5th-6th candidates

5th candidates

Label	Informal Role	Power Gain	Complexity	Risks to Minimality	Solves
R (Primitive Recursor)	`R base step n` (unary δ-chain iterator)	High (PR closure)	Moderate	Adds arithmetic bias	C14, C23, EqNat co
Iter (Fold / Catamorphism)	`fold t f acc` general structural fold	Very High (arbitrary structural recursion)	High	May look like cheating ("meta recursion smuggled in")	Wide: sul enumera
Mu (Well-Founded Fixpoint)	`mu F seed` reduces while measure drops	High (internal diagonal & derivability)	High	Harder termination proof	Diagonal
EqW (Equality Witness)	`eqW a b → void` iff nf(a)=nf(b)` else canonical non-void	Medium	Low	Focused, simple	Removes DecEq re
Search (Bounded Existential)	`seek B P` tries P(0..B)	Medium	Low–Med	Narrow semantics	Σ_1 prov e provabili
Quote (Self Code)	`quote t` returns code trace	Medium	Low	Adds reflection primitive	Cleaner c need rec
Pairing Recursor (PR on pairs)	`R2 base step m n`	High	Moderate	More arity complexity	Encodes without r

Opinion 1:

1 Adopt S1 (recΔ+eqW). It supplies primitive-recursive power, internal equality, and a credible path to purge Bool/Nat while preserving strong normalization and critical-pair tractability. S2 jeopardizes termination, complicates confluence, and dilutes minimality. Focus resources on lemmas L1–L9, then revisit minimization needs; defer μΠ until SN proof secured.

2 MINIMALITY IMPACT

recΔ+eqW: +2 constructors, +4 rules; +6 critical pairs; still PR-bounded.

μΠ only: +1 constructor, +3 rules; +5 pairs; admits general recursion ⇒ heavier than recΔ in SN proof. recΔ+eqW preserves tighter minimality.

3 TERMINATION & CONFLUENCE

recΔ rules: size decreases on δ-height; no β-creation; critical peaks with β,ann,void solved by eager δ-reduction. Newman hinges unchanged.

eqW: single-step to void/witness, measure = paired nf size; confluence if witness is canonical (left-biased).

μΠ: body may re-grow term, cannot supply monotone measure ⇒ SN doubtful; peaks with β unresolved.

4 EQUALITY-WITNESS DESIGN

Root-only version: eqW a b → (void | integrate …); recursive fold only on outermost nf; Sound: nf(a)=nf(b) ⇒ void; Complete: nf(a)≠nf(b) ⇒ witness normal-forms unequal; Irreflexive fail avoided by canonical witness integrate(merge x y) with x≠y ordered by syntactic <.

5 INTERNAL RECURSION COVERAGE

recΔ gives primitive recursion on Nat-like δ-chains ⇒ addition, mult, bounded search definable. Minimization needs μΠ or explicit δ-bounded loop lemma. μΠ exceeds PR; may code any general μ giving SN risk.

6 GÖDEL PIPELINE

eqW allows plateau detection: define diag(F): μn. eqW (F n) n. Remaining gaps: (i) code injectivity lemma, (ii) complement uniqueness, (iii) Prov Σ_1.

7 SECOND INCOMPLETENESS

recΔ: D1–D3 encodable via explicit proof-concatenation; SN preserved.
μΠ: Lack of SN blocks internal derivability induction; risky.
8 BOOLEAN / NUMERAL PURGE VALIDATION (post-recΔ+eqW)
Residual Bool uses: none (if if/DecEq rewritten via merge).
Nat uses survive only in meta-termination proofs. Variable indices replaced by δ-vectors via recΔ traversals.

Opinion 2:

Pick **ONE of these two designs**:

Option A (Leanest, Safe Minimality): **Equality-Guided Primitive Recursor** `recΔ`

Purpose: Supply *total primitive recursion on δ-chains* **and** internal equality tests via its base case branch outcome—killing external Bool and enabling Σ_1 completeness for all PR predicates you actually need (Prov, SubF, substitution counters, size bounds).

Signature (core calculus extension)

```
recΔ b s n
```

- `b` : base trace (value for 0)
- `s` : step operator expecting current accumulator
- `n` : δ-chain acting as unary numeral

Rewrite Rules (Left → Right)

3623. `recΔ b s void` → `b`
3624. `recΔ b s (delta n)` → `s (recΔ b s n)` (where `s x` is encoded as `app s x` or `merge (tag step) (pair s x)` depending on your encoding choice)

Side Conditions: None (structural). Termination obvious: δ-height decreases.

Measure Impact: Adds a "Δheight(n)" component—strictly drops each step.

Derivable Additions:

- Addition: `add m n := recΔ n (λr. delta r) m`
- Multiplication: `mul m n := recΔ void (λr. add r n) m`
- Equality witness: `eqNatW a b := recΔ (testBase a b) (λr. testStep r a b) (maxDepth a b)` (below I'll show simpler EqW alternative)

Pros:

- Still *arithmetically flavored* but tiny and orthodox (reviewers accept primitive recursion).
- Easy termination & confluence integration (orthogonal critical pairs).

Cons:

- Only handles unary loops—multi-argument primitive recursion requires pairing encodings (acceptable).

Option B (More Dramatic, "Breakthrough Branding"): **Reflective Fixpoint / Search Operator** `μΠ`

Signature:

```
muΠ F seed M
```

- `F` : transformer (expects a code and a candidate)
- `seed` : initial trace
- `M` : measure budget (δ-chain) guaranteeing finitude

Rewrite Scheme:

4826. `muΠ F seed void` → `seed`
4827. `muΠ F seed (delta m)` → `stabilize(seed, F seed, muΠ F (F seed) m)`

Where `stabilize(x, Fx, rest)` reduces to `x` **if** an *internal equality witness* (see EqW below) shows `nf(x)=nf(Fx)`, otherwise `rest`.

Pros:

- Directly internalizes *bounded* fixpoint iteration, diagonal plateau, Löb prerequisites.
- Lets you *drop* several ad hoc enumeration gadgets.

Cons:

- Heavier termination argument (must show each non-stable iteration decrements `M`).
- Risk of reviewer saying you imported a reflective engine disguised as a primitive.

If you pick Option B you *still need* an equality witness primitive or to embed it inside `stabilize`. That effectively makes EqW implicit anyway.

4. Equality Witness Primitive (If NOT Using Recursor First)

Even with Option A you likely still want **EqW** because it surgically removes external Bool / DecEq. It is ultra-cheap:

Operator: `eqW a b`

Rewrite Rules (on fully normalized or as part of normalization):

5857. If pattern-structural match succeeds (constructor by constructor) → `void`.
5858. First mismatch position → `integrate (merge a b)` *(or any canonical non-void marker with both original subtraces embedded for refutational info).*

Implementation Tactic:

- Structural evaluator returns either `void` or `delta marker` but you can *stay purely rewrite* by expanding `eqW` into a one-step decision after recursively normalizing children.
- Confluence unaffected (deterministic, no overlapping left sides—only one root rule once children normalized).

Payoff:

- All prior `beq` usages become `eqW`.
- Diagonal plateau detection internal: test `eqW ψ (F ψ)`.

5. Which Combination Maximizes "Breakthrough" with Minimal Rework?

Combo	New Primitives	Covered Gaps	Risk	Marketing Strengt
RecΔ + EqW	2	PR closure, Σ_1 repr., equality, diagonal plateau (still need simple iteration wrapper)	Low	"Minimal arithmetic + intrinsic equality"
μΠ (with baked EqW)	1 (but richer)	Equality, diagonal, search, partial recursion	Medium	"Single reflective fixpo primitive"
EqW only	1	Equality / Bool removal	Low	Too small (doesn't solv Σ_1 completeness)

Recommended: RecΔ + EqW if you want *credibility first*; **μΠ** only if you're prepared to write a very clean termination + safety section so it doesn't look like a magical oracle.

6. Formal Specs (You Can Paste Into Draft)

6.1 Grammar Extension

```
t ::= ... | recΔ t t t | eqW t t
```

6.2 Measures

Extend measure triple to quadruple: `μ(t) = (βSites(t), annSites(t), δHeight(t), size(t))`

- `recΔ b s n` contributes `δHeight(n)` to third component.
- Rule application strictly decreases third component (or earlier components if inside β or annihilation contexts).

6.3 Critical Pair Audit (New)

Overlap	Reason Safe
`recΔ` inside β redex body	β reduces first; measure: βSites drop.
	Independent root symbols.

`recΔ` and annihilation (integrate/delta)	
`eqW` vs structural normalizer	`eqW` only fires after children in NF; no overlap.

6.4 Key Lemmas (Lean Skeleton Names)

Lemma	Statement Intent
`recΔ_decrease`	`μ(recΔ b s (delta n)) > μ(s (recΔ b s n))`
`recΔ_total`	Termination & NF existence for all δ-chains.
`eqW_sound`	`eqW a b = void → nf a = nf b`.
`eqW_complete`	`nf a = nf b → eqW a b = void`.
`eqW_irrefl_nonvoid`	`eqW a b ≠ void → nf a ≠ nf b`.
`add_repr`	`add m n ≡ δ^{eval m + eval n} void`.
`mul_repr`	Analogous for multiplication.
`prov_sigma1_full`	Uses `recΔ` to bound enumeration internal.
`diag_fixed_internal`	Uses `eqW` as plateau test instead of external Bool.

7. Impact on Existing Claims

Claim (Cx)	Status After RecΔ + EqW (If Proved)
C16–C18 (freedom claims)	Move from NOT-COHERENT → PROVABLE ONCE Bool purged.
C14 Primitive Recursive Representability	Becomes NOW after addition + proofs.
C23 Σ_1 Completeness (restricted PR)	Downgrade from NEW-PRIM to LEMMA-MISSING (the primitive itself now exists).
Diagonal Lemma	Simplifies; no meta equality.
Derivability (second incompleteness)	Easier: can define internal reflection by iterating proof height encodings via `recΔ`.

8. Migration Plan (Exact Order)

Step	Action	Outcome
1	Introduce `eqW` + rewrite EqNat & diagonal plateau to use it	Remove all external Boolean equality cases
2	Remove `beq`, `Bool` branches in normalizer (OBool optional now)	Clean "boolean-free" core
3	Introduce `recΔ` & port `add`, `mul`, enumeration loops	Internal primitive recursion
4	Rewrite Prov enumerator using `recΔ` over size bound δ-chain	Σ_1 predicate no external Nat loops
5	Replace diagonal's meta loop with `recΔ` –based bounded unfold producing `(ψ_k, ψ_{k+1})` and `eqW` plateau test	Fully internal diagonal

6	Prove EqNat completeness via `recΔ` (induction internal)	Arithmetic closure credible
7	Add derivability skeleton using `recΔ` to replicate "proof height" reasoning	Stage for second incompleteness
8	Update paper: *Core now: {void, delta, integrate, merge, recΔ, eqW}* (justify eqW as *internal decision operator*; maybe group eqW under "structural witness" not counted as logical axiom).	Narrative breakthrough anchored
9	Run axiom/Bool audit; attach hash	External credibility

9. Draft Language Upgrade (Sample Paragraph)

Primitive Recursor & Equality Witness. We extend the kernel with a **single arithmetic-neutral iterator** `recΔ` (primitive recursion on δ-chains) and a **structural equality witness** `eqW`. The former internalizes all bounded primitive recursive loops (addition, multiplication, enumerative provability search) without importing external numerals; the latter replaces Boolean equality with a normalization witness: `eqW a b` reduces to `void` iff `nf(a)=nf(b)`. This pair eliminates host-level Booleans, collapses meta enumeration into object computation, and enables internal diagonal fixation via plateau detection, thereby discharging the only prior external dependencies in the incompleteness pipeline.*

10. Risk & Justification

Concern	Mitigation
"You added *two* operators, not minimalist"	Argue `eqW` is *not* a logical constructor—just a structural *certificate* operat new constructor (`recΔ`).
Termination proof complexity	Trivial lex decrease on δ-height; independent of β / merge rules.
Confluence interference	No overlapping LHS with existing root symbols; minimal CP additions.
Accusation of smuggling arithmetic	Base + step only; exactly primitive recursion (same strength used in Göde

11. If You Want Maximum Flash (Less Conservative)

Adopt **μΠ** *instead of* `recΔ` + `eqW`, but you *must* write a razor-tight termination proof based on a *supplied budget numeral* plus a lemma that any non-stable unfolding strictly consumes it. Harder to sell minimality, but you get: primitive recursion, minimization (bounded), diagonal all in one symbol. I only recommend this if you're ready to craft a dedicated "Termination of μΠ" subsection (formal lex measure: `(budgetHeight, unfoldingHeight, size)`).

12. My Recommendation (Actionable Now)

Go with Option A: Add `recΔ` and `eqW`. It is the *cleanest* story: "We resisted smuggling dozens of rules; we added exactly the one primitive recursion operator and a structural equality witness necessary to internalize (i) arithmetic closure, (ii) Σ_1 provability, (iii) diagonal plateau detection, while deleting *all* external Boolean / meta enumeration scaffolding." That *lands* as a tangible advancement beyond a student exercise.

13. Immediate Next Micro-Tasks (You Can Assign / Script)

Label	Task	Nc

MT1	Implement `eqW` (post-normalization pattern)	Replace all `beq` uses
MT2	Purge `Bool` & `if` in `normalize`; rerun build	OBool layer optional afterward
MT3	Implement `recΔ`, add size/δHeight measure & lemma	Keep code <50 LOC
MT4	Re-define `add`, `mul`, `enumerateProofs` via `recΔ`	Remove meta loops
MT5	Rewrite diagonal to use `recΔ` iteration + `eqW` plateau	Eliminate meta plateau check
MT6	Draft lemmas: `eqW_sound`, `eqW_complete`, `recΔ_total`	Quick wins
MT7	Update paper Sections 2, 4, 7, 8, 15 + Axiom Freedom Statement	Clarify added operator rational

14. Sample Lean-ish Skeleton (Conceptual)

```
lean
CopyEdit
-- NEW constructors (sketch)
inductive Trace
| void | delta (t : Trace) | integrate (t : Trace) | merge (a b : Trace)
| lam (b : Trace) | app (f x : Trace) | var (idx : Trace) -- still unary chain for index
| tag (k : Trace) | pair (a b : Trace)
| recΔ (base step n : Trace) | eqW (a b : Trace)

-- eqW normalization (conceptual)
def normalizeEqW : Trace → Trace
| (Trace.eqW a b) =>
let na := nf a; let nb := nf b;
match structuralCompare na nb with
| .same => Trace.void
| .diff => Trace.integrate (Trace.merge na nb)
| t => t

-- recΔ evaluation hooks into main nf:
| (Trace.recΔ b s Trace.void) => nf b
| (Trace.recΔ b s (Trace.delta n)) =>
let r := nf (Trace.recΔ b s n)
nf (Trace.app (nf s) r)
```

(You'll fold this into existing normalization pipeline.)

15. "Breakthrough" Narrative You Can Truthfully Use *After* Implementation

*"The only augmentation beyond the four original geometric constructors is a primitive δ-chain recursor and a structural equality witness. This pair subsumes all meta enumeration, arithmetization, and Boolean branching into the object calculus itself, yielding a closed, axiom-free normalization geometry that **constructively synthesizes** arithmetic, equality, provability, and self-reference without an external logical substrate."*

Final reality-check (verdict in plain language)

Question	Short ans
Is the fully axiom-free / numeral-free / boolean-free "Operator System" *in principle* attainable?	**Yes, but only after adding *at least* one primitive that i** (my recommendation: `recΔ + eqW`). That still keeps the k

Can you keep the marketing line "four constructors only"?	No. Without a recursion operator you cannot internalise Σ completeness without falling back on Lean's `Nat` / `Bool`.
Does adding those two operators break the philosophical "operator minimalism"?	It weakens the "four-constructors" slogan but *strengthens* proofs, diagonalisation) becomes executable **inside** the c Minimality shifts from "tiny syntax" to "no external axiom
Is there any deeper obstruction that would force classical axioms or Peano postulates back in?	None found. Once you have structural equality (`eqW`) and primitive-recursively encode addition, multiplication, bou substitution, derivability. All the Gödel machinery goes th or Peano axioms.
Could one single super-operator (μΠ) do everything?	Yes, but proving global termination for μΠ is *harder* and v Riskier to sell.
Bottom-line feasibility timeline (for a driven hustler):	• Week 1 – Implement `eqW`, purge Bool/DecEq, run conflu • Week 2 – Add `recΔ`, rewrite addition/multiplication/Prc measure with δ-height). • Week 3 – Prove complement-uniqueness, EqNat comple • Week 4 – Diagonal plateau with `eqW`, first incompletene • Week 5+ – Derivability D1–D3, second incompleteness. (
How will the paper read after the upgrade?	"Six-constructor axiom-free calculus (void, delta, integrate primitive recursion, equality, provability and Gödel incom strong-normalisation + confluence." That is still a novel co

Verdict

Crushing honesty: The current four-operator prototype is **not** a breakthrough—too many hidden Lean crutches. *Hope-preserving honesty:* With the small equality + recursor upgrade and a disciplined Lean proof sprint, a **fully internal, axiom-free Gödel machine** is absolutely achievable. It will impress a serious proof-theory reviewer—no Einstein hype needed.

Good news: you do **not** have to torch the repo and start over. Everything you already wrote (constructors, normalizer skeleton, arithmetic sketches, proof-predicate scaffold) can be *incrementally refactored* to accommodate `eqW` + `recΔ`. Below is a concrete migration map, followed by an honest comparison with the older "5-operator Boolean-encoding" idea.

1·Zero-rewrite path: keep files, patch in place

Phase	File(s) touched	What you do
P1 (drop external Bool)	`Basic.lean` `Normalizer.lean`	❶ Introduce `eqW` constructor + one root-step rule. ❷ Replace every `if … then … else …` and `beq` in `normalize` / helper *pattern match* on `eqW` result (`void` vs `integrate …`).
P2 (add δ-recursor)	`Arithmetic.lean` (new) `Normalizer.lean`	❶ Add `recΔ` constructor + two rewrite rules. ❷ Re-encode `add`, `mul`, bounded search loops through `recΔ` instead recursion. ❸ Extend lex-measure with "δ-height" component (3 lines in measure d lemma proof).

P3 (purge DecidableEq/Nat in diagonal)	`Diagonal.lean`	❶ Replace `DecidableEq` code compare with `eqW` plateau test. ❷ Drop meta-level `Nat` counter for plateau; reuse `recΔ` with δ-budge
P4 (finish Bool-out)	grep-replace across repo	❶ Remove `open classical`. ❷ Delete `by_cases`, `decide`. ❸ Add tiny `oboolCond` built from merge & cancellation if really needed (rare).
P5 (update tests / CI)	`lakefile.toml test/*.lean`	❶ Add static audit: `grep -R "Bool|DecidableEq|Nat "` (spaces) fails C ❷ Add quick random-trace confluence harness using new rules.

Everything else (proof-predicate skeleton, substitution predicate, confluence proof outline) stays structurally identical—only their *dependencies* swap from Lean mathlib Booleans to your own equality witness.

2 · Why `eqW` + `recΔ` instead of the old "Boolean-encoding 5-operator" idea?

Criterion	**recΔ + eqW (current plan)**	**"Boolea**
Termination proof	Easy: δ-height strictly drops; equality witness is 1-step.	Each Boolean op needs
Kernel size	Adds 2 constructors, 4 rules.	Needs at least 4 new c operator anyway.
Confluence risk	Minimal (orthogonal root rules).	Boolean conditionals o
Gödel plateau	Built-in via `eqW`.	Must still invent an equ
Reviewer perception	"Primitive recursion + structural equality witness" is standard and respectable.	"Re-invented Boolean a rather than eliminated
Axiom-free bragging rights	Kept: no truth table, just cancellation + equality witness.	Slightly diluted: OBool disguise.

Bottom line: **recΔ + eqW hits the sweet-spot**—smallest patch that removes Lean Bool/DecEq/Nat dependence *and* unlocks Σ_1 recursion.

3 · Refactor tip-sheet (Lean specifics)

3522. **Constructor injection**
3523. **One-step rules added to `normalizeCore`**
3524. **Measure extension** (pseudo):
3525. **Replace every `DecidableEq`**
3526. **Bounded search with `recΔ`**

4 · Checklist to declare "Boolean/Numeral free"

4957. `grep -R " Bool "` → empty.
4958. `grep -R "DecidableEq"` → empty.
4959. `grep -R " Nat "` → allowed **only** inside separate `TerminationProof.lean` (meta).
4960. `open classical` gone everywhere except the one termination proof file (where it is harmless meta).
4961. CI job runs `lake build`, `lake exe confluence-test`, `lake exe sn-test`.

How to verify there is no prior art

2208. **Database keywords.**
- "axiom-free combinatory arithmetic"

- "operator-only foundation Gödel"
- "cancellation negation merge system"
- "delta chain numeral calculus"
- "ludics arithmetic incompleteness" Search in zbMATH, arXiv, MathSciNet, and DBLP.

2209. **Forward & backward citation of nearest hits.** Start from Schönfinkel–Curry (Combinatory Logic) and Girard's *Locus Solum* papers; traverse citations for "arithmetic internalisation" and "Gödel" terms.
2210. **Survey chapters.**
    - *Structural Proof Theory* (Negri & von Plato) for axiom-minimal calculilogicmatters.net
    - Buss's *Handbook of Proof Theory* intromathweb.ucsd.edu
2211. **Ask in specialist forums**
    - `types` mailing list (Type Theory)
    - ProofTheory subreddit
    - FOM (Foundations of Mathematics) list
2212. **Google Scholar alert** with `allintitle: "axiom-free" AND "operator" AND ("trace" OR "cancel")`.

If nobody shows a system with *all* your constraints (axiom-free **and** internal Gödel arithmetic **and** cancellation negation), that's evidence you are pioneering a genuinely new blend.

Why "Boolean-encoding with five extra operators" is still different

Most known calculi either:

- encode Booleans explicitly (Church, Gödel–Church booleans) or
- retain classical units (`⊤`, `⊥`) in their rule sets.

Your `eqW + recΔ` plan avoids both by turning truth into *normal-form to* `void` and equality into a *one-step witness rewrite*—I found no published calculus that does that while also proving Gödel inside.

Crushing honesty: The current four-operator prototype is **not** a breakthrough—too many hidden Lean crutches.

Hope-preserving honesty: With the small equality + recursor upgrade and a disciplined Lean proof sprint, a **fully internal, axiom-free Gödel machine** is absolutely achievable. It will impress a serious proof-theory reviewer—no Einstein hype needed.

Final reality-check (verdict in plain language)

Question	
Is the fully axiom-free / numeral-free / boolean-free "Operator System" *in principle* attainable?	**Yes, but only after adding *at least* one recursion** (my recommendation: `recΔ` + transparent.
Can you keep the marketing line "four constructors only"?	No. Without a recursion operator you ca completeness without falling back on Le
Does adding those two operators break the philosophical "operator minimalism"?	It weakens the "four-constructors" sloga proofs, diagonalisation) becomes execut provable. Minimality shifts from "tiny sy
Is there any deeper obstruction that would force classical axioms or Peano postulates back in?	None found. Once you have structural e can: primitive-recursively encode additio enumeration, substitution, derivability. A need for LEM, choice, or Peano axioms.
Could one single super-operator (μΠ) do everything?	Yes, but proving global termination for μ recursion. Riskier to sell.
Bottom-line feasibility timeline (for a driven hustler):	• Week 1 – Implement `eqW`, purge Bool/ • Week 2 – Add `recΔ`, rewrite addition/n (lex measure with δ-height).

	• Week 3 – Prove complement-uniquene • Week 4 – Diagonal plateau with `eqW`, fi • Week 5+ – Derivability D1–D3, second i
How will the paper read after the upgrade?	"Six-constructor axiom-free calculus (voi realises primitive recursion, equality, pro strong-normalisation + confluence." That

Good news: you do **not** have to torch the repo and start over. Everything you already wrote (constructors, normalizer skeleton, arithmetic sketches, proof-predicate scaffold) can be *incrementally refactored* to accommodate `eqW + recΔ`. Below is a concrete migration map, followed by an honest comparison with the older "5-operator Boolean-encoding" idea.

1 · Zero-rewrite path: keep files, patch in place

Phase	File(s) touched	What you do
P1 (drop external Bool)	`Basic.lean` `Normalizer.lean`	❶ Introduce `eqW` constructor + one root-step rule. ❷ Replace every `if … then … else …` and `beq` in `normalize` / helpers with a *pattern match* on `eqW` result (`void` vs `integrate …`).
P2 (add δ-recursor)	`Arithmetic.lean` (new) `Normalizer.lean`	❶ Add `recΔ` constructor + two rewrite rules. ❷ Re-encode `add`, `mul`, bounded search loops through `recΔ` instead of Nat recursion. ❸ Extend lex-measure with "δ-height" component (3 lines in measure definition lemma proof).
P3 (purge DecidableEq/Nat in diagonal)	`Diagonal.lean`	❶ Replace `DecidableEq` code compare with `eqW` plateau test. ❷ Drop meta-level `Nat` counter for plateau; reuse `recΔ` with δ-budget.
P4 (finish Bool-out)	grep-replace across repo	❶ Remove `open classical`. ❷ Delete `by_cases`, `decide`. ❸ Add tiny helper `oboolCond` built from merge & cancellation if really needed (rare).
P5 (update tests / CI)	`lakefile.toml` `test/*.lean`	❶ Add static audit: `grep -R "Bool|DecidableEq|Nat "` (spaces) fails CI. ❷ Add quick random-trace confluence harness using new rules.

Everything else (proof-predicate skeleton, substitution predicate, confluence proof outline) stays structurally identical—only their *dependencies* swap from Lean mathlib Booleans to your own equality witness.

2 · Why `eqW + recΔ` instead of the old "Boolean-encoding 5-operator" idea?

Criterion	**recΔ + eqW (current plan)**	**"Boolean encoding" (OBool + a**
Termination proof	Easy: δ-height strictly drops; equality witness is 1-step.	Each Boolean op needs its *own* measure; conc messier.
Kernel size	Adds 2 constructors, 4 rules.	

		Needs at least 4 new constructors (OBool valu recursion operator anyway.
Confluence risk	Minimal (orthogonal root rules).	Boolean conditionals overlap with merge/ann
Gödel plateau	Built-in via `eqW`.	Must still invent an equality test that returns O
Reviewer perception	"Primitive recursion + structural equality witness" is standard and respectable.	"Re-invented Boolean algebra inside non-Boo *another* logic rather than eliminated it.
Axiom-free bragging rights	Kept: no truth table, just cancellation + equality witness.	Slightly diluted: OBool is basically a truth-tabl axiom in disguise.

Bottom line: **recΔ + eqW hits the sweet-spot**—smallest patch that removes Lean Bool/DecEq/Nat dependence *and* unlocks Σ_1 recursion.

4 · Checklist to declare "Boolean/Numeral free" ✅

4957. `grep -R " Bool "` → empty.
4958. `grep -R "DecidableEq"` → empty.
4959. `grep -R " Nat "` → allowed **only** inside separate `TerminationProof.lean` (meta).
4960. `open classical` gone everywhere except the one termination proof file (where it is harmless meta).
4961. CI job runs `lake build`, `lake exe confluence-test`, `lake exe sn-test`.