AGENT.md — All-in-One AI Guide for OperatorKernelO6 / OperatorMath

(FULL, NOTHING DROPPED — includes all missing ordinal API + helpers)

> **Audience:** LLMs/agents working on this repo.\ **Prime Directive:** Don't touch the kernel. Don't hallucinate lemmas/imports. Don't add axioms.\ **If unsure:** raise a **CONSTRAINT BLOCKER**.

---

0. TL;DR

1. **Kernel is sacred.** 6 constructors, 8 rules. No edits unless explicitly approved.
2. **Inside kernel:** no `Nat`, `Bool`, numerals, `simp`, `rfl`, `classical`, pattern-matches on non-kernel stuff. Only `Prop` + recursors.
3. **Meta land:** You may use Nat/Bool, classical, tactics, WF recursion, and *only* the imports/lemmas listed in §8.
4. **Main jobs:** SN, normalize-join confluence, arithmetic via `recΔ`, internal equality via `eqW`, provability & Gödel.
5. **Allowed outputs:** `PLAN`, `CODE`, `SEARCH`, **CONSTRAINT BLOCKER** (formats in §6).
6. **NEVER drop or "simplify" §8.** That's your API.

---

1. Project

Repo: OperatorKernelO6 / OperatorMath\ **What it is:** A *procedural*, **axiom-free, numeral-free, boolean-free** foundation where *everything* (logic, arithmetic, provability, Gödel) is built from one inductive `Trace` type + a deterministic normalizer. No Peano axioms, no truth tables, no imported equality axioms.

Core claims to protect:

- **Axiom freedom** (no external logical/arithmetic schemes).
- **Procedural truth:** propositions hold iff their trace normalizes to `void`.
- **Emergence:** numerals = δ-chains; negation = merge-cancellation; proofs/Prov/diag all internal.
- **Deterministic geometry:** strong normalization (μ-measure) + confluence → canonical normal forms.

Deliverables:

1. Lean artifact: kernel + meta proofs (SN, CR, arithmetic, Prov, Gödel) — sorry/axiom free.
2. Paper alignment: matches "Operator Proceduralism" draft; section numbers map 1:1.
3. Agent safety file (this doc): exhaustive API + rules for LLMs.

---

2. Prime Directive

- Do **not** rename/delete kernel code.
- Edit only what is required to fix an error.
- Keep history/audit trail.

---

3. Kernel Spec (Immutable)

```
namespace OperatorKernelO6

inductive Trace : Type
| void : Trace
| delta : Trace → Trace
| integrate : Trace → Trace
| merge : Trace → Trace → Trace
| recΔ : Trace → Trace → Trace → Trace
| eqW : Trace → Trace → Trace

open Trace

inductive Step : Trace → Trace → Prop
| R_int_delta : ∀ t, Step (integrate (delta t)) void
| R_merge_void_left : ∀ t, Step (merge void t) t
| R_merge_void_right : ∀ t, Step (merge t void) t
| R_merge_cancel : ∀ t, Step (merge t t) t
| R_rec_zero : ∀ b s, Step (recΔ b s void) b
| R_rec_succ : ∀ b s n, Step (recΔ b s (delta n)) (merge s (recΔ b s n))
| R_eq_refl : ∀ a, Step (eqW a a) void
| R_eq_diff : ∀ a b, Step (eqW a b) (integrate (merge a b))

inductive StepStar : Trace → Trace → Prop
| refl : ∀ t, StepStar t t
| tail : ∀ {a b c}, Step a b → StepStar b c → StepStar a c

def NormalForm (t : Trace) : Prop := ¬ ∃ u, Step t u

/-- Meta helpers; no axioms. --/
 theorem stepstar_trans {a b c : Trace} (h1 : StepStar a b) (h2 : StepStar b
c) : StepStar a c := by
  induction h1 with
  | refl => exact h2
  | tail hab _ ih => exact StepStar.tail hab (ih h2)

theorem stepstar_of_step {a b : Trace} (h : Step a b) : StepStar a b :=
```

```
  StepStar.tail h (StepStar.refl b)

theorem nf_no_stepstar_forward {a b : Trace} (hnf : NormalForm a) (h : StepStar a b) : a = b :=
  match h with
  | StepStar.refl _ => rfl
  | StepStar.tail hs _ => False.elim (hnf ⟨_, hs⟩)

end OperatorKernelO6
```

NO extra constructors or rules. No side-condition hacks. No Nat/Bool/etc. in kernel.

---

4. Meta-Level Freedom

Allowed (outside `OperatorKernelO6`): Nat, Bool, classical choice, tactics (`simp`, `linarith`, `ring`, …), WF recursion, ordinal measures, etc., **but only using §8's imports/lemmas**.\ Forbidden project-wide unless green-lit: `axiom`, `sorry`, `admit`, `unsafe`, stray `noncomputable`.\ Never push these conveniences back into the kernel.

---

5. Required Modules & Targets

1. **Strong Normalization (SN):** measure ↓ on every rule → `WellFounded`.
2. **Confluence:** use **normalize-join** (define `normalize`, prove `to_norm`, `norm_nf`, `nfp`, then `confluent_via_normalize`). Don't rewrite rules.
3. **Arithmetic & Equality:** numerals as δ-chains; `add`/`mul` via `recΔ`; compare via `eqW`.
4. **Provability & Gödel:** encode proofs as traces; diagonalize without external number theory.
5. **Fuzz Tests:** random deep rewrites to stress SN/CR.

---

6. Interaction Protocol

6.1 Output Forms

- **PLAN** – numbered outline, no code.
- **CODE** – single Lean block, no chatter.
- **SEARCH** – `name | file | line | purpose`.
- **CONSTRAINT BLOCKER** – exact missing theorem + reason.

6.2 Style

- Use `theorem`, not `lemma`.
- No comments inside Lean files (header ok).
- Kernel: avoid pattern-matching on non-kernel stuff.

- No axioms/unsafe.

6.3 When in Doubt

Raise **CONSTRAINT BLOCKER**; do **not** guess imports or lemmas.

---

7. Common Pitfalls

- `DecidableEq Trace` → don't derive in kernel. Decide via normal forms in meta.
- `termination_by` arity mismatch (Lean ≥ 4.6: no fn name in clause).
- Lex orders unification hell → unfold relations manually.
- Ordinal lemma missing? Check §8; if absent, restate locally or blocker.
- WF recursion slow? Use helper libs (outside kernel).

---

8. Canonical Imports, Lemmas & Snippets (NO DROPS)

8.1 Import Whitelist

```
import OperatorKernelO6.Kernel              -- kernel
import Init.WF                               -- WellFounded, Acc, InvImage.wf,
Subrelation.wf
import Mathlib.Data.Prod.Lex                 -- Prod.Lex for lex orders
import Mathlib.Tactic.Linarith               -- linarith, Nat.cast_lt
import Mathlib.Tactic.Ring                   -- ring, ring_nf
import Mathlib.Algebra.Order.SuccPred        -- Order.lt_add_one_iff,
add_one_le_iff
import Mathlib.SetTheory.Ordinal.Basic       -- ω, lt_wf, omega0_pos,
one_lt_omega0, nat_lt_omega0, lt_omega0
import Mathlib.SetTheory.Ordinal.Arithmetic  -- pow_succ, pow_pos,
add_lt_add_left, mul_lt_mul_of_pos_left,
                                             -- mul_le_mul_left',
mul_le_mul_right', le_mul_right
import Mathlib.SetTheory.Ordinal.Exponential -- opow, opow_pos, opow_add,
opow_le_opow_right,
                                             -- right_le_opow, left_le_opow,
IsNormal.strictMono
import Mathlib.Data.Nat.Cast.Order.Basic     -- Nat.cast_le, Nat.cast_lt
```

(Trim only if symbols vanish.)

8.2 Ordinal Toolkit (with Signatures)

Lemma / Theorem	Signature (exact)	Purpose	Module
`omega0_pos`	`0 < omega0`	ω positive	Ordinal.Basic
`one_lt_omega0`	`1 < omega0`	ω > 1	Ordinal.Basic
`lt_omega0`	`o < omega0 ↔ ∃ n : ℕ, o = n`	Finite ordinal characterization	Ordinal.Basic
`nat_lt_omega0`	`∀ n : ℕ, (n : Ordinal) < omega0`	ℕ embeds below ω	Ordinal.Basic
`Nat.cast_le` / `Nat.cast_lt`	`((m : Ordinal) ≤ (n : Ordinal)) ↔ m ≤ n` / analog for `<`	cast bridges	Nat.Cast.Order.Basic
`pow_succ`	`a ^ (k+1) = a ^ k * a`	exponent step	Ordinal.Arithmetic
`pow_pos`	`0 < a → 0 < a ^ b`	positivity of powers	Ordinal.Arithmetic
`add_lt_add_left`	`a < b → c + a < c + b`	add-left mono	Ordinal.Arithmetic
`mul_lt_mul_of_pos_left`	`{a b c} → a < b → 0 < c → c * a < c * b`	mul-left strict mono	Ordinal.Arithmetic
`mul_le_mul_left'`	`{a b c} → a ≤ b → c * a ≤ c * b`	mul-left mono (≤)	Ordinal.Arithmetic
`mul_le_mul_right'`	`{a b c} → a ≤ b → a * c ≤ b * c`	mul-right mono (≤)	Ordinal.Arithmetic
`le_mul_right`	`∀ a b, 0 < b → a ≤ b * a`	absorb into product	Ordinal.Arithmetic

Lemma / Theorem	**Signature (exact)**	Purpose	Module
`opow_pos`	`0 < a → 0 < a ^ b`	positivity for `opow`	Ordinal.Exponential
`opow_add`	`a ^ (b + c) = a ^ b * a ^ c`	exponent add law	Ordinal.Exponential
`opow_le_opow_right`	`0 < a → b ≤ c → a ^ b ≤ a ^ c`	monotone (≤) in exponent	Ordinal.Exponential
`right_le_opow`	`1 < a → b ≤ a ^ b`	base dominates exponent	Ordinal.Exponential
`left_le_opow`	`0 < b → a ≤ a ^ b`	exponent dominates base	Ordinal.Exponential
`IsNormal.strictMono`	`<strictMono f>` for normal `f`	mono normal functions	Ordinal.Exponential
`Order.lt_add_one_iff`	`x < y + 1` ↔️ `x ≤ y` (and dual)	successor arithmetic	Algebra.Order.SuccPred
`Order.add_one_le_of_lt`	`x < y → x + 1 ≤ y`	successor ≤ intro	Algebra.Order.SuccPred
`Ordinal.one_add_of_omega0_le`	`omega0 ≤ p → (1 : Ordinal) + p = p`	collapse 1+ on infinite	Ordinal.Basic
`Ordinal.natCast_add_of_omega0_le`	`omega0 ≤ p → (n : Ordinal) + p = p`	collapse n+ on infinite	Ordinal.Basic
`right_le_opow` / `left_le_opow`	see above	exponent/base dominance	Ordinal.Exponential
`right_le_opow` already listed but keep to avoid hallucinations			

8.2.1 Local Bridges (proved in meta files — copy/paste ok)

```
@[simp] theorem natCast_le {m n : ℕ} : ((m : Ordinal) ≤ (n : Ordinal)) ↔ m ≤
n := Nat.cast_le
@[simp] theorem natCast_lt {m n : ℕ} : ((m : Ordinal) < (n : Ordinal)) ↔ m <
n := Nat.cast_lt

theorem eq_nat_or_omega0_le (p : Ordinal) : (∃ n : ℕ, p = n) ∨ omega0 ≤ p := by
  classical
  cases lt_or_ge p omega0 with
  | inl h  => rcases (lt_omega0).1 h with ⟨n, rfl⟩; exact Or.inl ⟨n, rfl⟩
  | inr h  => exact Or.inr h

theorem one_left_add_absorb {p : Ordinal} (h : omega0 ≤ p) : (1 : Ordinal) + p
= p :=
  by simpa using (Ordinal.one_add_of_omega0_le h)

theorem nat_left_add_absorb {n : ℕ} {p : Ordinal} (h : omega0 ≤ p) : (n :
Ordinal) + p = p :=
  by simpa using (Ordinal.natCast_add_of_omega0_le (n := n) h)
```

8.2.2 Two-Sided `(*)` Monotonicity Helper (since no `Ordinal.mul_le_mul` exists)

```
/-- Two-sided monotonicity of `(*)` for ordinals, built from one-sided lemmas.
-/
theorem ord_mul_le_mul {a b c d : Ordinal} (h₁ : a ≤ c) (h₂ : b ≤ d) :
    a * b ≤ c * d := by
  have h₁' : a * b ≤ c * b := by
    simpa using (mul_le_mul_right' h₁ b)   -- mono in the left factor
  have h₂' : c * b ≤ c * d := by
    simpa using (mul_le_mul_left'  h₂ c)   -- mono in the right factor
  exact le_trans h₁' h₂'
```

8.2.3 μ-Measure Cheat Sheet (termination proofs)

- **Define** `mu : Trace → Ordinal` with big Cantor towers (see `Termination.lean`).
- **Goal form:** `Step a b → mu b < mu a`.
- **Typical moves:**
- Reduce to inequalities like `ω^k * (x+1) ≤ ω^(x + k')`.
- Use `le_omega_pow`, `right_le_opow`, `left_le_opow`, `opow_add`, then monotonicity lemmas.
- For coefficient padding, collapse finite left adds with `nat_left_add_absorb` / `one_left_add_absorb` once `ω ≤ p`.
- Combine multipliers via `ord_mul_le_mul` or the one-sided `mul_le_mul_*'` lemmas.

- **Successor arithmetic:** rewrite `p + 1` using `Order.lt_add_one_iff` / `Order.add_one_le_of_lt`.
- **Finite vs infinite split:** `eq_nat_or_omega0_le p` to branch on `p < ω` vs `ω ≤ p`.

  Keep these bullets close when proving bounds like `add3_plus1_le_plus4`, `termA_le`, etc. `(*)` for ordinals, built from one-sided lemmas. -/ theorem ord_mul_le_mul {a b c d : Ordinal} (h_1 : a ≤ c) (h_2 : b ≤ d) : a * b ≤ c * d := by have h_1' : a * b ≤ c * b := by simpa using (mul_le_mul_right' h_1 b) -- mono in the left factor have h_2' : c * b ≤ c * d := by simpa using (mul_le_mul_left' h_2 c) -- mono in the right factor exact le_trans h_1' h_2'

```
Everything it uses (`mul_le_mul_left'`, `mul_le_mul_right'`, `le_trans`) is
already whitelisted.

---

### 8.3 Minimal Import Blocks (pick what you need)
See §8.1. Don't import random modules.

---

### 8.4 Ready-Made Snippets
**Nat measure decrease:**
```lean
@[simp] def size : Trace → Nat
| void => 1
| delta t => size t + 1
| integrate t => size t + 1
| merge a b => size a + size b + 1
| recΔ b s n => size b + size s + size n + 1
| eqW a b => size a + size b + 1

theorem step_size_decrease {t u : Trace} (h : Step t u) : size u < size t := by
  cases h <;> simp [size]; linarith
```

Ordinal WF via measure:

```
def StepRev : Trace → Trace → Prop := fun a b => Step b a

theorem strong_normalization_forward
  (dec : ∀ {a b}, Step a b → mu b < mu a) : WellFounded (StepRev Step) := by
  have wfμ : WellFounded (fun x y : Trace => mu x < mu y) := InvImage.wf (f :=
mu) Ordinal.lt_wf
  have sub : Subrelation (StepRev Step) (fun x y => mu x < mu y) := by intro x
y h; exact dec h
  exact Subrelation.wf sub wfμ
```
```

Normalize-join confluence:

```
noncomputable def normalize (t : Trace) : Trace := (exists_normal_form t).choose

theorem confluent_via_normalize : Confluent := by
  intro a b c hab hac
  let n := normalize a
  have han := to_norm a
  have hbn := nfp hab han (norm_nf a)
  have hcn := nfp hac han (norm_nf a)
  exact ⟨n, hbn, hcn⟩
```

(Provide `exists_normal_form`, `to_norm`, `norm_nf`, `nfp` in `Normalize.lean`.)

8.5 Deprecated / Missing / Do-Not-Use

- `Std.Data.WellFounded` → use `Init.WF`.
- Old `Mathlib.Data.Ordinal.*` paths → now under `Mathlib.SetTheory.Ordinal.*`.
- No `Ordinal.mul_le_mul`, `Ordinal.mul_le_mul_left`, etc. → use primes or helper above.
- `succ_eq_add_one` (ordinal) not available; use `Nat.cast_succ` / successor lemmas.
- `le_of_not_lt` deprecated; use `le_of_not_gt`.
- Do **not** invent alternate relations (`StepDet`) unless you also prove equivalence with `Step` (prefer normalize-join instead).

8.6 Sanity Checklist (Per File)

- Imports ⊆ whitelist.
- No `axiom/sorry/admit/unsafe`.
- Kernel untouched.
- Every ordinal lemma either from §8.2 or proven locally.
- Confluence via normalize, not raw LC.
- If you need something new: **CONSTRAINT BLOCKER**.

---

9. Workflow Checklist

1. Kernel matches §3 verbatim.
2. SN: measure + decrease + WF.
3. Normalize: existence + `normalize` + `nfp`.
4. Confluence via normalize.
5. Arithmetic & equality via traces.
6. Provability & Gödel.
7. Fuzz tests.

8. Write/publish.

---

10. Output Examples

PLAN

```
PLAN
1. Define ordinal μ
2. Prove μ decreases on rules
3. WF via InvImage.wf
4. Build normalize + nfp
5. Confluence via normalize
```

CODE

```
CODE
-- StrongNorm.lean
import OperatorKernelO6.Kernel
import Init.WF
import Mathlib.Tactic.Linarith

namespace OperatorKernelO6.Meta
open Trace Step

@[simp] def size : Trace → Nat
| void => 1
| delta t => size t + 1
| integrate t => size t + 1
| merge a b => size a + size b + 1
| recΔ b s n => size b + size s + size n + 1
| eqW a b => size a + size b + 1

theorem step_size_decrease {t u : Trace} (h : Step t u) : size u < size t := by
  cases h <;> simp [size]; linarith

end OperatorKernelO6.Meta
```

SEARCH

```
SEARCH
size : Trace → Nat | StrongNorm.lean | 12 | measure for SN
```

```
step_size_decrease : Step t u → size u < size t | StrongNorm.lean | 22 | local
decrease
```

CONSTRAINT BLOCKER

```
CONSTRAINT BLOCKER
Needed theorem: opow_le_opow_right (a := omega0) : 0 < omega0 → p ≤ q → omega0 ^
p ≤ omega0 ^ q
Reason: bound head coefficient in μ(integrate (merge a b)). Not in §8, cannot
proceed.
```

---

11. Glossary

Trace, Step, StepStar, NormalForm, SN, CR, recΔ, eqW — same as §3. Keep semantics intact.

---

12. Final Reminders

- Kernel: be boring and exact.
- Meta: be clever but provable.
- Never hallucinate imports/lemmas.
- Ask when something smells off.

This file supersedes all earlier agent docs. If you spot a missing item, raise a blocker with the exact line you need.