on every bitstring. It also composes semantically with the concrete locked-NAND threshold builder. A subsequent fixed parser/carrier/controller work graph now implements exactly that pure compiler on every bitstring within one external-input polynomial. Its compiled machine never times out at the advertised bound, retains literal `RawRefinement`, packages `PolynomialReduction CNFSAT EncodedNANDSAT`, and explicitly composes with the strict locked-NAND reduction. The report-facing theorem `PNP.Main.locked_nand_threshold` now publishes that composition as a uniform all-bitstring reduction from `CNFSAT` to `EncodedLockedNANDThreshold`; its closure uses only `propext` and `Quot.sound`. This reduction still does not decide CNF-SAT, put the locked target in P, discharge residual-band/ZeroSlack/PCCMin, or prove $P = NP$.

The new rectangular schedule allocates exact constraint, clause, token, and raw-bit opportunities without reading the already-materialized program, formula, token encoding, or encoded formula as schedule inputs. Filtering populated slots reproduces those canonical objects, and the raw-bit slot count is exactly the external encoded-size polynomial. This remains a pure specification: no theorem charges constant time per slot, implements a raw builder, or proves a construction-runtime bound.

The direct formula cursor decodes constraint, clause, token, and bit coordinates without first constructing those complete canonical lists. Its nested options distinguish out-of-range lookup from valid empty padding, and Lean proves exact prefix, full, one-step-short, terminal, and excess-fuel behavior. Filtering the exact full cursor output yields the canonical encoding. These are specification-level evaluation theorems, not a constant-time raw slot interpreter or raw builder.

The first literal builder stage is a fixed 19-rule work machine. Starting at the proved all-input framer endpoint, it restores every source bit and appends exactly one unary tally symbol per bit in fresh right-side workspace. Its exact work cost is $2n^2 + 4n + 2$, and six-for-one compilation gives the exact raw cost $12n^2 + 24n + 12$. Malformed scan symbols and one-step-short fuel time out. This is only length preparation: it emits no formula bits and does not compose or refine a complete builder.

The executable input prefix now places the total framer and that tally in disjoint state images, with one total nine-symbol launch table between them. From every ordinary raw bitstring it reaches the same preserved-input unary-tally endpoint in exactly the sum of the two proved local traces plus one launch, and compilation is bounded by $18n^2 + 63n + 93$ in external input length. First-match isolation, timeout for the unused `zeroOne` tally scan symbol, and one-step-short timeout are explicit. No formula bit, cursor coordinate, or construction-time refinement is produced.

The standalone token appender carries one of the four canonical CNF tokens in its finite control state, scans the represented source, exact tally, and existing token suffix, writes exactly the selected two-bit symbol, and rewinds to the source focus. Its generic theorem quantifies every input, canonical prior token list, request, and exterior-left region. The distinguished start requests `T`; Lean proves direct formula-bit slots zero and one are populated true bits and that `CNFToken.t.bits = problem.encodedFormula.take 2` for every concrete tableau problem. The compiled first-token bound is $24n + 48$. Malformed tally/output symbols and one-step-short fuel time out.

The composed first-token prefix renames the complete 116-rule input prefix and complete 59-rule appender into disjoint injective state images and places nine symbol-preserving rules first in the literal table. Only the appender's accept/reject images are global halts. Every raw bitstring follows the exact framing, tally, launch, and append trace to the preserved workspace containing `[CNFToken.t]`. Its compiled external bound is $18n^2 + 87n + 147$, and blank-equivalent raw tapes reach the same encoded endpoint. Ending at the prefix endpoint before launch, malformed prefix or appender phases, and one-step-short fuel all remain timeout. This emits only two fixed formula bits: it does not compute the remaining header, interpret a dynamic cursor, or emit the complete formula.

The complete-header composition continues from that endpoint with a structurally generated unary `NatPolynomial` evaluator, a 16-rule unary-root controller, two complete 59-rule appender copies, and five total nine-symbol bridges under pairwise-disjoint injective state renamings. Its literal table has exactly `363 + evaluator.ruleCount` rules. Every raw input emits exactly `FormulaWidth` copies of `T` followed by `F`; the emitted bits equal `encodedFormula.take (2 * (FormulaWidth + 1))`, and an external `NatPolynomial` bounds the compiled run. This is the complete answer-independent width header only: dynamic cursor and formula-body emission, a complete builder refinement, and a concrete polynomial reduction remain absent.

The body-start composition continues from the complete header with a unary evaluator for the retained next-token coordinate, two total nine-symbol bridges, and one complete separator appender. Its literal table has exactly 440 plus the two evaluator rule counts. Every raw input reaches `T` repeated `FormulaWidth` times, followed by `F` and `Sep`; its bits equal the canonical encoded-formula prefix through that separator. The token coordinate is two past the formula variable slot bound and the corresponding bit coordinate is twice it. This retained coordinate is data, not an executable dynamic cursor; all subsequent body emission remains absent.

The first-literal composition continues from that endpoint with a third unary evaluator, three total nine-symbol bridges, and complete fixed `T` and `F` appender copies. Its literal table has exactly 585 plus the width, body-start next-slot, and first-literal next-slot evaluator rule counts. Every raw input reaches `T` repeated `FormulaWidth` times, followed by `F`, `Sep`, `T`, and `F`. A constructive schedule proof pins the final pair to the positive sign and unary-zero terminator of the first at-least-one shape-clause literal: positive variable zero. The emitted bits equal `encodedFormula.take (2 * (FormulaWidth + 4))`. The retained next token coordinate is four past the formula variable slot bound, but it remains data rather than an executable dynamic cursor; the rest of the formula body remains absent.

The first-clause composition continues from that endpoint with a fourth unary evaluator and a fixed 535-rule tail appender assembled from eight complete token-appender copies and seven total bridges. The global literal table has exactly 1138 plus the width, body-start next-slot, first-literal next-slot, and first-clause next-slot evaluator rule counts. Every raw input reaches `T` repeated `FormulaWidth` times, followed by `F`, `Sep`, `T`, `F`, `T`, `T`, `F`, `T`, `T`, `T`, `F`, and `Finish`. A constructive schedule proof identifies this as the complete positive at-least-one shape clause on variables zero, one, and two. The

emitted bits equal `encodedFormula.take (2 * (FormulaWidth + 12))`; the retained next token coordinate is twelve past the formula variable slot bound. It remains data rather than an executable dynamic cursor, and the remaining formula body is absent.

One literal token-cursor step continues from that first-clause endpoint. A total nine-symbol launch enters a fixed 45-rule cursor-advance table, giving a global table of exactly 1192 plus the four inherited unary-evaluator rule counts. Every raw input preserves the emitted clause while moving the retained unary coordinate from `FormulaVariableSlotBound + 12` to `FormulaVariableSlotBound + 13`. The direct decoder and token-level specification cursor prove that the consumed coordinate is the first in-range padding opportunity and therefore emits no token. Including launch, the suffix costs exactly `2*cursorWord.length + 8` work steps; the compiled run is bounded by `FirstClausePrefix.rawTimeBound + 48 + 12*cursorWord.length`. Malformed scratch, the unlaunched predecessor endpoint, and one-step-short total fuel remain timeout. This is one fixed padding transition, not a general cursor loop or arbitrary schedule decoder.

The remaining-padding composition continues from that one-step endpoint with two structurally generated unary evaluators and one fixed 25-rule countdown controller. Writing `D = (FormulaVariableSlotBound - 1) * (FormulaVariableSlotBound + 6)`, its literal table has exactly 1244 plus the six inherited and generated evaluator rule counts. Every raw input traverses all `D` remaining first-clause padding coordinates without emitting a token and reaches `FormulaVariableSlotBound + 1 + FormulaTokensPerClause`. Direct lookup at that coordinate is proved to return `Sep`, and the recursive specification run agrees across every no-emission step and the following separator observation. The exact compiled run has a verifier-fixed external input-size polynomial bound; malformed countdown scratch/root states, the unlaunched endpoint, and one-step-short fuel remain timeout. This identifies the second-clause boundary but does not emit the second clause or implement a general formula cursor.

The second-clause-separator composition continues with a selected 59-rule `Sep` appender, two total nine-symbol bridges, and the existing fixed 45-rule cursor advance. Its literal table has exactly 1366 plus the six inherited and generated evaluator rule counts. Every raw input emits the canonical prefix through the separator beginning clause two and advances the retained unary coordinate by one; direct lookup proves the following token is `F`. The emitted bits equal `encodedFormula.take (2 * (FormulaWidth + 13))`. The compiled run is bounded by the predecessor bound plus `246 + 24*n + 12*FormulaWidth + 12*cursorWord.length`. The combined 56-declaration audit has 15 empty closures, 11 using only `propext`, and 30 using only `propext` and `Quot.sound`. Malformed appender tally/output, malformed cursor scratch, both unlaunched endpoints, and one-step-short total fuel remain timeout. This is one fixed populated transition, not a general dynamic cursor or remaining-body builder, and it does not emit the following `F`.

Two further fixed compositions emit the complete negative literals on variables zero and one in clause two. They use selected `F`, `T`, and `F` appender copies, fixed cursor advances, and total symbol-preserving bridges. Exact schedule proofs identify each sign, unary index, and literal terminator; the second composition retains the following `Finish`.

The complete-second-clause composition emits that retained `Finish` with one selected 59-rule appender and advances the cursor once. Its suffix has exactly 113 rules, while the global table has exactly 2098 plus the six inherited and generated evaluator rule counts. Every raw input emits bits equal to `encodedFormula.take (2 * (FormulaWidth + 19))`. Direct lookup proves that the executed coordinate is the clause terminator and the retained next coordinate is in-range padding. The new external raw bound is the predecessor bound plus `390 + 24*n + 12*FormulaWidth + 12*cursorWord.length`. The exact 57-line audit has 15 empty, 10 `propext`, and 32 `propext`/`Quot.sound` closures. This completes clause two but does not by itself traverse its padding or implement a general formula cursor.

The second-clause-padding composition evaluates `D = (V - 1) * (V + 6) + 5`. With `C` abbreviating `formulaTokensPerClause`, Lean proves `D = C - 7`, traverses exactly those remaining padding

opportunities with the reused 25-rule countdown, and evaluates the target coordinate. Three total bridges and two fixed unary evaluators yield a global table with exactly 2150 plus the six inherited/generated evaluator counts and the two new evaluator counts. Every raw input reaches `V + 1 + 2 * formulaTokensPerClause`; every traversed coordinate is direct padding and the retained coordinate is the third clause's opening `Sep`. No token is emitted, so the bits remain `encodedFormula.take (2 * (FormulaWidth + 19))`. The exact 68-line audit has 26 empty, 9 `propext`, and 33 `propext/Quot.sound` closures. This reaches clause three only as a retained coordinate; it does not emit the separator or implement a general formula cursor.

The third-clause-separator composition reuses the selected 59-rule `Sep` appender and fixed 45-rule cursor advance behind one new total bridge. Its global table has exactly 2272 plus the eight inherited evaluator rule counts. Every raw input emits the separator beginning clause three, advances the retained coordinate to `V + 1 + 2 * formulaTokensPerClause + 1`, and proves the following direct token is `F`. The emitted bits equal `encodedFormula.take (2 * (FormulaWidth + 20))`. Its external bound adds `330 + 24*n + 12*FormulaWidth + 12*cursorWord.length` to the predecessor bound. The exact 56-line audit has 14 empty, 11 `propext`, and 31 `propext/Quot.sound` closures. This emits one fixed separator but does not emit the following `F`, complete clause three, or implement a general formula cursor.

The third-clause-first-literal composition reuses the audited 235-rule two-`F` appender/cursor suffix behind one total bridge. Its global table has exactly 2516 plus the eight inherited evaluator rule counts. Every raw input emits the complete negative literal on variable zero in clause three and retains `V + 1 + 2 * formulaTokensPerClause + 3`, the negative-sign coordinate on variable two. Constructive schedule proofs identify the third excluded pair as zero and two and prove that the emitted sign, unary-zero terminator, and retained sign are all `F`. The emitted bits equal `encodedFormula.take (2 * (FormulaWidth + 22))`. Its external bound adds `732 + 48*n + 24*FormulaWidth + 24*cursorWord.length` to the separator bound. The exact 87-line audit has 24 empty, 18 `propext`, and 45 `propext/Quot.sound` closures. This emits one fixed literal but does not emit the next negative sign, complete clause three, or implement a general formula cursor.

The third-clause-second-literal composition adds a fixed 479-rule `F T T F` appender/cursor suffix behind one total bridge. Its nested component tables have 113, 235, 357, and 479 rules, and its global table has exactly 3004 plus the eight inherited evaluator rule counts. Every raw input emits the complete negative literal on variable two and retains `V + 1 + 2 * formulaTokensPerClause + 7`, the following clause-terminator coordinate. Direct and specification schedule proofs establish `F`, `T`, `T`, `F`, and the following `Finish`. The emitted bits equal `encodedFormula.take (2 * (FormulaWidth + 26))`. Its external bound adds `1752 + 96*n + 48*FormulaWidth + 48*cursorWord.length` to the first-literal bound. The exact 145-line audit has 46 empty, 32 `propext`, and 67 `propext/Quot.sound` closures. This emits the second fixed literal but does not emit the following `Finish`, complete clause three, or implement a general formula cursor.

The complete-third-clause composition emits that retained `Finish` with one selected 59-rule appender and advances the cursor once. Its suffix has exactly 113 rules, while the global table has exactly 3126 plus the eight inherited evaluator rule counts. Every raw input emits bits equal to `encodedFormula.take (2 * (FormulaWidth + 27))`. Direct lookup proves that the executed coordinate is the clause terminator and the retained next coordinate is in-range padding. The new external raw bound is the predecessor bound plus `498 + 24*n + 12*FormulaWidth + 12*cursorWord.length`. The exact 57-line audit has 14 empty, 10 `propext`, and 33 `propext/Quot.sound` closures. This completes clause three but does not by itself traverse its padding or implement a general formula cursor.

The third-clause-padding composition evaluates `D = (V - 1) * (V + 6) + 4`. With `C` abbreviating `formulaTokensPerClause`, Lean proves `D = C - 8`, traverses exactly those remaining padding opportunities with the reused 25-rule countdown, and evaluates the target coordinate. Three total bridges and two fixed unary evaluators yield a global table with exactly 3178 plus the eight inherited/generated evaluator counts and the two new evaluator counts. Every raw input reaches

`V + 1 + 3 * formulaTokensPerClause`; every traversed coordinate is direct padding and the retained coordinate is the fourth clause's opening `Sep`. No token is emitted, so the bits remain `encodedFormula.take (2 * (FormulaWidth + 27))`. The exact 68-line audit has 26 empty, 9 `propext`, and 33 `propext/Quot.sound` closures. This reaches clause four only as a retained coordinate; it does not emit the separator or implement a general formula cursor.

The fourth-clause-separator composition reuses the selected 59-rule `Sep` appender and fixed 45-rule cursor advance behind one outer total nine-symbol bridge. Its 113-rule suffix yields a global table with exactly 3300 plus the ten inherited/generated unary-evaluator rule counts. Every raw input emits exactly the separator beginning clause four, advances the retained coordinate to `V + 1 + 3 * formulaTokensPerClause + 1`, and proves the following direct token is `F`. The emitted bits equal `encodedFormula.take (2 * (FormulaWidth + 28))`. Its external bound is the predecessor bound plus `426 + 24*n + 12*FormulaWidth + 12*cursorWord.length`. The exact 56-line audit covers all 48 new public declarations plus eight reused interfaces, with 14 empty, 11 `propext`, and 31 `propext/Quot.sound` closures. This emits one fixed separator but does not emit the following `F`, complete clause four, or implement a general formula cursor.

The fourth-clause-first-literal composition reuses the fixed 357-rule `F T F` appender/cursor suffix behind one outer total nine-symbol bridge. The global table has exactly 3666 plus the ten inherited/generated unary-evaluator rule counts. Every raw input emits the complete first negative literal on variable one, advances the retained coordinate to `V + 1 + 3 * formulaTokensPerClause + 4`, and proves the following direct token is `F`. The emitted bits equal `encodedFormula.take (2 * (FormulaWidth + 31))`. Its external bound is the predecessor bound plus `1422 + 72*n + 36*FormulaWidth + 36*cursorWord.length`. The exact 115-line audit covers 97 new declarations, 16 reused suffix interfaces, and two dead-state facts, with 33 empty, 25 `propext`, and 57 `propext/Quot.sound` closures. This emits one fixed literal but does not emit the second literal, complete clause four, or implement a general formula cursor.

The fourth-clause-second-literal composition reuses the fixed 479-rule `F T T F` appender/cursor suffix behind one outer total nine-symbol bridge. The global table has exactly 4154 plus the ten inherited/generated unary-evaluator rule counts. Every raw input emits the complete second negative literal on variable two, advances the retained coordinate to `V + 1 + 3 * formulaTokensPerClause + 8`, and proves the following direct token is `Finish`. The emitted bits equal `encodedFormula.take (2 * (FormulaWidth + 35))`. Its external bound is the predecessor bound plus `2232 + 96*n + 48*FormulaWidth + 48*BuilderFourthClauseSeparatorStep.cursorWord.length`. The exact 147-line audit covers 124 new declarations, 21 reused suffix interfaces, and two dead-state facts, with 46 empty, 32 `propext`, and 69 `propext/Quot.sound` closures. This emits one fixed literal but does not emit the following `Finish`, complete clause four, or implement a general formula cursor.

The complete-fourth-clause composition reuses a selected 59-rule `Finish` appender and the fixed 45-rule cursor behind two total nine-symbol bridges. The selected suffix has exactly 113 rules and the global table has exactly 4276 plus the ten inherited/generated unary-evaluator rule counts. Every raw input emits the terminator that completes clause four, advances the retained coordinate to `V + 1 + 3 * formulaTokensPerClause + 9`, and proves the following direct token is padding. The emitted bits equal `encodedFormula.take (2 * (FormulaWidth + 36))`. Its external bound is the predecessor bound plus `618 + 24*n + 12*FormulaWidth + 12*BuilderFourthClauseSeparatorStep.cursorWord.length`. The exact 57-line audit covers 55 new public declarations and two dead-state facts, with 14 empty, 10 `propext`, and 33 `propext/Quot.sound` closures. This completes clause four but does not traverse its padding or implement a general formula cursor.

At the current builder frontier, the second-constraint-first-literal terminator composition reuses the selected 59-rule `F` appender and fixed 45-rule cursor behind one outer total nine-symbol bridge. The global table has exactly 5164 plus the sixteen inherited/generated unary-evaluator rule counts. Every raw input emits exactly the terminating `F`, preserves `encodedFormula.take (2 * (FormulaWidth + 42))`, and retains `V + 1 + Q*C + 6`. A constructive schedule case split proves that the following

direct token is `Finish` when the tape width is one and the positive `T` beginning the next literal at wider widths; neither token is emitted. The external bound is the predecessor bound plus `594 + 24*n + 12*FormulaWidth + 12*cursorWord.length`. The 56-declaration audit has 14 empty, 11 `propext`, and 31 `propext/Quot.sound` closures. This completes one literal, not the second constraint or the formula builder.

The successor-token composition now evaluates the represented tableau width and selects one of two literal transitions inside a fixed 93-rule branch table. At width one it appends `Finish`; at every wider width it appends `T`. The global table has exactly 5284 plus the eighteen inherited/generated unary-evaluator rule counts, preserves `encodedFormula.take (2 * (FormulaWidth + 43))`, and retains `V + 1 + Q*C + 7`. The next opportunity is proved to be padding at width one and unary `T` at wider widths, but is not emitted. The external bound is the predecessor bound plus `600 + 24*n + 12*FormulaWidth + 12*width + 12*widthRootPrefixLength + 6*widthWorkSteps + 6*targetWorkSteps`. The 82-declaration audit has 37 empty, 12 `propext`, and 33 `propext/Quot.sound` closures, with no project axiom or `Classical.choice`. This is one width-selected token transition, not the remaining second constraint or a complete formula builder.

The next four opportunity machines use the same reviewed 93-rule optional-appender table. At width one each schedule position is padding, so its direct skip bridge emits nothing. At wider widths the first appends the first unary `T` of the second literal and the second appends its second unary `T`; the third appends its third unary `T`, and the fourth appends its fourth unary `T`. The fourth machine has exactly 5764 literal rules plus the twenty-six inherited/generated unary-evaluator rule counts, preserves `encodedFormula.take (2 * (FormulaWidth + 43 + if tapeWidth = 1 then 0 else 4))`, and retains `V + 1 + Q*C + 11`. The following position is proved to be padding at width one and the terminating `F` at wider widths, but is not consumed. Its external bound is the third opportunity bound plus `648 + 24*n + 12*FormulaWidth + 12*width + 12*widthRootPrefixLength + 6*widthWorkSteps + 6*targetWorkSteps`. The 82-declaration audit covers all 66 new declarations, fourteen reused optional-appender interfaces, and two strengthened schedule boundaries: 37 closures are empty, 12 use only `propext`, and 33 use only `propext` and `Quot.sound`.

The fifth opportunity machine uses a new 93-rule optional-terminator controller over the audited token-appender table. At width one it consumes padding and emits nothing; at wider widths it appends the second literal's terminating `F`. Its complete table has exactly 5884 literal rules plus twenty-eight inherited/generated unary-evaluator rule counts, preserves `encodedFormula.take (2 * (FormulaWidth + 43 + if tapeWidth = 1 then 0 else 5))`, and retains `V + 1 + Q*C + 12`. The following position is padding at width one and the opening unary `T` of the following literal at wider widths, but is not consumed. Its external bound is the fourth opportunity bound plus `660 + 24*n + 12*FormulaWidth + 12*width + 12*widthRootPrefixLength + 6*widthWorkSteps + 6*targetWorkSteps`. The 82-declaration audit covers 66 new outer declarations, fourteen new optional-terminator interfaces, and two strengthened schedule boundaries: 37 closures are empty, 12 use only `propext`, and 33 use only `propext` and `Quot.sound`.

The sixth opportunity machine reuses the reviewed 93-rule optional-appender controller. At width one it consumes padding and emits nothing; at wider widths it appends the following literal's opening positive `T`. Its complete table has exactly 6004 literal rules plus thirty inherited/generated unary-evaluator rule counts, preserves `encodedFormula.take (2 * (FormulaWidth + 43 + if tapeWidth = 1 then 0 else 6))`, and retains `V + 1 + Q*C + 13`. The following position is padding at width one and the first unary-index `T` of the following literal at wider widths, but is not consumed. Its external bound is the fifth opportunity bound plus `672 + 24*n + 12*FormulaWidth + 12*width + 12*widthRootPrefixLength + 6*widthWorkSteps + 6*targetWorkSteps`. The 82-declaration audit covers 66 new declarations, fourteen reused optional-appender interfaces, and two strengthened schedule boundaries: 37 closures are empty, 12 use only `propext`, and 33 use only `propext` and `Quot.sound`.

The seventh opportunity machine reuses the reviewed 93-rule optional-appender controller. At width one it consumes padding and emits nothing; at wider widths it appends the first unary-index `T` of the

following literal. Its complete table has exactly 6124 literal rules plus thirty-two inherited/generated unary-evaluator rule counts, preserves `encodedFormula.take (2 * (FormulaWidth + 43 + if tapeWidth = 1 then 0 else 7))`, and retains `V + 1 + Q*C + 14`. The following position is padding at width one and the second unary-index `T` of the following literal at wider widths, but is not consumed. Its external bound is the sixth opportunity bound plus `684 + 24*n + 12*FormulaWidth + 12*width + 12*widthRootPrefixLength + 6*widthWorkSteps + 6*targetWorkSteps`. The 82-declaration audit covers 66 new declarations, fourteen reused optional-appender interfaces, and two strengthened schedule boundaries: 37 closures are empty, 12 use only `propext`, and 33 use only `propext` and `Quot.sound`.

Separately, the global locked-NAND construction now has both final semantic branches and the exact typed threshold. A strict version-zero grammar serializes source circuits, the complete full candidate, and the source-derived baseline. Direct normalization semantics, codec round trips, fail-closed malformed input, and the pure all-bitstring source/target equivalence are proved. The 48-declaration audit has four empty closures, 37 using only `propext`, and seven using only `propext` and `Quot.sound`. That module is a pure semantic transformation and does not by itself supply an executable parser or emitter.

The following strict-v0 source-parser milestone supplies a direct nine-symbol machine with 228 control states and 2,052 pairwise query-distinct literal rules. For every input bitstring, its exact execution accepts exactly the valid source encodings, preserves valid source bytes verbatim, and returns the empty word for invalid grammar or references. The compiled machine cannot time out within $6 \cdot 4096 \cdot (n+1)^3$ raw transitions. It is packaged as a polynomial-time language machine, a nonexpanding polynomial-time validation function, and the validator program's exact leaf raw-machine refinement.

The subsequent target-emitter milestone supplies a literal finite controller with an exact all-input polynomial work bound, a separate output-size bound, exact target bytes, and recursive raw-machine refinement for the strict parser/emitter composition. Malformed grammar and intrinsically invalid references remain fail closed at that strict boundary.

The present package binds that exact composed function and the already-proved encoded language equivalence as a concrete `PolynomialReduction` from `EncodedNANDSAT` to `EncodedLockedNANDThreshold`. It preserves exact function identity, output bytes, a `ReducesTo` witness, and recursive raw-machine refinement. Its 16-declaration audit has two empty closures, two using only `propext`, and twelve using only `propext` and `Quot.sound`. The report-facing `PNP.Main.locked_nand_threshold` theorem now composes the CNF-to-NAND and strict locked-NAND reductions into an exact all-bitstring `ReducesTo CNFSAT EncodedLockedNANDThreshold` result. Its axiom transcript contains only `propext` and `Quot.sound`, not the legacy abstract string-handle assumption. A deterministic target decider, concrete CNFSAT NP-hardness transport, residual-band and ZeroSlack/PCCMin completion, and $P = NP$ remain unproved.

The residual-gain chain milestone reconstructs the iteration-count edge from legacy report Sections 16 and 17. For every finite disclosed chain whose adjacent implementations pass the strict equivalent-gain checker, Lean proves that the endpoint residual slack plus the chain length is at most the starting residual slack. The endpoint remains semantically equivalent and has the same exhaustive reference minimum. The already-proved locked-candidate bound then specializes every accepted chain to at most four gains. All 12 declarations in the universal module have empty axiom closure; the four locked declarations use only `propext` and `Quot.sound`. This does not find a route, certify stopping, prove ZeroSlack or exact minimization, or establish a polynomial checker or PCCMin runtime.

The global strict-gain stopping milestone reconstructs the corresponding semantic endpoint condition from legacy report Section 16. For every finite direct-wire implementation, positive residual slack is equivalent to the existence of some strictly smaller semantically equivalent implementation. Zero slack and semantic minimality are each equivalent to global absence of such an implementation. A verified

chain endpoint with separately proved global no-gain evidence therefore has zero slack and packages an exact minimum result equivalent to the starting implementation. All 12 public declarations have empty axiom closure. The positive witness comes from exhaustive reference minimization, so this is not a route generator or polynomial stopping algorithm; finite-list failure still cannot establish global absence, and the report's ZeroSlack certificate, PCCMin exactness/runtime, SAT-in-P conclusion, and (P=NP) theorem remain unproved.

The terminal full-carrier milestone reconstructs the direct-wire full-mode part of the terminal whole-carrier bridge from legacy report Section 8. A terminal realization agrees with the complete implementation on every input and output coordinate; terminalization preserves exact gate count. Lean states attainment and the universal lower bound independently, proves that the terminal minimum equals the exhaustive reference minimum (the direct-wire terminal `RW-MuBridge`), and proves that every cheaper whole-span realization gives strict residual descent. Positive slack is equivalent to existence of one and zero slack to its absence. All 22 public declarations have empty axiom closure. The quotient carrier and mode firewall, proper supports, saturation, BCEL/BN2–BN6, selector completeness, ZeroSlack/PCCMin, polynomial runtime, SAT-in-P, and $P = NP$ remain unproved.

Machine field	Current value
`mathematicalTheoremEstablished`	**false**
`publicTheoremEmissionAllowed`	**false**
`finalTheoremReady`	**false**
`publicTheoremStatement`	`null`
`rootLeanTheorem`	`PNP.Main.p_eq_np`
`rootLeanTheoremPresent`	**false**

2 Concrete publication gate

The compatibility entry is `PNP.Main.p_eq_np`; a matching name alone is insufficient. The separate publication target is `PNP.Main.ConcretePEqualsNP`. Eligibility also requires the exact theorem type, reviewed kernel fingerprints, and a tightly fixed Lean-standard axiom closure. The target is present as an inactive definition backed by finite charged-pipeline semantics; its presence is not a proof. The compiler/refinement link to the raw machine kernel is formalized, but no deterministic polynomial CNF-SAT decider or concrete publication root is present. The direct CNF verifier proves NP membership only.

The expected fingerprints are intentionally unset. This is a fail-closed migration gate, not a missing runtime input. A verifier must reject any attempt to set `passed=true` while a fingerprint is null, while the concrete model is ineligible, or while any other subcheck is false.

Gate subcheck	Result
standardComplexityModelEligible	**true**
concreteTargetPresent	**true**
concreteTargetIsDefinition	**true**
concreteTargetKernelTypeFingerprintConfigured	**false**
concreteTargetKernelTypeFingerprintMatches	**false**
concreteTargetKernelValueFingerprintConfigured	**false**
concreteTargetKernelValueFingerprintMatches	**false**
compatibilityRootPresent	**false**
compatibilityRootIsTheorem	**false**
compatibilityRootHasExactConcreteType	**false**
compatibilityRootKernelTypeFingerprintConfigured	**false**

Gate subcheck	Result
compatibilityRootKernelTypeFingerprintMatches	**false**
axiomClosureFingerprintConfigured	**false**
axiomClosureFingerprintMatches	**false**
sourceClosureFingerprintConfigured	**false**
sourceClosureFingerprintMatches	**false**
axiomClosureUsesOnlyLeanStandardAllowlist	**false**

Allowed foundation axioms are fixed to `Classical.choice`, `Quot.sound`, and `propext`. Project axioms, `sorryAx`, or any unknown axiom make the gate fail. The four currently disclosed project axioms are:

- `PNP.CheckPCCPackexp`
- `PNP.GeneratePCCPack`
- `PNP.LockedNANDThreshold`
- `PNP.ResidualBandExactMinimization`

2.1 Why the abstract bridge is ineligible

The current `PNP.PEqualsNP` abbreviation compares witness classes whose language and algorithm structures carry string names or code handles rather than concrete execution semantics and proved polynomial bounds. Consequently, even a kernel-clean theorem of that abstract type would not establish the standard P-versus-NP statement. It is compatibility information only.

The separate `PNP.Concrete.PEqualsNP` target uses proof-bearing P and NP witnesses, canonical input/certificate pairing, finite program syntax, and charged polynomial runtime and output bounds. Its recursive compiler/refinement link to the raw machine kernel is formalized. SAT completeness, SAT in P, and the root theorem remain absent.

3 Formal milestone ledger

Each earned row requires exact compiled names and theorem kinds, empty axiom closures, per-name domain-separated kernel-type SHA-256 matches for 2589 detailed candidates, and the pinned whole-Lean source closure. Human-readable scope remains conservative: type or source drift revokes credit.

Milestone	Status	Exact scope	Boundary
Concrete machine and cost kernel	formalized-foundation-only	One literal finite four-stage pipeline handles every raw bitstring. The total framer uses exactly 4 work steps on empty input, 4 * k * k + 9 * k + 7 for complete two-bit cells, and 4 * k * k + 9 * k + 5 when the final cell is partial; its compiled raw cost is bounded by 6 * m * m + 39 * m + 75. Every supplied exact n-step target run lifts to exactly 3 * n work steps. PipelineCompiler preserves one raw target's verdict and ordinary output at an explicit external polynomial. PipelineSequentialCompiler composes two raw targets in one literal finite table, passes either first verdict onward, preserves the second verdict and output, retains stuck-first timeout, and proves R(m) = PipelineRaw(p)(m) + 6 + PipelineRaw(q)(m + p(m) + 1). PipelineRefinement recursively applies that compiler to every FunctionProgram composition and DecisionProgram precomposition node. Its 16 audited declarations have empty axiom closure, and PolynomialTimeDecider.compileToMachine exposes the complete tree as one raw polynomial-time machine.	This closes the concrete complexity machine-link blocker only. It does not construct a deterministic polynomial-time CNF-SAT decider or an NP-completeness reduction. CNF-SAT in P, NP-completeness, and P = NP are not established; PNP.Main.p_eq_np remains absent and the publication gate remains false.
Concrete P, NP, and polynomial reductions	formalized	Bitstring languages, finite charged decision/verifier/function pipelines grounded in concrete machines, polynomial certificate/runtime/output bounds, recursively compiled exact raw-machine refinements, many-one reductions, and the NP-complete-in-P implication.	The recursive refinement compiler closes the charged-to-raw machine link, but it does not construct raw paired verifiers beyond the existing CNF-SAT membership witness, prove CNF-SAT is in P or NP-complete, or establish the publication root theorem.

Milestone	Status	Exact scope	Boundary
Concrete universal CNF-SAT verifier	formalized-np-membership-only	An unconditional formula-grammar outcome, universal bounded work outcome, exact accept/reject semantics, no-timeout theorem, literal finite-machine paired verifier, and CNF-SAT membership in NP.	This proves CNF-SAT membership in NP only; it does not prove CNF-SAT is in P, NP-completeness, or P = NP.
Concrete Cook-Levin dimensions and variable layout	formalized-foundation-only	Fuel padding preserves designated halts and stuck timeouts; every literal machine state is below an explicit finite ceiling; the input, time, and tape dimensions are executable NatPolynomial expressions; and tape-symbol, head, state, certificate-bit, and certificate-length variables occupy proved disjoint in-range blocks. Elementary at-least-one and pair-exclusion CNF clauses have exact propositional semantics. All 20 reviewed theorems have empty axiom closure.	This is the finite layout substrate only. It does not yet construct a transition tableau, prove tableau soundness or completeness, compile a formula emitter, define a polynomial reduction to CNFSAT, or establish CNFSAT NP-completeness, CNFSAT in P, or P = NP.
Concrete fixed-certificate execution tableau	formalized-foundation-only	For one fixed source input, certificate, finite-rule machine, and fuel budget, the canonical raw execution trace has exactly fuel + 1 configurations. Intrinsic first-match transition validity is equivalent to equality with that trace, every valid endpoint equals the ordinary run endpoint, and accept/reject/timeout verdicts agree exactly with boundedDecide. All 30 declarations and eight reviewed theorem types have empty axiom closure.	This is the semantic fixed-certificate tableau only. It does not yet quantify a variable certificate, encode tape windows or tableau rows as CNF, compile a formula emitter, define a polynomial reduction to CNFSAT, or establish CNFSAT NP-completeness, CNFSAT in P, or P = NP.

Milestone	Status	Exact scope	Boundary
Uniform bounded verifier tableaux	formalized-foundation-only	For an arbitrary proof-bearing verifier in the concrete NP model and one source input, the complete finite decision pipeline is compiled to one raw machine. Every certificate inside the explicit certificate polynomial shares one answer-independent raw fuel obtained by evaluating the compiled raw-time polynomial at the maximum encoded input size. Per-certificate fuel is below that bound, padding follows a proved halt, exact verdicts are preserved, and language membership is equivalent to an existential bounded accepting semantic tableau. All 41 declarations and nine reviewed theorem types have empty axiom closure.	This is a uniform semantic verifier-tableau equivalence only. It does not encode configurations, transition windows, or variable-length certificates as Boolean clauses; emit CNF; prove emitter size or runtime polynomials; define a polynomial reduction to CNFSAT; or establish CNFSAT NP-completeness, CNFSAT in P, or P = NP.
Finite local Cook-Levin CNF compiler	formalized-foundation-only	Width-indexed literals materialize exact Boolean assignments and compile to the concrete CNF semantics without out-of-range variables. Unit constraints, finite implications, and pairwise exactly-one groups have exact satisfaction equivalences; local programs have exact recursive clause counts, satisfiability reflection, and a proved well-scoped formula. All 75 declarations and nine reviewed theorem types have empty axiom closure.	This is a local clause compiler only. It does not enumerate the verifier tableau, encode initialization, transition windows, certificate length, or acceptance as a complete formula; prove an all-input emitter size/runtime polynomial; define a reduction to CNFSAT; or establish CNFSAT NP-completeness, CNFSAT in P, or P = NP.

Milestone	Status	Exact scope	Boundary
Finite whole-tableau Cook-Levin CNF syntax	formalized-foundation-only	A uniform bounded verifier-tableau problem is compiled answer-independently into one finite, well-scoped concrete CNF formula. The program emits one-hot symbol/head/state rows, first-match control updates, untouched-cell preservation, exact input-only or symbolic paired-certificate initialization, and a designated final accept constraint. Canonical encoding/decoding, exact local satisfaction reflection, and exact emitted clause counting are proved. All 81 declarations and ten reviewed theorem types have empty axiom closure.	This is finite formula syntax and local-clause reflection only. It does not yet prove that formula satisfiability is equivalent to an accepting raw verifier execution, prove external encoded-output-size or construction-runtime polynomials, define a concrete polynomial reduction to CNFSAT, or establish CNFSAT NP-completeness, CNFSAT in P, or P = NP.
Finite whole-tableau Cook-Levin CNF semantics	formalized-foundation-only	Every satisfying assignment decodes to a unique-symbol, unique-head, unique-state finite row at each bounded time. The decoded initial row is exact in both verifier modes, each adjacent row is the literal deterministic first-match successor, and the final state is accepting. Conversely, every such intrinsic finite accepting tableau constructs a satisfying assignment. Formula satisfiability and concrete CNFSAT membership are therefore equivalent to this finite semantics. All 161 explicit declarations are axiom-audited; the six reviewed theorem types depend only on the permitted Lean standard axioms Quot.sound and propext, with no project axiom.	This milestone itself proves equivalence only with an intrinsic finite-row model; the following raw-tape milestone supplies the execution bridge. External encoded-output-size and construction-runtime polynomials, a concrete polynomial reduction to CNFSAT, CNFSAT NP-completeness, CNFSAT in P, and P = NP remain absent.

Milestone	Status	Exact scope	Boundary
Finite tableau to raw Tape semantics	formalized-foundation-only	The finite Cook-Levin rows are proved exactly equivalent to ordinary focused raw Tape execution. Absolute two-sided tape observations cover explicit and implicit blanks; the local successor matches designated halts, missing-rule stutter, and the literal first matching raw rule; a centered head invariant rules out finite-window clamping; and both verifier input modes have exact initial rows, including reversible bounded-certificate reconstruction. Formula satisfiability and concrete CNFSAT membership are therefore equivalent to the concrete verifier language. All 54 explicit declarations are axiom-audited, and the nine reviewed theorem types use only the permitted Lean standard axiom allowlist with no project axiom.	This proves exact semantic reduction correctness only. It does not yet prove external encoded-formula-size or formula-construction-runtime polynomials, package a concrete PolynomialReduction, establish CNFSAT NP-completeness, CNFSAT in P, or P = NP.
External Cook-Levin encoded-formula size	formalized-foundation-only	The actual canonical unary-indexed CNF bitstring generated for a fixed concrete polynomial-time verifier is bounded by an explicit NatPolynomial evaluated only at the external source-input length. Exact codec lengths, both verifier input modes, every concrete tableau constraint family, program and emitted-clause counts, clause width, and the mode-sensitive exact formula-variable width polynomial are covered. All 110 explicit declarations are axiom-audited; the ten reviewed theorem types use only the permitted Lean standard axioms Quot.sound and propext, with no project axiom or choice axiom.	This does not implement or time a raw finite formula builder, package a concrete PolynomialReduction, establish CNFSAT NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Rectangular Cook-Levin formula schedule	formalized-foundation-only	For every concrete verifier-tableau problem, a finite answer-independent rectangular schedule allocates exact constraint, clause, token, and raw-bit slots. Removing empty slots in order reproduces the existing canonical local program, bounded clauses, CNF tokens, and encoded formula. The raw-bit schedule length is exactly the previously proved encodedFormulaSizePolynomial evaluated at external source-input length. All 79 explicit declarations are axiom-audited; the eight reviewed theorem types use only the permitted Lean standard axioms Quot.sound and propext, with no project axiom or choice axiom.	This is a pure schedule specification. It does not interpret a slot as a constant-time raw-machine action, implement or time a raw finite formula builder, construct a FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Direct Cook-Levin formula cursor	formalized-foundation-only	For every concrete verifier-tableau problem and natural coordinate, direct constraint, clause, token, and raw-bit decoders agree pointwise with the canonical rectangular schedules while preserving out-of-range, valid-padding, and populated states. Fuelled bit traversal proves exact bounded prefixes, complete traversal, one-step-short behavior, terminal behavior, excess-fuel stability, the external encoded-size polynomial, and canonical emitted output. A token-level specification cursor additionally exposes exact in-range, done, and terminal single-step laws. All 136 explicit declarations are axiom-audited; the 16 reviewed theorem types use only the permitted Lean standard axioms Quot.sound and propext, with no project axiom or choice axiom.	This is a Lean specification cursor. It does not prove constant-time raw slot interpretation, implement or time a raw finite formula builder, construct a FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Literal Cook-Levin builder input-length tally	formalized-foundation-only	A fixed 19-rule work machine starts at the proved all-input framer endpoint, preserves every external source bit, appends exactly one unary tally symbol per source bit in fresh right-side workspace, and returns to the source head. It accepts after exactly 2*n*n + 4*n + 2 work steps; the compiled raw machine reaches the encoded endpoint after exactly 12*n*n + 24*n + 12 steps. Malformed internal scan symbols and one-step-short fuel both time out. All 39 public declarations are axiom-audited; the ten reviewed theorem types use only the permitted Lean standard axioms Quot.sound and propext, with no project or choice axiom.	This is only the input-length preparation stage. It does not emit formula bits, interpret direct cursor slots as raw transitions, compose a complete raw formula builder, construct a FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-completeness, establish CNFSAT in P, or prove P = NP.
Executable Cook-Levin builder input prefix	formalized-foundation-only	One literal finite work machine contains collision-free renamed copies of the total all-input framer and fixed 19-rule unary tally, with a total nine-symbol tape/head-preserving launch between them. Every raw bitstring reaches the exact preserved-input tally endpoint in totalInputFramerWorkSteps(input) + 1 + 2*n*n + 4*n + 2 work transitions and within the external compiled polynomial 18*n*n + 63*n + 93. All 40 public declarations are axiom-audited: 29 have empty closure, one uses only propext, and ten use only propext and Quot.sound.	This is still only an executable input-preparation prefix. It emits no formula bit, performs no raw formula-cursor interpretation, supplies no complete formula builder or construction-time RawRefinement, packages no concrete PolynomialReduction, and establishes neither CNFSAT NP-completeness, CNFSAT in P, nor P = NP.

Milestone	Status	Exact scope	Boundary
Standalone Cook-Levin builder token appender	formalized-foundation-only	A fixed 59-rule work machine appends any of the four canonical two-bit CNF tokens selected only by its finite control state after a represented source word and exact unary input-length tally. For source length n and k existing tokens, it accepts at the exact endpoint after 2 * (max 1 n + n + k + 3) work steps. Its distinguished start state appends T, the first canonical formula header token, within the external compiled bound 24*n + 48; the two emitted bits are proved equal to the first two bits of every concrete verifier-tableau encoding. Malformed tally/output phases and one-step-short fuel remain timeouts. All 68 public declarations are axiom-audited: 42 have empty closure, 13 use only propext, and 13 use only propext and Quot.sound.	This milestone audits the token appender independently of its later composed use. It does not compute the remaining width header, interpret a dynamic formula cursor coordinate, emit a complete formula, construct a builder RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Composed Cook-Levin builder first-token prefix	formalized-foundation-only	One literal 184-rule finite work machine contains all 116 executable input-prefix rules, nine total symbol-preserving bridge rules, and all 59 token-appender rules under injective disjoint state renamings. Every raw bitstring runs through total framing, exact unary input tallying, the bridge, and the appender to preserve the workspace and emit exactly the first T header token. The exact all-input work trace compiles within 18*n*n + 87*n + 147 raw steps; the final two bits equal the first two bits of every canonical encoded verifier-tableau formula. Prefix-endpoint, malformed tally/output, and one-step-short negative cases time out. All 37 public declarations are axiom-audited: 21 have empty closure, three use only propext, and 13 use only propext and Quot.sound, with no project axiom or Classical.choice.	This emits exactly the fixed answer-independent first T token, hence only the first two canonical formula bits. It does not compute the remaining width header, implement a dynamic cursor, emit a complete formula, construct a builder RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Composed Cook-Levin complete width header	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine contains the 184-rule raw-input/first-token prefix, a structurally generated unary NatPolynomial evaluator, a 16-rule unary-root controller, two complete 59-rule token appenders, and five total nine-symbol bridges under injective pairwise-disjoint state renamings. Its table has exactly 363 plus the evaluator rule count rules. Every raw input follows an exact all-input trace, evaluates the verifier's mode-sensitive formula width into unary scratch space without calling NatPolynomial.eval in executable rules, and emits exactly FormulaWidth copies of T followed by F. The resulting bits are the complete canonical encoded-formula width header, and an external NatPolynomial bounds the compiled raw run. The evaluator's 74 and composition's 84 public declarations are completely axiom-audited: every closure is empty or uses only propext and Quot.sound, with no project axiom or Classical.choice.	This endpoint is the complete answer-independent width header only. It does not implement the dynamic formula cursor or body emission, construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Composed Cook-Levin body-start prefix	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine contains the complete width-header machine, a structurally generated unary evaluator for the next canonical token slot, one complete 59-rule token appender, and two total nine-symbol bridges under injective pairwise-disjoint state renamings. Its table has exactly 440 plus the width-evaluator and next-token-slot-evaluator rule counts. Every raw input follows an exact all-input trace, preserves the unary input tally and header workspace, retains next token coordinate FormulaVariableSlotBound + 2 and bit cursor twice that coordinate, and emits exactly FormulaWidth copies of T followed by F and Sep. The resulting bits are the canonical encoded-formula prefix through the first body separator; the compiled run is bounded by the complete-header bound plus 72, six times the unary next-slot evaluator work count, 24*n, and 12*width. All 60 public declarations are axiom-audited: 30 have empty closure, five use only propext, and 25 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits only the fixed answer-independent body-opening separator after the complete width header and retains its coordinate as data. It does not implement a dynamic formula cursor or subsequent body emission, construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Composed Cook-Levin first-literal prefix	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine contains the complete body-start-prefix machine, a structurally generated unary evaluator for token coordinate FormulaVariableSlotBound + 4, two complete 59-rule token appenders, and three total nine-symbol bridges under four injective pairwise-disjoint state renamings. Its table has exactly 585 plus the width, body-start next-slot, and first-literal next-slot evaluator rule counts. Every raw input follows an exact all-input trace, preserves the unary tally and earlier token workspace, retains the corresponding doubled bit cursor, and emits exactly FormulaWidth copies of T followed by F, Sep, T, and F. The final T/F pair is constructively pinned to the positive sign and unary-zero terminator of the first scheduled literal, positive variable zero, and all emitted bits equal encodedFormula.take (2 * (FormulaWidth + 4)). The compiled run is bounded by the body-start bound plus 174, six times the new unary evaluator work count, 48*n, and 24*width. All 74 current public declarations are axiom-audited: 21 have empty closure, three use only propext, and 50 use only propext and Quot.sound, with no project axiom or Classical.choice; the added halt-separation interface supports the reviewed first-clause composition.	This milestone emits only the first canonical literal after the body-opening separator and retains the following coordinate as data. It does not implement a dynamic formula cursor or remaining body emission, construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Composed Cook-Levin first-clause prefix	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine contains the complete first-literal-prefix machine, a structurally generated unary evaluator for token coordinate FormulaVariableSlotBound + 12, and a fixed 535-rule tail made from eight complete 59-rule token appenders and seven total nine-symbol bridges under disjoint injective state renamings. Its table has exactly 1138 plus the width, body-start next-slot, first-literal next-slot, and first-clause next-slot evaluator rule counts. Every raw input follows an exact all-input trace, preserves the unary tally and earlier token workspace, retains the corresponding doubled bit cursor, and emits exactly FormulaWidth copies of T followed by F, Sep, T, F, T, T, F, T, T, T, F, and Finish. The final eight tokens are constructively pinned to the remainder of the complete positive at-least-one shape clause on variables zero, one, and two; the following direct token slot is proved to be valid padding. All emitted bits equal encodedFormula.take (2 * (FormulaWidth + 12)). The compiled run is bounded by the first-literal bound plus 1158, six times the new unary evaluator work count, 192*n, and 96*width. All 79 public declarations are axiom-audited: 38 have empty closure, 13 use only propext, and 28 use only propext and Quot.sound, with no project axiom or Classical.choice; the 80-line combined audit additionally covers the predecessor halt-separation theorem.	This milestone emits only the complete first canonical clause after the width header and retains the following coordinate as data. It does not implement a dynamic formula cursor or remaining formula-body emission, construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Literal Cook-Levin token-cursor padding step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete first-clause prefix with a total nine-symbol bridge and a fixed 45-rule cursor-advance table under disjoint nested state images. Its table has exactly 1192 plus the four inherited unary-evaluator rule counts. Every raw input follows an exact all-input trace, preserves the complete first-clause output, and advances the retained unary token coordinate from FormulaVariableSlotBound + 12 to FormulaVariableSlotBound + 13. The direct decoder proves that the consumed coordinate is the first in-range padding slot, so the token-level specification cursor advances without emitting a token. The cursor suffix takes exactly 2*cursorWord.length + 8 work steps including launch, and the compiled run is bounded by FirstClausePrefix.rawTimeBound + 48 + 12*cursorWord.length. All 47 public declarations, including two reviewed dead-state dispatch facts used downstream, are axiom-audited: 14 have empty closure, seven use only propext, and 26 use only propext and Quot.sound, with no project axiom or Classical.choice.	This is one fixed padding-coordinate transition after the complete first clause. It is not a general dynamic cursor loop or raw decoder for arbitrary schedule coordinates, emits no additional formula token, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin first-clause remaining-padding run	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the preceding first-clause cursor step with structurally generated unary evaluators for D = (FormulaVariableSlotBound - 1) * (FormulaVariableSlotBound + 6) and the second-clause coordinate, plus one fixed 25-rule countdown controller. Its table has exactly 1244 plus the six inherited and generated evaluator rule counts. Every raw input follows an exact all-input trace, preserves the complete first-clause output, traverses all D remaining first-clause padding coordinates without emitting a token, and reaches FormulaVariableSlotBound + 1 + FormulaTokensPerClause, where direct schedule lookup is proved to return Sep. The recursive specification run agrees at every padding coordinate and the following specification step observes Sep. The compiled exact run has an explicit external input-size polynomial bound, accepts within that bound, and remains fail-closed for malformed countdown scratch/root states, the unlaunched prefix endpoint, and one-step-short fuel. All 83 public declarations plus one predecessor transport theorem are axiom-audited: 37 have empty closure, 11 use only propext, and 36 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone executes exactly the remaining first-clause padding block and identifies the second-clause separator boundary. It is not a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the second clause or remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-clause separator step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete first-clause padding run with a selected 59-rule Sep appender, two total nine-symbol bridges, and the existing fixed 45-rule cursor advance under collision-free nested state images. Its table has exactly 1366 plus the six inherited and generated unary-evaluator rule counts. Every raw input follows an exact all-input trace, emits the canonical token prefix through the Sep beginning clause two, and advances the retained unary coordinate by one. The emitted bits equal encodedFormula.take (2 * (FormulaWidth + 13)), and direct schedule lookup proves that the following token is F. The compiled run is bounded by BuilderFirstClausePaddingRun.rawTimeBound + 246 + 24*n + 12*FormulaWidth + 12*cursorWord.length. Malformed appender tally/output, malformed cursor scratch, both unlaunched endpoints, and one-step-short total fuel remain timeout. All 54 new public declarations plus two reviewed predecessor dispatch facts are axiom-audited: 15 have empty closure, 11 use only propext, and 30 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits only the fixed populated Sep transition beginning clause two and advances to the following F coordinate. It is not a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit that F or the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-clause first-literal prefix	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete second-clause-separator prefix with two selected 59-rule F appenders, four total symbol-preserving bridges, and two copies of the existing 45-rule cursor advance under collision-free nested state images. One F/cursor component has 113 rules, the two-component suffix has 235 rules, and the global table has exactly 1610 plus the six inherited and generated unary-evaluator rule counts. Every raw input follows an exact all-input trace, emits the canonical prefix through the complete negative literal on variable zero in clause two, and retains the negative-sign coordinate on variable one. The emitted bits equal encodedFormula.take (2 * (FormulaWidth + 15)); direct schedule lookup proves the sign, unary-zero terminator, and following sign are all F. The compiled run is bounded by BuilderSecondClauseSeparatorStep.rawTimeBound + 564 + 48*n + 24*FormulaWidth + 24*cursorWord.length. All four pre-launch endpoints, malformed tally/output in both appenders, malformed scratch in both cursors, and one-step-short fuel remain timeout. All 85 new public declarations plus two reviewed cursor dead-loop facts are axiom-audited: 25 have empty closure, 18 use only propext, and 44 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits only the fixed negative literal on variable zero in clause two and advances to the following negative-sign coordinate. It does not complete clause two, implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, emit the remaining formula body, construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-clause second-literal prefix	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete clause-two first-literal prefix with selected 59-rule F, T, and F appenders, six total symbol-preserving bridges, and three copies of the existing 45-rule cursor advance under collision-free nested state images. The selected T/cursor component has 113 rules, the T/F tail has 235 rules, the complete three-token suffix has 357 rules, and the global table has exactly 1976 plus the six inherited and generated unary-evaluator rule counts. Every raw input follows an exact all-input trace, emits the canonical prefix through the complete negative literal on variable one in clause two, and retains the following Finish coordinate. The emitted bits equal encodedFormula.take (2 * (FormulaWidth + 18)); direct schedule lookup proves the sign F, unary unit T, terminator F, and following Finish. The compiled run is bounded by BuilderSecondClause-FirstLiteralPre-fix.rawTimeBound + 1026 + 72*n + 36*FormulaWidth + 36*cursorWord.length. All six pre-launch endpoints, malformed tally/output in all three appenders, malformed scratch in all three cursors, and one-step-short fuel remain timeout. All 113 new public declarations plus two reviewed cursor dead-loop facts are axiom-audited: 34 have empty closure, 25 use only propext, and 56 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits only the fixed negative literal on variable one in clause two and advances to the following Finish coordinate. It does not emit the clause terminator, complete clause two, implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, emit the remaining formula body, construct a complete formula builder or FunctionPro-gram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin complete second-clause prefix	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete clause-two second-literal prefix with a selected 59-rule Finish appender, two total symbol-preserving bridges, and one copy of the existing 45-rule cursor advance under collision-free state images. The fixed suffix has exactly 113 rules and the global table has exactly 2098 plus the six inherited and generated unary-evaluator rule counts. Every raw input follows an exact all-input trace, emits the canonical prefix through the complete second clause, and retains its first in-range padding coordinate. The emitted bits equal encodedFormula.take (2 * (FormulaWidth + 19)); direct schedule lookup proves the executed opportunity is Finish and the retained next opportunity is padding. The compiled run is bounded by BuilderSecondClauseSecondLiteralPrefix.rawTimeBound + 390 + 24*n + 12*FormulaWidth + 12*cursorWord.length. Both pre-launch endpoints, malformed appender tally/output, malformed cursor scratch, and one-step-short fuel remain timeout. All 55 new public declarations plus two reviewed cursor dead-loop facts are axiom-audited: 15 have empty closure, 10 use only propext, and 32 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits only the fixed Finish terminator that completes clause two and advances to its first padding coordinate. It does not traverse clause-two padding, reach clause three, implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, emit the remaining formula body, construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-clause remaining-padding run	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete second-clause prefix with structurally generated unary evaluators for D = (FormulaVariableSlotBound - 1) * (FormulaVariableSlotBound + 6) + 5 and the third-clause coordinate, plus the reused fixed 25-rule PaddingCountdown controller. Three total symbol-preserving WorkChain bridges yield a table with exactly 2150 plus the six inherited/generated predecessor evaluator rule counts and the two new evaluator rule counts. Every raw input follows an exact all-input trace, preserves the complete second-clause output, traverses all D = FormulaTokensPerClause - 7 remaining second-clause padding coordinates without emitting a token, and reaches FormulaVariableSlotBound + 1 + 2 * FormulaTokensPerClause, where direct schedule lookup is proved to return Sep. The recursive specification run agrees at every padding coordinate and the following specification step observes Sep. The emitted bits remain encodedFormula.take (2 * (FormulaWidth + 19)). The compiled exact run is bounded by BuilderSecondClausePrefix.rawTimeBound + 18 + 6 * countEvaluator.workSteps + 6 * (D * (2 * countRootPrefixLength + 8) + D * D) + 6 * targetEvaluator.workSteps, accepts within that external input-size polynomial, and remains fail-closed for malformed countdown scratch/root states, the unlaunched	This milestone executes exactly the remaining second-clause padding block and identifies the third-clause separator boundary without emitting a token. It reaches clause three only as a retained coordinate. It is not a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the third-clause separator or remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin third-clause separator step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete second-clause padding run with the reused selected 59-rule Sep appender, two total nine-symbol bridges, and the existing fixed 45-rule cursor advance under collision-free nested state images. Its table has exactly 2272 plus eight inherited unary-evaluator rule counts. Every raw input follows an exact all-input trace, emits the canonical token prefix through the Sep beginning clause three, and advances the retained unary coordinate by one. The emitted bits equal encodedFormula.take (2 * (FormulaWidth + 20)), and direct schedule lookup proves that the following token is F. The compiled run is bounded by BuilderSecondClausePaddingRun.rawTimeBound + 330 + 24*n + 12*FormulaWidth + 12*cursorWord.length. All 48 new public declarations plus eight reviewed suffix interfaces are axiom-audited: 14 have empty closure, 11 use only propext, and 31 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits only the fixed populated Sep transition beginning clause three and advances to the following F coordinate. It is not a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit that F or the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin third-clause first-literal prefix	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete third-clause-separator prefix with the reused 235-rule two-F appender/cursor suffix behind one total nine-symbol bridge under collision-free nested state images. One F/cursor component has 113 rules, the two-component suffix has 235 rules, and the global table has exactly 2516 plus the eight inherited and generated unary-evaluator rule counts. Every raw input follows an exact all-input trace, emits the canonical prefix through the complete negative literal on variable zero in clause three, and retains the negative-sign coordinate on variable two. The emitted bits equal encodedFormula.take (2 * (FormulaWidth + 22)); direct schedule lookup proves the sign, unary-zero terminator, and following sign are all F, with the third excluded pair constructively identified as variables zero and two. The compiled run is bounded by BuilderThirdClauseSeparatorStep.rawTimeBound + 732 + 48*n + 24*FormulaWidth + 24*cursorWord.length. All four pre-launch endpoints, malformed tally/output in both appenders, malformed scratch in both cursors, and one-step-short fuel remain timeout. All 74 new public declarations plus eleven reviewed suffix declarations and two reviewed cursor dead-loop facts are axiom-audited: 24 have empty closure, 18 use only propext, and 45 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits only the fixed negative literal on variable zero in clause three and advances to the following negative-sign coordinate. It does not emit that following F, complete clause three, implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, emit the remaining formula body, construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin third-clause second-literal prefix	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete third-clause-first-literal prefix with a fixed 479-rule F/T/T/F appender/cursor suffix behind one total nine-symbol bridge under collision-free nested state images. The nested component tables have 113, 235, 357, and 479 rules, and the global table has exactly 3004 plus the eight inherited and generated unary-evaluator rule counts. Every raw input follows an exact all-input trace, emits the canonical prefix through the complete negative literal on variable two in clause three, and retains the following Finish coordinate. The emitted bits equal encodedFormula.take (2 * (FormulaWidth + 26)); direct schedule lookup proves the sign F, both unary units T, the terminator F, and the following Finish. The compiled run is bounded by BuilderThirdClause-FirstLiteralPre-fix.rawTimeBound + 1752 + 96*n + 48*FormulaWidth + 48*cursorWord.length. All eight pre-launch endpoints, malformed tally/output in all four appenders, malformed scratch in all four cursors, and one-step-short fuel remain timeout. All 145 public declarations are axiom-audited: 46 have empty closure, 32 use only propext, and 67 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits only the fixed negative literal on variable two in clause three and advances to the following Finish coordinate. It does not emit the following Finish, complete clause three, implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, emit the remaining formula body, construct a complete formula builder or FunctionPro-gram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin complete third-clause prefix	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete clause-three second-literal prefix with a selected 59-rule Finish appender, two total symbol-preserving bridges, and one copy of the existing 45-rule cursor advance under collision-free state images. The fixed suffix has exactly 113 rules and the global table has exactly 3126 plus the eight inherited unary-evaluator rule counts. Every raw input follows an exact all-input trace, emits the canonical prefix through the complete third clause, and retains its first in-range padding coordinate. The emitted bits equal encodedFormula.take (2 * (FormulaWidth + 27)); direct schedule lookup proves the executed opportunity is Finish and the retained next opportunity is padding. The compiled run is bounded by BuilderThirdClauseSecondLiteralPrefix.rawTimeBound + 498 + 24*n + 12*FormulaWidth + 12*BuilderThirdClauseSeparatorStep.cursorWord.length. Both pre-launch endpoints, malformed appender tally/output, malformed cursor scratch, and one-step-short fuel remain timeout. All 55 new public declarations plus two reviewed cursor dead-loop facts are axiom-audited: 14 have empty closure, 10 use only propext, and 33 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits only the fixed Finish terminator that completes clause three and advances to its first padding coordinate. It does not traverse clause-three padding, implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, emit the remaining formula body, construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin third-clause remaining-padding run	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete third-clause prefix with structurally generated unary evaluators for D = (FormulaVariableSlotBound - 1) * (FormulaVariableSlotBound + 6) + 4 and the fourth-clause coordinate, plus the reused fixed 25-rule PaddingCountdown controller. Three total symbol-preserving WorkChain bridges yield a table with exactly 3178 plus the eight inherited/generated predecessor evaluator rule counts and the two new evaluator rule counts. Every raw input follows an exact all-input trace, preserves the complete third-clause output, traverses all D = FormulaTokensPerClause - 8 remaining third-clause padding coordinates without emitting a token, and reaches FormulaVariableSlotBound + 1 + 3 * FormulaTokensPerClause, where direct schedule lookup is proved to return Sep. The recursive specification run agrees at every padding coordinate and the following specification step observes Sep. The emitted bits remain encodedFormula.take (2 * (FormulaWidth + 27)). The compiled exact run is bounded by BuilderThirdClausePrefix.rawTimeBound + 18 + 6 * countEvaluator.workSteps + 6 * (D * (2 * countRootPrefixLength + 8) + D * D) + 6 * targetEvaluator.workSteps, accepts within that external input-size polynomial, and remains fail-closed for malformed countdown scratch/root states, the unlaunched	This milestone executes exactly the remaining third-clause padding block and identifies the fourth-clause separator boundary without emitting a token. It reaches clause four only as a retained coordinate. It is not a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the fourth-clause separator or remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin fourth-clause separator step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete third-clause padding run with the reused selected 59-rule Sep appender and 45-rule cursor advance through one outer total nine-symbol WorkChain bridge. The global table has exactly 3300 plus the ten inherited/generated unary-evaluator rule counts. Every raw input follows exact prefix, appender, cursor, bridge, suffix, and combined traces; emits exactly the fourth-clause Sep; preserves encodedFormula.take (2 * (FormulaWidth + 28)); and retains coordinate FormulaVariableSlotBound + 1 + 3 * FormulaTokensPerClause + 1, whose direct next schedule token is F. The external compiled bound evaluates to BuilderThirdClausePaddingRun.rawTimeBound + 426 + 24 * n + 12 * FormulaWidth + 12 * cursorWord.length. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, malformed-workspace, both unlaunched-endpoint, and one-step-short results are proved. All 48 new public declarations plus eight reviewed reused separator/cursor interfaces are axiom-audited: 14 have empty closure, 11 use only propext, and 31 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits exactly one token: the fixed Sep that starts clause four. It observes but does not emit the following F, does not complete clause four, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin fourth-clause first-literal prefix	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete fourth-clause separator prefix with the reused 357-rule F/T/F appender/cursor suffix through one outer total nine-symbol WorkChain bridge. The global table has exactly 3666 plus the ten inherited/generated unary-evaluator rule counts. Every raw input follows exact prefix, appender, cursor, bridge, suffix, and combined traces; emits the complete first negative literal on variable one in clause four; preserves encodedFormula.take (2 * (FormulaWidth + 31)); and retains coordinate FormulaVariableSlotBound + 1 + 3 * FormulaTokensPerClause + 4, whose direct next schedule token is F. The external compiled bound evaluates to BuilderFourthClauseSeparatorStep.rawTimeBound + 1422 + 72 * n + 36 * FormulaWidth + 36 * cursorWord.length. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, malformed-workspace, all six unlaunched-endpoint, and one-step-short results are proved. All 97 new public declarations, 16 reviewed reused suffix interfaces, and two cursor dead-state facts are axiom-audited: 33 have empty closure, 25 use only propext, and 57 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits exactly the fixed first negative literal on variable one in clause four. It observes but does not emit the following second-literal F, does not complete clause four, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin fourth-clause second-literal prefix	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete fourth-clause first-literal prefix with the reused 479-rule F/T/T/F appender/cursor suffix through one outer total nine-symbol WorkChain bridge. The global table has exactly 4154 plus the ten inherited/generated unary-evaluator rule counts. Every raw input follows exact prefix, appender, cursor, bridge, suffix, and combined traces; emits the complete second negative literal on variable two in clause four; preserves encodedFormula.take (2 * (FormulaWidth + 35)); and retains coordinate FormulaVariableSlotBound + 1 + 3 * FormulaTokensPerClause + 8, whose direct next schedule token is Finish. The external compiled bound evaluates to BuilderFourthClauseFirstLiteralPrefix.rawTimeBound + 2232 + 96 * n + 48 * FormulaWidth + 48 * BuilderFourthClauseSeparatorStep.cursorWord.length. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, malformed-workspace, all eight unlaunched-endpoint, and one-step-short results are proved. All 124 new public declarations, 21 reviewed reused suffix interfaces, and two cursor dead-state facts are axiom-audited: 46 have empty closure, 32 use only propext, and 69 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits exactly the fixed second negative literal on variable two in clause four. It observes but does not emit the following Finish, does not complete clause four, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin complete fourth-clause prefix	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the fourth-clause second-literal prefix with a selected 59-rule Finish appender, one existing 45-rule cursor advance, and two total nine-symbol WorkChain bridges. The selected suffix has exactly 113 rules and the global table has exactly 4276 plus the ten inherited/generated unary-evaluator rule counts. Every raw input follows exact prefix, appender, cursor, bridge, suffix, and combined traces; emits the Finish that completes clause four; preserves encodedFormula.take (2 * (FormulaWidth + 36)); and retains coordinate FormulaVariableSlotBound + 1 + 3 * FormulaTokensPerClause + 9, whose direct next schedule token is padding. The external compiled bound evaluates to BuilderFourthClauseSecondLiteralPrefix.rawTimeBound + 618 + 24 * n + 12 * FormulaWidth + 12 * BuilderFourthClauseSeparatorStep.cursorWord.length. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, malformed-workspace, both unlaunched-endpoint, and one-step-short results are proved. All 55 new public declarations and two cursor dead-state facts are axiom-audited: 14 have empty closure, 10 use only propext, and 33 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits exactly the fixed Finish terminator that completes clause four and advances to its first padding coordinate. It does not traverse clause-four padding, implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, emit the remaining formula body, construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin fourth-clause remaining-padding run	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete fourth-clause prefix with two structurally generated unary evaluators, the reused 25-rule PaddingCountdown machine, and three total nine-symbol WorkChain bridges. Its table has exactly 4328 plus twelve inherited/generated unary-evaluator rule counts. For every raw bitstring it traverses exactly FormulaTokensPerClause - 9 padding opportunities without emitting a token, preserves encodedFormula.take (2 * (FormulaWidth + 36)), and retains coordinate FormulaVariableSlotBound + 1 + 4 * FormulaTokensPerClause. Direct schedule lookup and the specification cursor both prove that this first opportunity in the intentionally empty fifth fixed-width clause slot is padding. The external compiled bound is BuilderFourthClausePrefix.rawTimeBound + 18 + six times the count-evaluator work, countdown bound, and target-evaluator work. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, malformed-countdown, unlaunched-predecessor, and one-step-short results are proved. All 65 new public declarations and three reused countdown interfaces are axiom-audited: 26 have empty closure, 9 use only propext, and 33 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone traverses only the remaining padding in clause four and retains the first opportunity in the intentionally empty fifth clause rectangle. It does not traverse that empty rectangle, reach the next constraint, emit another token, implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, emit the remaining formula body, construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin empty fifth-clause padding run	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the fourth-clause padding predecessor with two structurally generated unary evaluators, the reused 25-rule PaddingCountdown machine, and three total nine-symbol WorkChain bridges. Its table has exactly 4380 plus fourteen inherited/generated unary-evaluator rule counts. For every raw bitstring it traverses exactly FormulaTokensPerClause padding opportunities in the intentionally empty fifth fixed-width clause rectangle without emitting a token, preserves encodedFormula.take (2 * (FormulaWidth + 36)), and retains coordinate FormulaVariableSlotBound + 1 + 5 * FormulaTokensPerClause. Direct schedule lookup and the specification cursor prove that every traversed fifth-slot opportunity and the first opportunity in the intentionally empty sixth slot are padding. The external compiled bound is BuilderFourthClausePaddingRun.rawTimeBound + 18 + six times the count-evaluator work, countdown bound, and target-evaluator work. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, malformed-countdown, unlaunched-predecessor, and one-step-short results are proved. All 65 new public declarations and three reused countdown interfaces are axiom-audited: 28 have empty closure, 9 use only propext, and 31 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone traverses only the intentionally empty fifth clause rectangle and retains the first opportunity in the intentionally empty sixth clause rectangle. It does not traverse that sixth rectangle, reach the next constraint, emit another token, implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, emit the remaining formula body, construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin remaining first-constraint padding run	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the fifth-clause padding predecessor with two structurally generated unary evaluators, the reused 25-rule PaddingCountdown machine, and three total nine-symbol WorkChain bridges. Its table has exactly 4432 plus sixteen inherited/generated unary-evaluator rule counts. For every raw bitstring it traverses exactly (FormulaVariableSlotBound - 2) * (FormulaVariableSlotBound + 2) * FormulaTokensPerClause = (FormulaClauseSlotsPerConstraint - 5) * FormulaTokensPerClause remaining empty token opportunities of the first scheduled constraint without emitting a token, preserves encodedFormula.take (2 * (FormulaWidth + 36)), and retains coordinate FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause. Direct schedule lookup and the specification cursor prove that every traversed opportunity is padding and the endpoint is the Sep beginning the second scheduled constraint. The external compiled bound is BuilderFifthClausePaddingRun.rawTimeBound + 18 + six times the count-evaluator work, countdown bound, and target-evaluator work. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, malformed-countdown, unlaunched-predecessor, and one-step-short results are proved. All 65 new public declarations and three reused countdown interfaces are axiom-audited: 26 have empty closure, 9 use only	This milestone traverses only the remaining empty clause rectangles of the first scheduled constraint and retains the Sep beginning the second scheduled constraint. It observes but does not emit that separator, does not emit the next constraint's first literal, implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, emit the remaining formula body, construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-constraint separator step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete first-constraint padding run with the reused selected 59-rule Sep appender and 45-rule cursor advance through one outer total nine-symbol WorkChain bridge. The global table has exactly 4554 plus the sixteen inherited/generated unary-evaluator rule counts. Every raw input follows exact prefix, appender, cursor, bridge, suffix, and combined traces; emits exactly the Sep beginning the second scheduled constraint; preserves encodedFormula.take (2 * (FormulaWidth + 37)); and retains coordinate FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause + 1, whose direct next schedule token is T. The external compiled bound evaluates to BuilderFirstConstraintPaddingRun.rawTimeBound + 534 + 24 * n + 12 * FormulaWidth + 12 * cursorWord.length. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, malformed-workspace, both unlaunched-endpoint, and one-step-short results are proved. All 48 new public declarations plus eight reviewed reused separator/cursor interfaces are axiom-audited: 14 have empty closure, 11 use only propext, and 31 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits exactly one token: the fixed Sep that starts the second scheduled constraint. It observes but does not emit the following T, does not emit the first literal or traverse the second constraint, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-constraint first-literal sign step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete second-constraint separator step with the reused selected 59-rule T appender and 45-rule cursor advance through one outer total nine-symbol WorkChain bridge. The global table has exactly 4676 plus the sixteen inherited/generated unary-evaluator rule counts. Every raw input follows exact prefix, appender, cursor, bridge, suffix, and combined traces; emits exactly the positive sign beginning the second scheduled constraint's first literal; preserves encodedFormula.take (2 * (FormulaWidth + 38)); and retains coordinate FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause + 2, whose direct next schedule token is the first unary T of a nonzero variable index. The external compiled bound evaluates to BuilderSecondConstraintSeparatorStep.rawTimeBound + 546 + 24 * n + 12 * FormulaWidth + 12 * cursorWord.length. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, malformed-workspace, both unlaunched-endpoint, and one-step-short results are proved. All 48 new public declarations plus eight reviewed reused true-token/cursor interfaces are axiom-audited: 14 have empty closure, 11 use only propext, and 31 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits exactly one token: the fixed positive T sign that begins the second scheduled constraint's first literal. It observes but does not emit the following unary T, does not complete that literal or traverse the second constraint, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-constraint first-literal first unary-unit step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete second-constraint first-literal sign step with the reused selected 59-rule T appender and 45-rule cursor advance through one outer total nine-symbol WorkChain bridge. The global table has exactly 4798 plus the sixteen inherited/generated unary-evaluator rule counts. Every raw input follows exact prefix, appender, cursor, bridge, suffix, and combined traces; emits exactly the first unary T of the second scheduled constraint's first variable index; preserves encodedFormula.take (2 * (FormulaWidth + 39)); and retains coordinate FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause + 3. A constructive schedule proof establishes that the index is at least three, so the direct next schedule token is the second unary T. The external compiled bound evaluates to BuilderSecondConstraintFirstLiteralSignStep.rawTimeBound + 558 + 24 * n + 12 * FormulaWidth + 12 * cursorWord.length. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, malformed-workspace, both unlaunched-endpoint, and one-step-short results are proved. All 48 new public declarations plus eight reviewed reused true-token/cursor interfaces are axiom-audited: 14 have empty closure, 11 use only propext, and 31 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits exactly one token: the first unary T of the second scheduled constraint's first variable index. It observes but does not emit the following second unary T, does not complete that literal or traverse the second constraint, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-constraint first-literal second unary-unit step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete second-constraint first-literal first-unary-unit step with the reused selected 59-rule T appender and 45-rule cursor advance through one outer total nine-symbol WorkChain bridge. The global table has exactly 4920 plus the sixteen inherited/generated unary-evaluator rule counts. Every raw input follows exact prefix, appender, cursor, bridge, suffix, and combined traces; emits exactly the second unary T of the second scheduled constraint's first variable index; preserves encodedFormula.take (2 * (FormulaWidth + 40)); and retains coordinate FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause + 4. A constructive schedule proof establishes that the index is at least three, so the direct next schedule token is the third unary T. The external compiled bound evaluates to BuilderSecondConstraintFirstLiteralFirstUnaryUnitStep.rawTimeBound + 570 + 24 * n + 12 * FormulaWidth + 12 * cursorWord.length. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, malformed-workspace, both unlaunched-endpoint, and one-step-short results are proved. All 48 new public declarations plus eight reviewed reused true-token/cursor interfaces are axiom-audited: 14 have empty closure, 11 use only propext, and 31 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits exactly one token: the second unary T of the second scheduled constraint's first variable index. It observes but does not emit the following third unary T, does not complete that literal or traverse the second constraint, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-constraint first-literal third unary-unit step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete second-constraint first-literal second-unary-unit step with the reused selected 59-rule T appender and 45-rule cursor advance through one outer total nine-symbol WorkChain bridge. The global table has exactly 5042 plus the sixteen inherited/generated unary-evaluator rule counts. Every raw input follows exact prefix, appender, cursor, bridge, suffix, and combined traces; emits exactly the third and final unary T of the second scheduled constraint's first variable index; preserves encodedFormula.take (2 * (FormulaWidth + 41)); and retains coordinate FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause + 5. A constructive schedule case split proves the selected variable index is exactly three in both the width-one head-variable and wider position-one blank-symbol branches, so the direct next schedule token is the terminating F. The external compiled bound evaluates to BuilderSecondConstraintFirstLiteralSecondUnaryUnitStep.rawTimeBound + 582 + 24 * n + 12 * FormulaWidth + 12 * cursorWord.length. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, malformed-workspace, both unlaunched-endpoint, and one-step-short results are proved. All 48 new public declarations plus eight reviewed reused true-token/cursor interfaces are axiom-audited: 14 have empty closure, 11 use only propext, and 31 use only propext and Quot.sound,	This milestone emits exactly one token: the third and final unary T of the second scheduled constraint's first variable index. It observes but does not emit the following terminating F, does not complete that literal or traverse the second constraint, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-constraint first-literal terminator step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete second-constraint first-literal third-unary-unit step with the reused selected 59-rule F appender and 45-rule cursor advance through one outer total nine-symbol WorkChain bridge. The global table has exactly 5164 plus the sixteen inherited/generated unary-evaluator rule counts. Every raw input follows exact prefix, appender, cursor, bridge, suffix, and combined traces; emits exactly the terminating F of the second scheduled constraint's first literal; preserves encodedFormula.take (2 * (FormulaWidth + 42)); and retains coordinate FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause + 6. A constructive schedule case split proves the direct next schedule token is Finish when tapeWidth is one and the positive T beginning the next literal at wider widths. The external compiled bound evaluates to BuilderSecondConstraintFirstLiteralThirdUnaryUnitStep.rawTimeBound + 594 + 24 * n + 12 * FormulaWidth + 12 * cursorWord.length. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, malformed-workspace, both unlaunched-endpoint, and one-step-short results are proved. All 48 new public declarations plus eight reviewed reused false-token/cursor and dead-state interfaces are axiom-audited: 14 have empty closure, 11 use only propext, and 31 use only propext and Quot.sound, with no project axiom or Classical.choice.	This milestone emits exactly one token: the terminating F of the second scheduled constraint's first literal. It observes but does not emit the following Finish in the width-one case or the following positive T in wider cases, does not emit the remainder of the second constraint or traverse that constraint, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-constraint first-literal successor token step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete second-constraint first-literal terminator with the represented-width unary evaluator, one 93-rule finite width branch that enters a single reused 59-rule token appender at either Finish or T, and the retained-coordinate unary evaluator through three total nine-symbol WorkChain bridges. The global table has exactly 5284 plus the eighteen inherited/generated unary-evaluator rule counts. Every raw input follows exact predecessor, width-evaluator, branch/appender, retained-coordinate, bridge, suffix, and combined traces; emits Finish exactly when tapeWidth is one and T at every wider width; preserves encodedFormula.take (2 * (FormulaWidth + 43)); and retains coordinate FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause + 7. Direct lookup and the specification cursor prove the following opportunity is padding at width one and unary T at wider widths. The external compiled bound evaluates to BuilderSecondConstraintFirstLiteralTerminatorStep.rawTimeBound + 600 + 24 * n + 12 * FormulaWidth + 12 * width + 12 * widthRootPrefixLength + 6 * widthWorkSteps + 6 * targetWorkSteps. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, predecessor-unlaunched-endpoint, and one-step-short results are proved while component malformed-workspace behavior remains fail-closed. All 80 new public declarations plus two strengthened	This milestone emits exactly one width-selected token after the terminating F of the second scheduled constraint's first literal: Finish at width one or positive T at wider widths. It observes but does not emit the following padding opportunity at width one or unary T at wider widths, does not emit the remainder of the second constraint or traverse that constraint, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-constraint padding-or-unary opportunity step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete second-constraint first-literal successor-token predecessor with the represented-width unary evaluator, one 93-rule finite optional-appender table, and the retained-coordinate unary evaluator through three total nine-symbol WorkChain bridges. The global table has exactly 5404 plus the twenty inherited/generated unary-evaluator rule counts. Every raw input follows exact predecessor, width-evaluator, width-one skip or wider-width T-appender, retained-coordinate, bridge, suffix, and combined traces. At tapeWidth one it consumes padding and emits no token; at every wider width it appends exactly the first unary T of the second literal. The output equals encodedFormula.take (2 * (FormulaWidth + 43 + if tapeWidth = 1 then 0 else 1)) and the retained coordinate is FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause + 8. Direct lookup and the specification cursor prove the following slot is again padding at width one and the second unary T at wider widths. The external compiled bound evaluates to BuilderSecondConstraintFirstLiteralSuccessorTokenStep.rawTimeBound + 612 + 24 * n + 12 * FormulaWidth + 12 * width + 12 * widthRootPrefixLength + 6 * widthWorkSteps + 6 * targetWorkSteps. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, predecessor-unlaunched-endpoint, and one-step-short results are	This milestone consumes exactly one width-selected schedule opportunity after the second scheduled constraint first literal: padding with no emitted token at width one or the first unary T of the second literal at wider widths. It observes but does not consume the following padding opportunity at width one or second unary T at wider widths, does not complete the second literal or traverse the remainder of the second constraint, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-constraint second padding-or-unary opportunity step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete first padding-or-unary-opportunity predecessor with the represented-width unary evaluator, the same reviewed 93-rule finite optional-appender table, and the retained-coordinate unary evaluator through three total nine-symbol WorkChain bridges. The global table has exactly 5524 plus the twenty-two inherited/generated unary-evaluator rule counts. Every raw input follows exact predecessor, width-evaluator, width-one skip or wider-width T-appender, retained-coordinate, bridge, suffix, and combined traces. At tapeWidth one it consumes padding and emits no token; at every wider width it appends exactly the second unary T of the second literal. The output equals encodedFormula.take (2 * (FormulaWidth + 43 + if tapeWidth = 1 then 0 else 2)) and the retained coordinate is FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause + 9. Direct lookup and the specification cursor prove the following slot is again padding at width one and the third unary T at wider widths. The external compiled bound evaluates to BuilderSecondConstraintPaddingOrUnaryOpportunityStep.rawTimeBound + 624 + 24 * n + 12 * FormulaWidth + 12 * width + 12 * widthRootPrefixLength + 6 * widthWorkSteps + 6 * targetWorkSteps. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, predecessor-unlaunched-endpoint, and	This milestone consumes exactly one additional width-selected schedule opportunity after the second scheduled constraint first literal: padding with no emitted token at width one or the second unary T of the second literal at wider widths. It observes but does not consume the following padding opportunity at width one or third unary T at wider widths, does not complete the second literal or traverse the remainder of the second constraint, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-constraint third padding-or-unary opportunity step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete second padding-or-unary-opportunity predecessor with the represented-width unary evaluator, the same reviewed 93-rule finite optional-appender table, and the retained-coordinate unary evaluator through three total nine-symbol WorkChain bridges. The global table has exactly 5644 plus the twenty-four inherited/generated unary-evaluator rule counts. Every raw input follows exact predecessor, width-evaluator, width-one skip or wider-width T-appender, retained-coordinate, bridge, suffix, and combined traces. At tapeWidth one it consumes padding and emits no token; at every wider width it appends exactly the third unary T of the second literal. The output equals encodedFormula.take (2 * (FormulaWidth + 43 + if tapeWidth = 1 then 0 else 3)) and the retained coordinate is FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause + 10. Direct lookup and the specification cursor prove the following slot is again padding at width one and the fourth unary T at wider widths. The external compiled bound evaluates to BuilderSecondConstraintSecondPaddingOrUnaryOpportunityStep.rawTimeBound + 636 + 24 * n + 12 * FormulaWidth + 12 * width + 12 * widthRootPrefixLength + 6 * widthWorkSteps + 6 * targetWorkSteps. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, predecessor-unlaunched-endpoint, and	This milestone consumes exactly one additional width-selected schedule opportunity after the second scheduled constraint first literal: padding with no emitted token at width one or the third unary T of the second literal at wider widths. It observes but does not consume the following padding opportunity at width one or fourth unary T at wider widths, does not complete the second literal or traverse the remainder of the second constraint, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-constraint fourth padding-or-unary opportunity step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete third padding-or-unary-opportunity predecessor with the represented-width unary evaluator, the same reviewed 93-rule finite optional-appender table, and the retained-coordinate unary evaluator through three total nine-symbol WorkChain bridges. The global table has exactly 5764 plus the twenty-six inherited/generated unary-evaluator rule counts. Every raw input follows exact predecessor, width-evaluator, width-one skip or wider-width T-appender, retained-coordinate, bridge, suffix, and combined traces. At tapeWidth one it consumes padding and emits no token; at every wider width it appends exactly the fourth unary T of the second literal. The output equals encodedFormula.take (2 * (FormulaWidth + 43 + if tapeWidth = 1 then 0 else 4)) and the retained coordinate is FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause + 11. Direct lookup and the specification cursor prove the following slot is padding at width one and the terminating F at wider widths. The external compiled bound evaluates to BuilderSecondConstraintThirdPaddingOrUnaryOpportunityStep.rawTimeBound + 648 + 24 * n + 12 * FormulaWidth + 12 * width + 12 * widthRootPrefixLength + 6 * widthWorkSteps + 6 * targetWorkSteps. Compiled exact, bounded, blank-equivalent, accepting, non-timeout, predecessor-unlaunched-endpoint, and	This milestone consumes exactly one additional width-selected schedule opportunity after the second scheduled constraint first literal: padding with no emitted token at width one or the fourth unary T of the second literal at wider widths. It observes but does not consume the following padding opportunity at width one or terminating F at wider widths, does not complete the second literal or traverse the remainder of the second constraint, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-constraint fifth padding-or-terminator opportunity step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete fourth padding-or-unary-opportunity predecessor with the represented-width unary evaluator, a new reviewed 93-rule finite optional-terminator table over the audited token-appender rules, and the retained-coordinate unary evaluator through three total nine-symbol WorkChain bridges. The global table has exactly 5884 plus the twenty-eight inherited/generated unary-evaluator rule counts. Every raw input follows exact predecessor, width-evaluator, width-one skip or wider-width F-appender, retained-coordinate, bridge, suffix, and combined traces. At tapeWidth one it consumes padding and emits no token; at every wider width it appends exactly the terminating F of the second literal. The output equals encodedFormula.take (2 * (FormulaWidth + 43 + if tapeWidth = 1 then 0 else 5)) and the retained coordinate is FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause + 12. Direct lookup and the specification cursor prove the following slot is padding at width one and the opening unary T of the following literal at wider widths. The external compiled bound evaluates to BuilderSecondConstraintFourthPaddingOrUnaryOpportunityStep.rawTimeBound + 660 + 24 * n + 12 * FormulaWidth + 12 * width + 12 * widthRootPrefixLength + 6 * widthWorkSteps + 6 * targetWorkSteps. Compiled exact, bounded,	This milestone consumes exactly one additional width-selected schedule opportunity after the second scheduled constraint first literal: padding with no emitted token at width one or the terminating F of the second literal at wider widths. It observes but does not consume the following padding opportunity at width one or opening unary T at wider widths, does not complete the following literal or traverse the remainder of the second constraint, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-constraint sixth padding-or-opening-unary opportunity step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete fifth padding-or-terminator-opportunity predecessor with the represented-width unary evaluator, the reviewed 93-rule finite optional-appender table, and the retained-coordinate unary evaluator through three total nine-symbol WorkChain bridges. The global table has exactly 6004 plus the thirty inherited/generated unary-evaluator rule counts. Every raw input follows exact predecessor, width-evaluator, width-one skip or wider-width opening-T appender, retained-coordinate, bridge, suffix, and combined traces. At tapeWidth one it consumes padding and emits no token; at every wider width it appends exactly the opening positive T of the following literal. The output equals encodedFormula.take (2 * (FormulaWidth + 43 + if tapeWidth = 1 then 0 else 6)) and the retained coordinate is FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause + 13. Direct lookup and the specification cursor prove the following slot is padding at width one and the first unary-index T of the following literal at wider widths. The external compiled bound evaluates to BuilderSecondConstraintFifthPaddingOrTerminatorOpportunityStep.rawTimeBound + 672 + 24 * n + 12 * FormulaWidth + 12 * width + 12 * widthRootPrefixLength + 6 * widthWorkSteps + 6 * targetWorkSteps. Compiled exact, bounded, blank-equivalent, accepting,	This milestone consumes exactly one additional width-selected schedule opportunity after the second scheduled constraint second literal: padding with no emitted token at width one or the opening positive T of the following literal at wider widths. It observes but does not consume the following padding opportunity at width one or first unary-index T at wider widths, does not complete the following literal or traverse the remainder of the second constraint, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Cook-Levin second-constraint seventh padding-or-unary opportunity step	formalized-foundation-only	For every fixed concrete polynomial-time verifier, one literal finite work machine composes the complete sixth padding-or-opening-unary-opportunity predecessor with the represented-width unary evaluator, the reviewed 93-rule finite optional-appender table, and the retained-coordinate unary evaluator through three total nine-symbol WorkChain bridges. The global table has exactly 6124 plus the thirty-two inherited/generated unary-evaluator rule counts. Every raw input follows exact predecessor, width-evaluator, width-one skip or wider-width first-unary-index-T appender, retained-coordinate, bridge, suffix, and combined traces. At tapeWidth one it consumes padding and emits no token; at every wider width it appends exactly the first unary-index T of the following literal. The output equals encodedFormula.take (2 * (FormulaWidth + 43 + if tapeWidth = 1 then 0 else 7)) and the retained coordinate is FormulaVariableSlotBound + 1 + FormulaClauseSlotsPerConstraint * FormulaTokensPerClause + 14. Direct lookup and the specification cursor prove the following slot is padding at width one and the second unary-index T of the following literal at wider widths. The external compiled bound evaluates to BuilderSecondConstraintSixthPaddingOrOpeningUnaryOpportunityStep.rawTimeBound + 684 + 24 * n + 12 * FormulaWidth + 12 * width + 12 * widthRootPrefixLength + 6 * widthWorkSteps + 6 * targetWorkSteps. Compiled exact bounded	This milestone consumes exactly one additional width-selected schedule opportunity after the second scheduled constraint following literal opening: padding with no emitted token at width one or the first unary-index T of the following literal at wider widths. It observes but does not consume the following padding opportunity at width one or second unary-index T at wider widths, does not complete the following literal or traverse the remainder of the second constraint, does not implement a general dynamic formula cursor or raw decoder for arbitrary schedule coordinates, does not emit the remaining formula body, and does not construct a complete formula builder or FunctionProgram.RawRefinement, package a concrete PolynomialReduction, establish CNFSAT NP-hardness or NP-completeness, establish CNFSAT in P, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Typed direct-wire NAND semantics	formalized	Typed topological Boolean NAND programs and ordered multi-output direct-wire semantics.	This does not establish circuit minimization, SAT, or P = NP.
Finite enumeration, equivalence, and reference minimum	formalized	Exhaustive finite Boolean direct-wire search under the empty-profile reference model.	No polynomial-runtime result is obtained from this exhaustive reference search.
Concrete framed replacement and slack	formalized	The concrete serial framed context with support outputs and explicit bypass wires.	This is not an arbitrary-support replacement theorem for the report's global family.
Locked-NAND local candidates and baseline accounting	formalized	Typed local macro candidates, source-derived counts, and five finite local square baselines.	Local minima are not a global BaselineDistinct or locked-NAND threshold theorem.
Locked-NAND global carrier and trace equivalence	formalized	Exact X/T/O/R/L/z carrier separation and both trace-equivalence directions for arbitrary finite topological NAND circuits.	Carrier/trace equivalence does not assemble the complete exposed candidates, prove cross-instance BaselineDistinct or final-output laws, construct the uniform polynomial builder, or establish the locked-NAND threshold.
Locked-NAND global baseline and four-gate candidate assembly	formalized	Exact source-derived B/B baseline and B+4/B+1 extension for arbitrary finite topological NAND circuits.	Candidate assembly alone does not prove global BaselineDistinct, either conditional final-output branch law, the locked-NAND threshold, residual slack at most four, or a uniform polynomial bitstring builder.
Locked-NAND global baseline distinctness	formalized	All exposed baseline outputs are nonconstant, nonprojections, pairwise semantically distinct, and have exact exhaustive reference minimum B for arbitrary finite topological NAND circuits.	BaselineDistinct does not prove either whole-carrier final-output branch law, instantiate the complete threshold premise package, establish the locked-NAND threshold or global residual-slack bound, or construct the uniform polynomial bitstring builder.
Locked-NAND global unsatisfiable final-zero branch	formalized	For every finite topologically ordered NAND circuit, unsatisfiability makes the full final coordinate identically false on the whole carrier and fixes the exhaustive reference minimum at B.	This does not prove satisfiable FinalLockSeparation, instantiate the complete threshold package, establish the global locked-NAND threshold or residual-slack bound, or construct the uniform polynomial builder.

Milestone	Status	Exact scope	Boundary
Locked-NAND satisfiable separation and typed semantic threshold	formalized	For every finite topologically ordered NAND circuit, one answer-independent full candidate instantiates all six semantic premises, has residual slack at most four, and crosses the exact source-derived minimum threshold exactly when the source circuit is satisfiable.	This typed semantic theorem does not construct or compile the report's encoded polynomial-time SAT-to-locked-NAND builder, establish CNFSAT in P, prove NP-hardness transport, discharge the abstract locked-NAND threshold axiom, or prove P = NP.
Encoded locked-NAND semantic boundary	formalized-semantic-boundary	A strict version-zero bit grammar round-trips normalized NAND circuits and complete locked-NAND candidates; the pure all-bitstring transformation is fail-closed and preserves source satisfiability at the exact target threshold.	This is not a parser/validator machine, emitter machine, RawRefinement, PolynomialReduction, construction-runtime or output-size bound, abstract locked-NAND threshold discharge, CNFSAT-in-P result, or P = NP.
Concrete strict-v0 locked-NAND source parser	formalized-foundation-only	One literal nine-symbol finite work machine validates every strict version-zero source bitstring: it accepts exactly ValidEncodedCircuit, preserves valid bytes, clears invalid bytes, cannot time out within the proved compiled cubic bound, and supplies polynomial-time machine/function witnesses plus the validator's exact leaf RawRefinement.	This source parser alone does not emit the locked-NAND target or establish the source-to-target PolynomialReduction. The downstream emitter now supplies its own runtime/output bounds and strict composition, but the abstract locked-NAND threshold assumption, CNFSAT-in-P result, NP-hardness or NP-completeness transport, and P = NP remain absent.
Concrete strict-v0 locked-NAND target emitter	formalized-foundation-only	One literal 1,387,921-rule grammar-only controller emits the exact direct locked-NAND target on every grammar-decoded circuit, rejects malformed grammar with empty output, cannot time out within an explicit all-input polynomial, has an explicit quadratic output-size bound, and supplies compiled polynomial-time machine/function witnesses, exact leaf RawRefinement, and strict parser/emitter composition computing buildLocked-NANDInstance.	The standalone emitter intentionally accepts every grammar-decoded raw circuit, including intrinsically invalid references; strict fail-closed semantics come from parser composition. The standalone emitter does not itself package the language equivalence as PolynomialReduction; the downstream concrete reduction milestone now does. The abstract locked-NAND threshold assumption, CNFSAT-in-P result, NP-hardness transport, and P = NP remain absent.

Milestone	Status	Exact scope	Boundary
Concrete strict-v0 locked-NAND polynomial reduction	formalized-polynomial-reduction	The existing strict parser/emitter composition is packaged as a concrete polynomial many-one reduction from EncodedNANDSAT to EncodedLocked-NANDThreshold, with exact function identity, exact output, all-bitstring language equivalence, a ReducesTo witness, and recursive raw-machine refinement.	The downstream all-input CNF compiler now identifies CNFSAT with this concrete source language through a fixed finite machine and a direct polynomial reduction, then composes with this reduction. This milestone does not discharge the abstract target-language assumption, prove the report-level locked-NAND threshold theorem, put CNFSAT in P, complete residual minimization or ZeroSlack, or prove P = NP.
Concrete CNF-to-NAND semantic compiler	formalized-semantic-boundary	A total answer-independent compiler transforms every strict canonical CNF formula into an intrinsically topological well-formed NAND circuit, preserves satisfiability exactly, proves the exact gate count and a quadratic serialized-output bound, fails closed on every malformed bitstring, and composes semantically with the concrete locked-NAND threshold builder.	This milestone is the pure semantic and size-bound layer; the subsequent all-input milestone supplies the finite-machine, PolynomialTimeFunction, RawRefinement, and PolynomialReduction interfaces. Neither layer decides CNF-SAT, proves CNFSAT is in deterministic polynomial time, discharges the abstract report-level locked-NAND premise, completes ZeroSlack/PCCMin, or proves P = NP.

Milestone	Status	Exact scope	Boundary
Concrete all-input CNF-to-NAND polynomial reduction	formalized-polynomial-reduction	One fixed 135,070-rule three-node parser/carrier/controller work graph halts on every bitstring, rejects malformed CNF words with empty output, emits exactly compileEncodedCNFToNAND on every valid source, has one external encoded-input polynomial, compiles to a non-timeout PolynomialTimeFunction, retains literal RawRefinement, packages a direct PolynomialReduction from CNFSAT to EncodedNANDSAT, and composes it with the strict locked-NAND reduction to EncodedLockedNANDThreshold.	This syntax-directed compiler does not itself decide CNF-SAT, put CNFSAT in deterministic polynomial time, establish SAT NP-hardness or CNFSAT NP-completeness, connect the concrete locked-NAND target to the abstract report-level threshold theorem, complete residual minimization or ZeroSlack/PCCMin, discharge any project assumption, or prove P = NP.
Conditional locked-NAND threshold boundary	formalized-with-premises	A proof-bearing six-premise candidate boundary for an arbitrary satisfiable proposition.	The premises are not instantiated by a uniform SAT-to-locked-NAND builder.
Fail-closed explicit-list residual routes	formalized-explicit-list-only	Executable strict-gain search over a caller-supplied finite implementation list.	Unresolved excludes no gain outside the supplied list and cannot imply ZeroSlack.
Universal verified residual-gain chain bound	formalized-iteration-bound-only	Every finite proof-bearing or executably verified chain of adjacent strict equivalent gains preserves semantics and the exhaustive reference minimum, while its endpoint residual slack plus its length is at most its starting residual slack. For the complete locked-NAND candidate, the existing residual-slack-at-most-four theorem specializes this to at most four verified gain steps.	This milestone bounds only a disclosed, independently verified sequence. It does not find the next gain, prove route or candidate-list completeness, justify stopping after fewer than the bound, construct ZeroSlack, compute an exact minimizer, establish polynomial checker or PCCMin runtime, put SAT in P, discharge a project assumption, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Global strict-gain stopping specification	formalized-semantic-stopping-only	For every finite direct-wire implementation, positive exhaustive-reference residual slack is equivalent to the existence of some strictly smaller semantically equivalent implementation; zero slack and semantic minimality are each equivalent to global absence of such an implementation. A verified chain endpoint with separately proved global no-gain evidence therefore has zero slack and packages an exact minimum result.	This is a semantic stopping criterion, not a stopping algorithm. It uses the exhaustive reference minimum as a mathematical witness and requires a proof quantifying over every finite implementation at the endpoint. It does not derive global absence from a finite scan, generate a route, prove candidate-list or route completeness, construct the manuscript's ZeroSlack certificate, establish polynomial checking or PCCMin runtime, put SAT in P, discharge a project assumption, or prove P = NP.
Terminal full-carrier residual bridge	formalized-terminal-full-mode-semantic-bridge	For every finite direct-wire implementation, terminalization preserves the exact whole implementation, gate count, and semantics at every input/output coordinate. An independently stated terminal minimum is attained, universally lower-bounds every complete terminal realization, and equals the exhaustive semantic reference minimum. Positive residual slack is equivalent to a cheaper whole-span full realization, every such realization gives strict residual descent, and zero slack is equivalent to absence of one.	This is the direct-wire terminal full-mode specialization of the manuscript bridge. It does not formalize the quotient carrier or quotient-to-full firewall, proper or governed supports, SaturatePositive, BCEL/BN2-BN6, packet or selector completeness, route generation, the ZeroSlack certificate, PCCMin, polynomial minimum search or checking, SAT in P, discharge a project assumption, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Terminal quotient/full mode firewall	formalized-terminal-mode-firewall	For every finite direct-wire implementation, a computed finite profile observer records the ten terminal carrier roles and an explicit forgetful projection selects the quotient coordinates. Projection retains the exact implementation, gate count, and complete multi-output Boolean semantics. A quotient comparison has a checked full lift exactly when every forgotten profile coordinate agrees, lossless projections lift directly, and obligation discharge transports across a checked lift.	This is a terminal comparison/lifting firewall only. It supplies no proper or governed supports, arbitrary quotient construction, support or projection-defect minimum, saturation, Package E, BCEL/BN2-BN6, packet or selector completeness, global residual route, ZeroSlack certificate, PCCMin exactness or polynomial runtime, SAT-in-P result, discharged project assumption, or proof that P = NP.
Terminal full/quotient projection minima	formalized-terminal-projection-minimum	For every finite direct-wire implementation, computed finite terminal-profile observer, and explicit forgetful projection, complete enumeration through the current gate count computes an attained full-profile minimum and an attained quotient-profile minimum. Both minima universally lower-bound every matching realization, forgetting coordinates cannot increase the minimum, the full minimum decomposes as the quotient minimum plus a nonnegative projection defect, and that defect is zero exactly when an attained quotient minimum has a checked full lift.	These are exhaustive finite reference minima through the supplied implementation size. This milestone proves no polynomial runtime, proper or governed support construction, arbitrary manuscript quotient carrier, SaturatePositive, Package E, BCEL/BN2-BN6, complete residual routing, ZeroSlack certificate, PCCMin exactness, SAT-in-P result, discharged project assumption, or proof that P = NP.

Milestone	Status	Exact scope	Boundary
Terminal projection transfer identity	formalized-terminal-projection-transfer	For every finite direct-wire four-corner terminal-profile family sharing one computed observer and one explicit projection, signed full and quotient minimum deltas obey the exact Section 5.2 transfer identity. The projection excess is the quotient delta minus the full delta; if meet and both side defects are zero while the join defect is D, the excess equals D and is positive whenever D is positive.	This is signed arithmetic over four supplied corners. It does not construct or certify a proper governed support square, prove SaturatePositive, discharge Package E or BCEL/BN2-BN6, generate a complete residual route, prove ZeroSlack or PCCMin, establish polynomial runtime, put SAT in P, remove a project assumption, or prove P = NP.
Terminal saturation closure	formalized-terminal-saturation-closure	For every finite terminal primitive-record universe and every explicit Boolean dependency system tagged by the manuscript's ten closure mechanisms, the generated reflexive transitive closure contains the seed, is dependency-closed, is least among closed supersets, is monotone and idempotent, and has exactly the closed supports as fixed points.	This closure theorem does not derive the dependency relation from an arbitrary circuit, construct proper support, prove support completion or square legitimacy, instantiate a projection-compatible square, prove SaturatePositive or BCELReady, discharge Package E or BCEL/BN2-BN6, generate a complete residual route, prove ZeroSlack or PCCMin, establish polynomial runtime, put SAT in P, remove a project assumption, or prove P = NP.
Terminal executable and physical support completion	formalized-terminal-physical-support-completion	For every finite direct-wire candidate, explicit terminal dependency system, and finite seed list, a deterministic finite work list computes exactly the inductive saturation, then the actual program computes canonically ordered incoming boundary and outgoing interface wires. Lean proves no crossing wire is omitted or added and the composed physical support is compatible.	The terminal dependency system remains explicit data rather than an extracted profile frontier. This milestone does not construct proper positive support, prove support completion in the manuscript's full sense or square legitimacy, instantiate the required projection square, prove SaturatePositive, discharge Package E or BCEL/BN2-BN6, generate a complete residual route, prove ZeroSlack or PCCMin, establish polynomial runtime, put SAT in P, remove a project assumption, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Arbitrary terminal support extraction	formalized-terminal-support-extraction	For every finite direct-wire candidate and finite terminal record list, including noncontiguous selections, the actual program is structurally extracted over its exact canonical incoming boundary and ordered outgoing interface. The extracted candidate equals an independently defined open-support function for every boundary valuation and recovers the original interface values on whole-circuit-induced boundaries; the construction also composes with executable terminal saturation.	The record list and terminal dependency system remain explicit inputs rather than the manuscript's derived profile frontier. This milestone does not construct a proper positive support, prove full governed support completion or square legitimacy, instantiate the required projection square, prove SaturatePositive, discharge Package E or BCEL/BN2-BN6, generate a complete residual route, prove ZeroSlack or PCCMin, establish polynomial runtime, put SAT in P, remove a project assumption, or prove P = NP.
Governed proper-positive terminal support search	formalized-governed-proper-positive-support-search	For every finite direct-wire candidate and explicit terminal dependency system, Lean enumerates the complete canonical finite universe of primitive-record seeds, saturates and physically completes each seed, extracts its exact open support, and computes exact local gain from the exhaustive semantic reference minimum. The search returns a proof-bearing nonempty proper support with positive gain whenever one exists in that canonical seed universe, and its none result is equivalent to the absence of such a seed.	The terminal dependency system remains explicit input rather than a profile frontier derived from the circuit, and the search is exhaustive reference computation rather than a polynomial algorithm. This milestone does not prove global gain completeness, full manuscript support completion or square legitimacy, instantiate a projection-compatible square, prove SaturatePositive or BCELReady, discharge Package E or BCEL/BN2-BN6, generate a complete residual route, prove ZeroSlack or PCCMin, put SAT in P, remove a project assumption, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Saturated terminal support-square closure	formalized-terminal-saturated-support-square-closure	For every finite direct-wire candidate, every explicit terminal dependency system, and every pair of finite terminal seeds, Lean computes saturated left and right corners, their canonical closed meet, and their closed saturated-union join. It proves the exact greatest-lower-bound and least-upper-bound laws, seed extensionality, computed physical compatibility, exact gate count, open-support semantics, and induced whole-circuit recovery for all four corners.	The terminal dependency system remains explicit input rather than a profile frontier derived from the circuit. This milestone proves finite closed-corner algebra and computed physical extraction, not the manuscript's obstruction routing, frontier pushout, projection-compatible square, side-tight four-corner minima, BN2 square legitimacy, SaturatePositive, Package E, BCEL/BN2-BN6, complete residual routing, ZeroSlack, PCCMin, polynomial runtime, SAT in P, removal of a project assumption, or P = NP.
Governed terminal support completion	formalized-terminal-governed-support-completion	For every finite direct-wire candidate, explicit terminal dependency system, finite seed list, and computed saturated support-square corner, Lean computes the exact physical boundary, ordered interface, and partition of selected profile coordinates among all ten terminal profile roles. It proves exact membership, no duplicates, pairwise disjointness, record coverage, dependency closure, physical compatibility, and retention of each exact corner.	The terminal dependency system remains explicit input rather than a profile frontier derived from the circuit. This milestone computes a governed finite completion of each saturated support-square corner, not the manuscript's obstruction routing, frontier pushout, projection-compatible square, side-tight four-corner minima, BN2 square legitimacy, SaturatePositive, Package E, BCEL/BN2-BN6, complete residual routing, ZeroSlack, PCCMin, polynomial runtime, SAT in P, removal of a project assumption, or P = NP.

Milestone	Status	Exact scope	Boundary
Governed terminal frontier pushout	formalized-terminal-governed-frontier-pushout	For every finite direct-wire candidate, explicit terminal dependency system, and computed saturated support square, Lean constructs the governed boundary, interface, and role-preserving profile pushout from the two side completions alone. The independently completed join frontier equals that gluing, the meet profile is the exact shared overlap, and every side physical coordinate is either retained externally or witnessed as internalized.	The terminal dependency system remains explicit input rather than a profile frontier derived from the circuit. This milestone proves exact frontier gluing for computed saturated support squares, not projection compatibility, side-tight four-corner minima, BN2 square legitimacy, SaturatePositive, Package E, BCEL/BN2-BN6, complete obstruction routing, ZeroSlack, PCCMin, polynomial runtime, SAT in P, removal of a project assumption, or P = NP.
Governed terminal projection square	formalized-terminal-governed-projection-square	For every finite direct-wire candidate, explicit terminal dependency system, computed saturated support square, and every forgetful terminal projection, Lean retains the exact physical frontier, filters all ten role profiles exactly, proves projected meet is the shared side overlap, and proves projected join is the side-only projected pushout without reading the join corner.	The terminal dependency system remains explicit input rather than a profile frontier derived from the circuit. This milestone proves structural projection commutation for computed saturated support squares, not side-tight four-corner minima, BN2 square legitimacy, SaturatePositive, Package E, BCEL/BN2-BN6, complete obstruction routing, ZeroSlack, PCCMin, polynomial runtime, SAT in P, removal of a project assumption, or P = NP.
Side-tight four-corner minimum arithmetic	formalized-residual-terminal-side-tight-minimum-arithmetic	For every finite terminal projection four-corner family and every independently attained typed full or quotient basis, Lean proves componentwise minimum bounds and the exact signed four-slack identity. A fail-closed Boolean and Option gate returns the corresponding existing delta only when meet, left, right, and join all attain their exact minima; both canonical independently attained minimum bases pass.	The canonical corner minima are independently attained. This milestone proves numerical arithmetic and fail-closed exactness, not construction of one coherent four-corner basis, coherent completion, maximization over a finite tight family, BN2 square legitimacy, SaturatePositive, Package E, BCELReady or BCEL/BN2-BN6, complete obstruction routing, ZeroSlack, PCCMin, polynomial runtime, SAT in P, removal of a project assumption, or P = NP.

Milestone	Status	Exact scope	Boundary
Checked four-corner carrier transport	formalized-residual-terminal-four-corner-carrier-transport	For every finite computed saturated terminal support square, direct-wire candidate, and forgetful terminal projection, Lean derives all four exact governed and extracted endpoints in common ambient coordinates, proves duplicate-free boundary, interface, and profile lists, transports meet and join profiles exactly, and classifies each present side physical coordinate as identically retained or constructively internalized through fail-closed queries.	This milestone supplies a checked common ambient carrier for the computed square. It does not transport four optimum realizers, prove the full four-corner optimum carrier-compatibility obligation, construct a coherent four-corner optimum, prove side-tight completion or BN2 square legitimacy, establish SaturatePositive, Package E, BCELReady or BCEL/BN2-BN6, complete obstruction routing, prove ZeroSlack, PCCMin, polynomial runtime, SAT in P, removal of a project assumption, or P = NP.
Four-corner optimum carrier compatibility	formalized-residual-terminal-four-corner-optimum-carrier-compatibility	For every finite computed saturated terminal support square and every explicit observer, Lean embeds all four exact corner candidates into one common ambient carrier, proves reversible semantic and gate-count preservation, proves exact ambient and corner reference minima agree, and localizes canonical full and quotient optima from one shared observer and projection without changing their exact minimum counts.	This milestone compares independently attained full and quotient optima on one reversible common carrier. It does not prove coherent transport along the square legs, construct a coherent four-corner optimum, prove sideTightCompletionExists or BN2 square legitimacy, derive the terminal dependency system, establish SaturatePositive, Package E, BCELReady or BCEL/BN2-BN6, complete obstruction routing, prove ZeroSlack, PCCMin, polynomial runtime, SAT in P, removal of a project assumption, or P = NP.
Four-corner optimum coherence dichotomy	formalized-residual-terminal-four-corner-optimum-coherence-dichotomy	For every finite computed terminal support square, every explicit observer, every terminal projection, and either full or quotient coherence mode, Lean checks the four square legs in a deterministic order and returns either one coherent canonical optimum tuple with exact transport, side-tight, and incidence facts or the exact deterministic first failure.	This milestone classifies coherent transport or its exact first failure. It does not prove that every square is coherent, construct the later no-outcome route, prove sideTightCompletionExists or BN2 square legitimacy, establish SaturatePositive, Package E, BCELReady or BCEL/BN2-BN6, complete obstruction routing, prove ZeroSlack, PCCMin, polynomial runtime, SAT in P, removal of a project assumption, or P = NP.

Milestone	Status	Exact scope	Boundary
Four-corner side-tight completion under local route silence	formalized-residual-terminal-four-corner-side-tight-completion-under-local-route-silence	For every finite computed terminal support square, every explicit observer, and either full or quotient coherence mode, the exact first local coherence query returns a proof-bearing sound route or, under computed local route silence, Lean supplies the complete checked side-tight coherent optimum tuple with exact minimum incidence value while retaining the separate quotient-promotion firewall.	This milestone closes only the local completion edge under computed local route silence. It does not prove universal route silence, connect a local obstruction to the complete global no-outcome route system, prove BN2 square legitimacy, derive the terminal dependency system, enumerate or maximize the complete tight-basis family, establish SaturatePositive, Package E, BCELReady or BCEL/BN2-BN6, complete obstruction routing, prove ZeroSlack, PCCMin, polynomial runtime, SAT in P, remove a project assumption, or prove P = NP.
Complete four-corner BN2 tight-basis maximum	formalized-residual-terminal-four-corner-complete-tight-basis-maximum	For every finite computed terminal support square, every explicit observer, and either full or quotient mode, Lean enumerates the complete finite tight-basis family, retains every exact profile-constrained minimum implementation at each corner, filters the full Cartesian product with the arbitrary-family coherence query, and proves under exact local route silence that the signed maximum equals the selected delta.	This milestone closes the remaining local BN2 tight-basis maximum under computed local route silence. It does not prove universal route silence, connect a local obstruction to the complete global no-outcome route system, prove BN2 square legitimacy, derive the terminal dependency system, establish SaturatePositive, Package E, BCELReady or BCEL/BN2-BN6, complete obstruction routing, prove ZeroSlack, PCCMin, polynomial runtime, SAT in P, remove a project assumption, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Computed terminal BN2 square legitimacy	formalized-residual-terminal-computed-bn2-square-legitimacy	For every finite computed terminal support square built from two finite seeds under one explicit terminal dependency system, direct-wire candidate, observer, and forgetful projection, Lean constructs the exact compatible governed frontier and projection square, keeps full and quotient minimum quantities on the same carrier, and returns either the complete local conclusion under exact local route silence or the deterministic full-then-quotient proof-bearing first coherence route.	This milestone packages computed structural legitimacy and the exact local no-route conclusion. It does not derive the terminal dependency system from an arbitrary circuit, prove universal route silence, connect a local failure to the complete global no-outcome route system, identify a BCEL anchor square, establish SaturatePositive, Package E, BCELReady or later BCEL/BN2-BN6 conclusions, prove ZeroSlack, PCCMin, polynomial runtime, SAT in P, remove a project assumption, or prove P = NP.
Computed terminal BCEL anchor nucleus and cut-square dichotomy	formalized-residual-terminal-computed-bcel-anchor-nucleus	For every finite direct-wire candidate, explicit terminal dependency system, computed governed proper-positive support, forgetful projection, executable ambient observer, and positive whole-support projection defect, Lean computes the canonical minimum-cardinality positive anchor nucleus and returns either an insufficient nucleus, the exact first anchor-algebra mismatch, the exact first proper-cut defect mismatch, the proof-bearing first full-before-quotient local route, or exact constant-cut and local BN2 conclusions for every proper cut.	This milestone assumes a positive whole-support projection defect and an explicit terminal dependency system. It does not derive either premise, identify manuscript activation or charge equivalence classes absent from the terminal model, connect a local failure to the complete global no-outcome route system, establish SaturatePositive, Package E, BCELReady or later BCEL/BN2-BN6 conclusions, prove ZeroSlack, PCCMin, polynomial runtime, SAT in P, remove a project assumption, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Computed terminal saturation-positivity firewall	formalized-residual-terminal-saturation-positivity-firewall	For every finite direct-wire candidate, explicit terminal dependency system, computed governed proper-positive support, forgetful projection, and executable ambient observer, Lean computes the whole-support defect: zero projection defect returns an attained quotient minimum with a checked full lift, while positive defect delegates exactly to the existing fail-closed BCEL anchor-nucleus classifier.	This closes only projectionPositivityNotLostSilently in the current finite terminal model. It assumes an explicit terminal dependency system and an already computed governed proper-positive support. It does not discharge transparentSaturationCostBalanced, interfaceExposureRoutesToE, originKernelObligationClosureRouted, or firstNontransparentStepRecorded; establish full SaturatePositive, Package E, BCELReady or later BCEL/BN2-BN6 conclusions; prove ZeroSlack, PCCMin, polynomial runtime, SAT in P; remove a project assumption; or prove P = NP.
Candidate-derived terminal saturation cost balance	formalized-residual-terminal-candidate-saturation-cost-balance	For every finite direct-wire candidate, executable ambient observer, forgetful projection, and finite terminal seed, Lean computes the candidate-derived dependency system and deterministic rule-labelled saturation trace, then returns proof that every event is exactly cost-balanced with preserved full slack and nondecreasing projection defect, or records the exact first nontransparent event and complete transparent prefix.	This closes only the finite terminal forms of transparentSaturationCostBalanced and firstNontransparentStepRecorded. The executable observer and forgetful projection remain explicit model inputs, and a nontransparent event is recorded rather than routed. It does not discharge interfaceExposureRoutesToE or originKernelObligationClosureRouted; establish full SaturatePositive, Package E, BCELReady or later BCEL/BN2-BN6 conclusions; prove ZeroSlack, PCCMin, polynomial runtime, SAT in P; remove a project assumption; or prove P = NP.

Milestone	Status	Exact scope	Boundary
Finite terminal interface-exposure routing	formalized-residual-terminal-interface-exposure-routing	For every finite direct-wire candidate, executable ambient observer, forgetful projection, and finite terminal seed, Lean recognizes only an exact candidate-derived interface-consumer edge. Each recognized event is transparently cost-balanced or produces a proof-bearing local E-route; the production trace result records the exact first nontransparent event and complete transparent prefix, while non-interface first failures remain fail-closed.	This closes only the finite local form of interfaceExposureRoutesToE. The proof-bearing local E-route is an exposure-obligation coordinate, not a full Package E VerifyDW acceptance, a verified global gain, or global route completeness. The executable observer and forgetful projection remain explicit model inputs. It does not discharge originKernelObligationClosureRouted; establish full SaturatePositive, Package E, BCELReady or later BCEL/BN2-BN6 conclusions; prove ZeroSlack, PCCMin, polynomial runtime, SAT in P; remove a project assumption; or prove P = NP.
Finite terminal SaturatePositive composition	formalized-residual-terminal-finite-saturate-positive-composition	For every finite direct-wire candidate, executable ambient observer, forgetful projection, and proof-bearing candidate BCEL anchor problem whose normalized seed has positive full slack, Lean recognizes exact candidate-derived origin, kernel, and obligation closures in both gate/profile orientations; checks cost transparency, obligation discharge, and forgotten-profile stability; preserves positive full slack across an all-safe trace into the checked-lift or BCEL firewall; or returns the exact first interface, closure, or other fail-closed nontransparent route with its complete safe prefix.	This closes the finite local form of originKernelObligationClosureRouted and composes the five reconstructed terminal sub-obligations only for an explicit proof-bearing problem. A local route is not a complete global outcome, Package E VerifyDW acceptance, verified gain, or global route-completeness result. The positive initial full-slack premise remains explicit. It does not establish manuscript-wide SaturatePositive, BCELReady, RankWF, ZeroSlack, PCCMin, polynomial runtime, SAT in P; remove a project assumption; or prove P = NP.

Milestone	Status	Exact scope	Boundary
Residual terminal RankWF	formalized-residual-terminal-rank-wf	For the fixed manuscript residual rank of exactly ten natural coordinates in the stated witness-type, span-type, mode, frontier-defect, projection-defect, saturation-defect, anchor-count, charge-size, profile-size, canonical-code priority order, Lean provides the exact lexicographic proposition, an equivalent executable comparison, all ten priority witnesses, proof-bearing descent, accessibility, induction, and kernel-checked well-foundedness.	This establishes the fixed residual rank domain and RankWF only. It does not map the current finite terminal routes into the manuscript's complete global outcome system, prove that any existing route strictly decreases the rank, establish route completeness or Package E, remove the explicit positive premise from the finite composition, establish full manuscript-wide SaturatePositive or BCELReady, prove ZeroSlack, PCCMin, polynomial runtime, SAT in P, remove a project assumption, or prove P = NP.
Candidate-derived finite BN3 request envelope	formalized-residual-terminal-bn3-request-envelope	From every successful computed finite BCEL anchor nucleus, Lean uses one canonical duplicate-free primitive-record identity list across every proper cut; gives exact executable monotone request membership stable under extensional transport and exact singleton minimal consumers; accounts active incidences without duplicates; selects one canonical full or quotient side-tight coherent basis for every proper cut; and preserves all upstream proof-bearing classifier failures in a total outcome.	This establishes one exact candidate-derived finite BN3 envelope only after the existing computed BCEL anchor-nucleus classifier succeeds. Proper cuts are enumerated through all subsets, so the construction is exponential reference computation rather than a polynomial algorithm. It does not derive the terminal dependency system, map local routes into the manuscript's complete global outcome system, construct BN4-BN6, prove selector or realizer completeness, establish global ZeroSlack or PCCMin, prove SAT in P, remove a project assumption, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Finite BN4 activation-exact cancellation kernel	formalized-residual-terminal-bn4-activation-cancellation	After a successful computed finite BN3 envelope, Lean gives every request atom a canonical singleton activation code; proves activation-code equality exactly equivalent to equality of activation functions without enumerating cuts; checks equality of a complete typed key containing the atom, explicit semantic signature, and explicit transport type; totals positive and negative natural mass only at that same complete key; classifies a canonical balanced, positive, or negative residual; proves exact integer mass conservation, complete-key preservation, positive residual mass, and absence of opposite-sign residual pairs; computes duplicate-free ledger keys; and preserves all upstream failure branches while rejecting foreign request atoms.	This is a finite cancellation kernel over an explicit typed cell ledger. It does not derive cells, semantic signatures, or transport types from four-corner bases and is not the full historical BN4 theorem. It supplies no polynomial construction or size bound; does not construct PkgC or BN6; does not complete global routes, selectors, or realizers; does not establish ZeroSlack or PCCMin; does not put SAT in P; does not remove a project assumption; and does not prove P = NP.
Finite BN5 full-shadow localization kernel	formalized-residual-terminal-bn5-full-shadow-localization	For every explicit finite negative-unit refinement and quotient-shadow ledger, Lean preserves the complete exact-coordinate data, validates the negative mass refinement, computes whether the cut is silent, and otherwise returns either complete multiplicity coverage or a strict Hall deficit with a literal smaller shadow-neighbor fibre, complete-coordinate preservation, and a proof-bearing local X1 route that prevents active unmatched units from disappearing silently.	This kernel starts from explicit finite inputs: one complete BN4 key, a negative cancellation result, a payload list, a cut, and a quotient-shadow coordinate list. It does not derive payloads or shadows from four-corner bases, connect complete matching back to a BN4 contradiction, or prove the full CritC/Q/E/L/X2/X3/X4 diagnosis, so it is not the full historical BN5 theorem. It does not construct PkgC or BN6; complete global routes, selectors, or realizers; establish polynomial generation or runtime, ZeroSlack, or PCCMin; put SAT in P; remove a project assumption; or prove P = NP.

Milestone	Status	Exact scope	Boundary
Finite PkgC separating-consumer restoration dichotomy	formalized-residual-terminal-pkgc-separating-consumers	For an arbitrary finite explicit minimal-consumer antichain, Lean canonically scans for the first disjoint pair that is not singleton-singleton. Absence proves exactly V54's singletonization premise. A found pair's atoms are canonically indexed into exact-coordinate quotient units and an explicit full-restoration universe is classified into complete multiplicity coverage or a strict Hall deficit with a deterministic local Q route.	The restoration coordinate universe remains explicit. This theorem does not derive consumers or restorations from a terminal candidate, connect complete coverage back to a BN4 or BN5 contradiction, embed the Hall route into the complete global outcome system, prove global route silence, or establish the full historical PkgC theorem. It does not prove full BN6 or Packet selector-realizer completeness, polynomial generation or runtime, ZeroSlack or PCCMin, SAT in P, remove a project assumption, or prove P = NP.
Finite PkgC typed restoration realization	formalized-residual-terminal-pkgc-typed-restoration	For an arbitrary finite explicit minimal-consumer antichain and a typed coordinate-preserving restoration operation, Lean materializes typed full-restoration candidates for every atom of the canonical first disjoint nonsingleton pair, proves exact candidate count and positional coordinate preservation, derives complete equality-fibre multiplicity coverage, excludes a strict Hall deficit for that graph, and otherwise proves V54 singletonization.	The typed restoration operation remains explicit caller data. This milestone does not construct it from a terminal candidate or prove its full semantic adequacy. It does not connect complete restoration to a BN4 or BN5 contradiction, embed local routes into the complete global outcome system, prove global PkgC route silence or the full historical PkgC theorem, establish full BN6 or Packet selector-realizer completeness, polynomial generation or runtime, ZeroSlack or PCCMin, put SAT in P, remove a project assumption, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Finite PkgC typed restoration same-key cancellation	formalized-residual-terminal-pkgc-same-key-cancellation	For an arbitrary finite explicit minimal-consumer antichain and typed exact-BN5-coordinate restoration operation, Lean mechanically pairs every quotient atom with its restored full candidate as opposite-sign unit cells, proves the complete BN5 coordinate gives the same nested BN4 key, proves exact cell count and positive/negative multiplicity equality at every BN4 key, computes an empty canonical residual and zero signed mass at every key, and derives V54 singletonization from exact absence of every such proof-bearing cancellation outcome.	The typed restoration operation and its complete coordinate maps remain explicit inputs, and the generated opposite-sign cells are not yet proved to be the cells of the terminal candidate's ambient BN4 ledger. This milestone does not construct semantic restorations from a terminal candidate, embed cancellation or Hall outcomes into the complete global route system, prove global route silence or the full historical PkgC theorem, establish full BN6 or Packet selector-realizer completeness, polynomial generation or runtime, ZeroSlack or PCCMin, put SAT in P, remove a project assumption, or prove P = NP.
Finite PkgC ambient BN4 ledger embedding	formalized-residual-terminal-pkgc-ambient-bn4-ledger	For arbitrary finite explicit BN4 cell ledgers, Lean proves that a proof-bearing exact multiset embedding identifies the generated PkgC opposite-sign cancellation ledger with an ambient subledger and preserves every duplicate. Positive and negative mass decompose at every complete key; removing the balanced generated subledger leaves the ambient signed mass and executable residual signed contribution exactly equal to an explicit remainder. A successful candidate-derived BN4 kernel additionally proves every embedded generated cell uses its canonical request-atom space, and complete bindings plus exact absence of every computed bridge imply V54 singletonization.	The ambient ledger, typed restoration operation, exact permutation certificate or canonical serialization, and successful candidate-derived BN4 kernel remain explicit proof-bearing inputs. This milestone does not derive the ambient ledger or restorer from a terminal candidate, prove the restorer's semantic adequacy, embed local cancellation or Hall outcomes into the complete global route system, prove global PkgC route silence or the full historical PkgC theorem, establish full BN6 or Packet selector-realizer completeness, polynomial generation or runtime, ZeroSlack or PCCMin, put SAT in P, remove a project assumption, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Finite V54 consumer-antichain normal form	formalized-residual-terminal-v54-consumer-antichain-normal-form	For an arbitrary finite carrier and its explicit minimal-consumer antichain, Lean proves monotonicity and empty-request inactivity, proves that nonzero two-sided cut activation is equivalent to the existence of a disjoint consumer pair, and under the exact singletonized-disjoint-pair premise proves literal equality with the corresponding footprint cut indicator.	This finite kernel starts from an explicit minimal-consumer antichain and an explicit proof that every disjoint consumer pair is singletonized. It does not construct PkgC, derive that singletonization premise, or connect the footprint back to the full BN6 proof. It does not construct complete global routes, selectors, or realizers; establish polynomial generation or runtime, ZeroSlack, or PCCMin; put SAT in P; remove a project assumption; or prove P = NP.
Finite V53 constant-cut hypergraph rigidity	formalized-residual-terminal-v53-constant-cut-hypergraph-rigidity	For an arbitrary finite duplicate-free carrier and sparse nonnegative weighted hypergraph with positive listed cells, exact equality of every nonempty proper cut proves the complete V53 q=2, q=3, and q>=4 classification: full-span weight D; one common pair weight p with w_A + 2p = D; or zero weight on every proper footprint with full-span weight D.	This theorem consumes an explicit sparse positive hypergraph and an explicit proof that every nonempty proper cut has the same positive value. It does not construct PkgC, derive the hypergraph from a terminal candidate or the V54 consumer system, build BN6 cells or payloads, complete global routes, selectors, or realizers, establish polynomial generation or runtime, prove ZeroSlack or PCCMin, put SAT in P, remove a project assumption, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Finite BN6 grouped hypergraph packet bridge	formalized-residual-terminal-bn6-hypergraph-packet	For an arbitrary finite duplicate-free anchor carrier and explicit already-grouped family of positive payload-bearing survivor cells, V54 activation is transported exactly into the constructed V53 hypergraph cut sum. A positive BCEL constant-cut premise then yields the complete pair, mixed three-anchor balanced-triple/full-span, or four-or-more-anchor full-span classification with original payload witnesses.	This finite bridge consumes explicit exact footprint grouping, PkgC singletonization proofs, positive atom ledgers, payload data, and the BCEL constant-cut equation. It does not construct PkgC, derive or group survivors from a terminal candidate, establish full historical BN6 or Packet selector/realizer completeness, complete global routes, prove polynomial generation or runtime, ZeroSlack or PCCMin, put SAT in P, remove a project assumption, or prove P = NP.
Global locked-NAND construction and threshold	formalized-concrete-locked-nand-threshold	A uniform encoded polynomial-time SAT instance builder and the report-level locked-NAND threshold theorem linked to that builder.	This closes the uniform all-bitstring CNFSAT-to-concrete-locked-threshold builder and report-facing linkage in the finite charged-pipeline model. It does not put the concrete locked threshold language in P, discharge residual-band minimization, ZeroSlack or PCCMin, prove concrete CNFSAT NP-hardness, activate the legacy string-handle bridge, or prove P = NP.

Milestone	Status	Exact scope	Boundary
Global ZeroSlack, PCCMin, and polynomial runtime	not-formalized	Complete residual routing, global ZeroSlack contradiction, exact minimization, and polynomial bounds.	The finite candidate-derived BN3 envelope supplies stable request identities and one jointly side-tight canonical basis family; the finite BN4 kernel supplies activation-exact same-key integer cancellation over an explicit typed cell ledger; and the finite BN5 kernel localizes explicit full/shadow multiplicity failure to a strict Hall deficit and local X1 route. The construction still does not derive the BN4 ledger or BN5 payload/shadow universe from the bases, connect matching back to a contradiction, establish the full historical BN4 or BN5 theorems, construct PkgC or BN6, map all residual routes into a decreasing complete global outcome system, or provide selector/realizer and polynomial-runtime completeness; global ZeroSlack and polynomial PCCMin therefore remain unformalized.
Concrete standard P-versus-NP target and root theorem	not-formalized	Raw-machine-linked complexity classes, concrete SAT completeness and a SAT decider, plus the publication root theorem.	The concrete target definition remains inactive: CNF-SAT now has a raw-machine verifier, NP-membership proof, and an all-input finite-machine polynomial reduction through NAND to the concrete locked target, but a deterministic polynomial-time CNF-SAT decider, the concrete-to-abstract threshold link, SAT NP-hardness and CNFSAT NP-completeness transport, and PNP.Main.p_eq_np remain absent; the legacy string-handle bridge is ineligible.

4 Compiled inventory summary

The inventory contains 27794 exported `PNP.*` kernel declarations from 250 modules. It excludes 15008 private compiler auxiliaries. Counts describe the compiled environment; they do not measure mathematical completeness.

Module	Declarations	Theorems
PNP.Bridge	93	22
PNP.Complexity	93	20
PNP.Concrete.BitString	123	60
PNP.Concrete.CNF	247	60
PNP.Concrete.CNFSourceParserCompiled	15	11
PNP.Concrete.CNFSourceParserCorrectness	198	133
PNP.Concrete.CNFSourceParserMachine	106	44
PNP.Concrete.CNFSourceParserSpec	18	15
PNP.Concrete.CNFToNAND	202	115
PNP.Concrete.CNFToNANDCarrierEncoder	563	256
PNP.Concrete.CNFToNANDCarrierTokenReader	70	27
PNP.Concrete.CNFToNANDCompilerCompiled	15	10
PNP.Concrete.CNFToNANDCompilerMachine	30	18
PNP.Concrete.CNFToNANDCompilerPolynomialBound	19	15
PNP.Concrete.CNFToNANDCompilerTotalTrace	7	7
PNP.Concrete.CNFToNANDCompilerTrace	16	15
PNP.Concrete.CNFToNANDController	473	103
PNP.Concrete.CNFToNANDControllerBlocks	79	29
PNP.Concrete.CNFToNANDControllerCanonicalTrace	146	105
PNP.Concrete.CNFToNANDControllerCompletionTrace	45	33
PNP.Concrete.CNFToNANDControllerCountTrace	238	202
PNP.Concrete.CNFToNANDControllerPolynomialBound	45	36
PNP.Concrete.CNFToNANDControllerTotalBound	2	1
PNP.Concrete.CNFToNANDControllerTotalTrace	4	4
PNP.Concrete.CNFToNANDEmitterPlan	224	125
PNP.Concrete.CNFToNANDPolynomialReduction	15	11
PNP.Concrete.CNFToNANDWorkspace	96	77
PNP.Concrete.CNFVerifier	10	7
PNP.Concrete.CNFWorkCorrectness	855	521
PNP.Concrete.CNFWorkFrameCorrectness	16	4
PNP.Concrete.CNFWorkInput	29	14
PNP.Concrete.CNFWorkMachine	85	3
PNP.Concrete.CNFWorkTransitions	64	63
PNP.Concrete.CNFWorkUniversalCorrectness	294	145
PNP.Concrete.Complexity	312	92
PNP.Concrete.CookLevinBuilderBodyStartPrefix	85	66
PNP.Concrete.CookLevinBuilderCompleteHeader	134	85
PNP.Concrete.CookLevinBuilderDynamicTokenCursorStep	68	54
PNP.Concrete.CookLevinBuilderFifthClausePaddingRun	143	109
PNP.Concrete.CookLevinBuilderFirstClausePaddingRun	160	114
PNP.Concrete.CookLevinBuilderFirstClausePrefix	120	85
PNP.Concrete.CookLevinBuilderFirstConstraintPaddingRun	158	124
PNP.Concrete.CookLevinBuilderFirstLiteralPrefix	121	99
PNP.Concrete.CookLevinBuilderFirstTokenPrefix	48	36
PNP.Concrete.CookLevinBuilderFourthClauseFirstLiteral Prefix	138	102
PNP.Concrete.CookLevinBuilderFourthClausePaddingRun	148	114
PNP.Concrete.CookLevinBuilderFourthClausePrefix	97	80
PNP.Concrete.CookLevinBuilderFourthClauseSecondLiteral Prefix	187	137
PNP.Concrete.CookLevinBuilderFourthClauseSeparatorStep	85	70
PNP.Concrete.CookLevinBuilderInputLength	47	25
PNP.Concrete.CookLevinBuilderInputPrefix	52	39
PNP.Concrete.CookLevinBuilderSecondClauseFirstLiteral Prefix	135	106
PNP.Concrete.CookLevinBuilderSecondClausePaddingRun	126	92
PNP.Concrete.CookLevinBuilderSecondClausePrefix	86	69
PNP.Concrete.CookLevinBuilderSecondClauseSecondLiteral Prefix	158	118
PNP.Concrete.CookLevinBuilderSecondClauseSeparatorStep	86	69

Module	Declarations	Theorems
PNP.Concrete.CookLevinBuilderSecondConstraintFifthPaddingOrTerminatorOpportunityStep	114	82
PNP.Concrete.CookLevinBuilderSecondConstraintFirstLiteralFirstUnaryUnitStep	94	79
PNP.Concrete.CookLevinBuilderSecondConstraintFirstLiteralSecondUnaryUnitStep	94	79
PNP.Concrete.CookLevinBuilderSecondConstraintFirstLiteralSignStep	99	84
PNP.Concrete.CookLevinBuilderSecondConstraintFirstLiteralSuccessorTokenStep	111	79
PNP.Concrete.CookLevinBuilderSecondConstraintFirstLiteralTerminatorStep	310	295
PNP.Concrete.CookLevinBuilderSecondConstraintFirstLiteralThirdUnaryUnitStep	94	79
PNP.Concrete.CookLevinBuilderSecondConstraintFourthPaddingOrUnaryOpportunityStep	96	67
PNP.Concrete.CookLevinBuilderSecondConstraintPaddingOrUnaryOpportunityStep	114	82
PNP.Concrete.CookLevinBuilderSecondConstraintSecondPaddingOrUnaryOpportunityStep	96	67
PNP.Concrete.CookLevinBuilderSecondConstraintSeparatorStep	93	78
PNP.Concrete.CookLevinBuilderSecondConstraintSeventhPaddingOrUnaryOpportunityStep	96	67
PNP.Concrete.CookLevinBuilderSecondConstraintSixthPaddingOrOpeningUnaryOpportunityStep	96	67
PNP.Concrete.CookLevinBuilderSecondConstraintThirdPaddingOrUnaryOpportunityStep	96	67
PNP.Concrete.CookLevinBuilderThirdClauseFirstLiteralPrefix	131	105
PNP.Concrete.CookLevinBuilderThirdClausePaddingRun	134	100
PNP.Concrete.CookLevinBuilderThirdClausePrefix	93	76
PNP.Concrete.CookLevinBuilderThirdClauseSecondLiteralPrefix	213	158
PNP.Concrete.CookLevinBuilderThirdClauseSeparatorStep	81	66
PNP.Concrete.CookLevinBuilderTokenAppender	99	64
PNP.Concrete.CookLevinBuilderUnaryPolynomial	207	118
PNP.Concrete.CookLevinFormulaCursor	259	121
PNP.Concrete.CookLevinFormulaSchedule	109	78
PNP.Concrete.CookLevinFormulaSize	168	149
PNP.Concrete.CookLevinLayout	204	71
PNP.Concrete.CookLevinLocalCNF	172	53
PNP.Concrete.CookLevinRawTapeBridge	107	95
PNP.Concrete.CookLevinTableau	71	23
PNP.Concrete.CookLevinTableauCNF	179	60
PNP.Concrete.CookLevinTableauCNFSemantics	190	136
PNP.Concrete.CookLevinVerifierTableau	58	25
PNP.Concrete.LockedNANDEncoding	611	215
PNP.Concrete.LockedNANDPolynomialReduction	8	6
PNP.Concrete.LockedNANDRawBuilder	292	169
PNP.Concrete.LockedNANDReduction	21	13
PNP.Concrete.LockedNANDSourceParserCompiled	15	11
PNP.Concrete.LockedNANDSourceParserCorrectness	38	35
PNP.Concrete.LockedNANDSourceParserFailureShapes	124	44
PNP.Concrete.LockedNANDSourceParserMachine	216	41
PNP.Concrete.LockedNANDSourceParserSemantics	36	36
PNP.Concrete.LockedNANDSourceParserSpec	17	14
PNP.Concrete.LockedNANDSourceParserTotalTrace	54	28
PNP.Concrete.LockedNANDSourceParserValidTrace	356	235
PNP.Concrete.LockedNANDTargetEmitterBlockCompiler	66	38
PNP.Concrete.LockedNANDTargetEmitterCapacity	160	130
PNP.Concrete.LockedNANDTargetEmitterCheckStack	341	149
PNP.Concrete.LockedNANDTargetEmitterController	220	31
PNP.Concrete.LockedNANDTargetEmitterControllerCheckTrace	21	18
PNP.Concrete.LockedNANDTargetEmitterControllerCompiled	19	11
PNP.Concrete.LockedNANDTargetEmitterControllerCompletionTrace	33	17

Module	Declarations	Theorems
PNP.Concrete.LockedNANDTargetEmitterControllerGate Bound	65	41
PNP.Concrete.LockedNANDTargetEmitterControllerGateList Trace	34	18
PNP.Concrete.LockedNANDTargetEmitterControllerGate Trace	82	47
PNP.Concrete.LockedNANDTargetEmitterControllerHeader Bound	29	21
PNP.Concrete.LockedNANDTargetEmitterControllerHeader Trace	13	10
PNP.Concrete.LockedNANDTargetEmitterControllerInitial Trace	14	13
PNP.Concrete.LockedNANDTargetEmitterController NormalizationBound	45	33
PNP.Concrete.LockedNANDTargetEmitterController NormalizationTrace	27	19
PNP.Concrete.LockedNANDTargetEmitterControllerOutput Bound	47	28
PNP.Concrete.LockedNANDTargetEmitterControllerOutput Trace	54	41
PNP.Concrete.LockedNANDTargetEmitterController PolynomialBound	39	28
PNP.Concrete.LockedNANDTargetEmitterControllerPrefix Bound	75	43
PNP.Concrete.LockedNANDTargetEmitterControllerPrefix Trace	49	34
PNP.Concrete.LockedNANDTargetEmitterControllerSource Trace	41	32
PNP.Concrete.LockedNANDTargetEmitterControllerTotal Trace	15	15
PNP.Concrete.LockedNANDTargetEmitterControllerTrace	76	66
PNP.Concrete.LockedNANDTargetEmitterCursorAppender	118	71
PNP.Concrete.LockedNANDTargetEmitterCursorControl	39	22
PNP.Concrete.LockedNANDTargetEmitterCursorFinalizer	32	15
PNP.Concrete.LockedNANDTargetEmitterCursorNatLoop	202	95
PNP.Concrete.LockedNANDTargetEmitterFinalizer	29	14
PNP.Concrete.LockedNANDTargetEmitterGrammarScanner	294	117
PNP.Concrete.LockedNANDTargetEmitterLedger	854	359
PNP.Concrete.LockedNANDTargetEmitterMachine	178	67
PNP.Concrete.LockedNANDTargetEmitterMarkedSource Reload	229	110
PNP.Concrete.LockedNANDTargetEmitterNatLoop	187	82
PNP.Concrete.LockedNANDTargetEmitterNavigator	127	39
PNP.Concrete.LockedNANDTargetEmitterPlan	538	186
PNP.Concrete.LockedNANDTargetEmitterPrimitiveCompiler	56	44
PNP.Concrete.LockedNANDTargetEmitterProgramSemantics	367	222
PNP.Concrete.LockedNANDTargetEmitterRuntime	15	10
PNP.Concrete.LockedNANDTargetEmitterRuntimeCheckStack	18	14
PNP.Concrete.LockedNANDTargetEmitterRuntimeLayout	38	23
PNP.Concrete.LockedNANDTargetEmitterRuntimePrimitives	94	84
PNP.Concrete.LockedNANDTargetEmitterRuntimeProgram	132	45
PNP.Concrete.LockedNANDTargetEmitterRuntimeProgram Bound	64	58
PNP.Concrete.LockedNANDTargetEmitterRuntimeProgram Safety	141	93
PNP.Concrete.LockedNANDTargetEmitterRuntimeSource Control	20	15
PNP.Concrete.LockedNANDTargetEmitterSchedule	28	10
PNP.Concrete.LockedNANDTargetEmitterScratchAddSlot	177	79
PNP.Concrete.LockedNANDTargetEmitterScratchCompare Slot	110	31
PNP.Concrete.LockedNANDTargetEmitterScratchCompare SlotExact	38	34
PNP.Concrete.LockedNANDTargetEmitterScratchIncrement	66	21
PNP.Concrete.LockedNANDTargetEmitterScratchReset	77	28
PNP.Concrete.LockedNANDTargetEmitterSemantic Completion	16	7

Module	Declarations	Theorems
PNP.Concrete.LockedNANDTargetEmitterSemanticFinal	7	6
PNP.Concrete.LockedNANDTargetEmitterSemantic Normalization	57	41
PNP.Concrete.LockedNANDTargetEmitterSemanticOutput	15	12
PNP.Concrete.LockedNANDTargetEmitterSemanticPrefix	65	32
PNP.Concrete.LockedNANDTargetEmitterSemanticPrefix Bridge	2	1
PNP.Concrete.LockedNANDTargetEmitterSemanticSchedule	72	46
PNP.Concrete.LockedNANDTargetEmitterSlotIncrement	202	88
PNP.Concrete.LockedNANDTargetEmitterSourceCapture	210	73
PNP.Concrete.LockedNANDTargetEmitterSpec	57	51
PNP.Concrete.LockedNANDThresholdPublication	1	1
PNP.Concrete.Machine	284	67
PNP.Concrete.PipelineCompiler	38	27
PNP.Concrete.PipelineInputFramer	106	18
PNP.Concrete.PipelineMachineSimulation	175	79
PNP.Concrete.PipelineOutputHandoff	23	4
PNP.Concrete.PipelinePairedCompiler	29	25
PNP.Concrete.PipelineRefinement	68	24
PNP.Concrete.PipelineSequentialCompiler	36	28
PNP.Concrete.PipelineSequentialStateNamespace	30	21
PNP.Concrete.PipelineStageBridges	77	59
PNP.Concrete.PipelineStateNamespace	77	34
PNP.Concrete.PipelineTapeGeometry	20	12
PNP.Concrete.PipelineTerminalBridge	73	62
PNP.Concrete.TapeBlankEquivalence	25	17
PNP.Concrete.TapeHandoff	18	11
PNP.Concrete.Target	2	1
PNP.Concrete.TerminalOutputPacker	111	36
PNP.Concrete.WorkInput	23	14
PNP.Concrete.WorkMachine	308	108
PNP.Concrete.WorkMachineBlankEquivalence	51	33
PNP.Concrete.WorkMachineChain	28	20
PNP.Concrete.WorkMachineProgramGraph	225	118
PNP.Concrete.WorkMachineProgramPath	23	8
PNP.Concrete.WorkMachineProgramPathBlankEquivalence	1	1
PNP.DirectWireBaseline	48	25
PNP.LockedNAND	11	4
PNP.LockedNANDBaseline	57	22
PNP.LockedNANDCarrierTrace	324	198
PNP.LockedNANDDirect	84	57
PNP.LockedNANDGlobalCandidates	180	107
PNP.LockedNANDGlobalSemanticThreshold	8	7
PNP.LockedNANDGlobalUnsatisfiableFinalZero	2	2
PNP.LockedNANDLocalBaseline	30	22
PNP.LockedNANDMacros	288	98
PNP.LockedNANDPrefix	84	40
PNP.LockedNANDResidualGainBound	4	3
PNP.LockedNANDThresholdBoundary	60	35
PNP.Main	36	12
PNP.NANDComposition	77	38
PNP.NANDEnumerator	120	46
PNP.NANDMinimum	56	23
PNP.NANDSemantics	198	80
PNP.NANDSlack	15	12
PNP.NANDTruthTable	72	28
PNP.PCCMin	44	8
PNP.ResidualBand	9	2
PNP.ResidualGainChain	22	9
PNP.ResidualGainStopping	14	12
PNP.ResidualRoutes	141	46
PNP.ResidualTerminalBCELAnchorNucleus	408	138
PNP.ResidualTerminalBN2SquareLegitimacy	40	22
PNP.ResidualTerminalBN3RequestEnvelope	84	44
PNP.ResidualTerminalBN4ActivationCancellation	227	97
PNP.ResidualTerminalBN5FullShadowLocalization	278	100
PNP.ResidualTerminalBN6HypergraphPacket	94	37
PNP.ResidualTerminalCandidateSaturation	52	8
PNP.ResidualTerminalConstantCutHypergraphRigidity	155	109

Module	Declarations	Theorems
PNP.ResidualTerminalConsumerAntichainNormalForm	64	38
PNP.ResidualTerminalExecutableSaturation	90	31
PNP.ResidualTerminalFiniteSaturatePositive	68	23
PNP.ResidualTerminalFourCornerCarrier	93	43
PNP.ResidualTerminalFourCornerOptimumCoherence	179	56
PNP.ResidualTerminalFourCornerOptimumCompatibility	83	38
PNP.ResidualTerminalFourCornerSideTightCompletion	65	24
PNP.ResidualTerminalFourCornerTightBasisMaximum	57	23
PNP.ResidualTerminalFrontierPushout	52	25
PNP.ResidualTerminalFullBridge	56	23
PNP.ResidualTerminalGovernedSupportCompletion	68	30
PNP.ResidualTerminalInterfaceExposureRouting	224	75
PNP.ResidualTerminalModeFirewall	148	41
PNP.ResidualTerminalOriginKernelObligationRouting	330	97
PNP.ResidualTerminalPhysicalSupportCompletion	93	34
PNP.ResidualTerminalPkgCAmbientBN4Ledger	60	22
PNP.ResidualTerminalPkgCSameKeyCancellation	75	37
PNP.ResidualTerminalPkgCSeparatingConsumers	131	51
PNP.ResidualTerminalPkgCTypedRestoration	86	35
PNP.ResidualTerminalProjectionMinimum	36	23
PNP.ResidualTerminalProjectionSquare	21	17
PNP.ResidualTerminalProjectionTransfer	29	8
PNP.ResidualTerminalProperSupport	56	27
PNP.ResidualTerminalRankWF	54	23
PNP.ResidualTerminalSaturation	170	60
PNP.ResidualTerminalSaturationCostBalance	162	48
PNP.ResidualTerminalSaturationPositivityFirewall	56	23
PNP.ResidualTerminalSideTightMinimum	98	35
PNP.ResidualTerminalSupportExtraction	63	32
PNP.ResidualTerminalSupportSquareClosure	77	30
PNP.SAT	4	1
PNP.ZeroSlack	141	21

5 Remaining formal blockers

Five formal blockers remain active:

1. `Formal.ConcreteSAT`
2. `Formal.ResidualBandMinimizer`
3. `Formal.ZeroSlack`
4. `Formal.PolynomialRuntimeAndCertificateBounds`
5. `Formal.RootTheoremAndAxiomAudit`

The conditional locked-NAND threshold module assumes typed baseline and full candidates, baseline conditions, preservation of existing outputs, an unsatisfiable final-zero law, and satisfiable final-output laws. The explicit residual scanner searches only a caller-supplied finite list. The gain-chain theorem bounds every supplied verified sequence, including locked-family sequences by four, but does not generate or complete that sequence. None of these results proves global ZeroSlack, polynomial PCCMin, SAT in P, or P equals NP.

6 Historical report provenance

Earlier 56-page report revisions stated a direct checker-mediated P-versus-NP claim. Those bytes remain historical audit material at their pinned legacy coordinates and through the explicit archive and supersession records. They are not imported into, quoted as authority by, or used to generate this report. File identity and replay of historical predicates do not establish the named mathematical propositions.

Source tag	`final-pnp-proof-report-hardened-7072f8d`
Source commit	`7072f8d0bda6d44d240f9bb3fad624fd357e1278`
Archive manifest	`archive/legacy-v0/ARCHIVE.json`

7 Verification

The current reconstruction can be checked with:

```
lake build PNP
node scripts/export-lean-theorem-inventory.mjs --check
node scripts/generate-formal-publication.mjs --check
node pcc-formal-reconstruction-status0.mjs --json
npm run pnp:verify -- --no-write
npm run report:check
```

The inventory coordinate is `PNP-LEAN-THEOREM-INVENTORY-2026-08-12-133`. Its exact byte digest is `696c76220a092e5a84e7caa804fd1c57889f193968d1285b520c408f8237f5c1`. The report is regenerated from that inventory and the reviewed publication map. The reviewed milestone source-closure SHA-256 is `9b8afc2bac8c5f5b5fbe3c086f22602358c3f9b641aeb91e7de708f9f1001154`; the computed and pinned values match. Successful regeneration confirms consistency of these status surfaces; it does not turn an absent concrete theorem into a proof.