AGENT.md — FINAL (2025-07-29)

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

> **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`, 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, rename, or “simplify” rules or imports without approval.**

---

## 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)

```lean
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**. `ring` is on the project whitelist (`Mathlib.Tactic.Ring`); use it for integer equalities. `simp` and `linarith` are also allowed. Forbidden project-wide unless green-lit: `axiom`, `sorry`, `admit`, `unsafe`, stray `noncomputable`. Never push these conveniences back into the kernel.

Tactics whitelist (Meta): `simp`, `linarith`, `ring`.

---

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`).
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

Outputs: PLAN / CODE / SEARCH / CONSTRAINT BLOCKER.
Style: use `theorem`; no comments inside `.lean`; no axioms/unsafe.
If unsure: raise a blocker (don't guess imports/lemmas).

---

7. Common Pitfalls

- Do **not** assume `μ s ≤ μ (δ n)` in `recΔ b s n`. `s` and `n` are independent; the inequality is **false** in general (counterexample and explanation in `ordinal-toolkit.md`).
- Don't derive `DecidableEq Trace` in the kernel. Decide via normal forms in meta.
- `termination_by` (Lean ≥ 4.6) takes **no function name**.
- Lex orders: unfold relations manually.
- Ordinal lemma missing? Check §8 here; then see `ordinal-toolkit.md`. If still missing, raise a blocker.

---

8. Canonical Imports & Ordinal Basics (Slim but Exact)

8.1 Import whitelist (authoritative)

```lean
import OperatorKernelO6.Kernel              -- kernel
import Init.WF                               -- WellFounded, Acc, InvImage.wf, Subrelation.wf
import Mathlib.Data.Prod.Lex                 -- lex orders
import Mathlib.Tactic.Linarith               -- linarith
import Mathlib.Tactic.Ring                   -- ring
import Mathlib.Algebra.Order.SuccPred        -- Order.lt_add_one_iff, Order.add_one_le_of_lt
import Mathlib.SetTheory.Ordinal.Basic       -- omega0_pos, one_lt_omega0, nat_lt_omega0,
lt_omega0
import Mathlib.SetTheory.Ordinal.Arithmetic  -- Ordinal.add_*, Ordinal.mul_* (ordinal API)
import Mathlib.SetTheory.Ordinal.Exponential -- opow, opow_add, isNormal_opow,
```

```
Ordinal.opow_le_opow_right
import Mathlib.Data.Nat.Cast.Order.Basic     -- Nat.cast_le, Nat.cast_lt
-- NOTE: `mul_le_mul_left` is **generic** (not ordinal-specific) and lives in
-- `Mathlib.Algebra.Order.Monoid.Defs`. Do **not** use it for ordinals.
```

8.2 Name-prefix rules (must be explicit in code)

- **Exponent ≤-monotone:** `Ordinal.opow_le_opow_right` (never the bare name).
- **Exponent <-monotone at base ω:** use the local theorem `opow_lt_opow_right` from `ordinal-toolkit.md`.
- **Product monotonicity:** `Ordinal.mul_lt_mul_of_pos_left` (strict) and `Ordinal.mul_le_mul_iff_left` / the primed variants `mul_le_mul_left'`, `mul_le_mul_right'` (weak). Prefer the `Ordinal.*` forms for ordinal multiplication.
- **Successor bridge:** `Order.lt_add_one_iff` and `Order.add_one_le_of_lt` (keep the `Order.` prefix).

8.3 Quick ordinal facts kept inline

- `omega0_pos : 0 < omega0`, `one_lt_omega0 : 1 < omega0`.
- `nat_lt_omega0 : ∀ n : ℕ, (n : Ordinal) < omega0` and `lt_omega0 : o < omega0 ↔ ∃ n, o = n`.

8.4 Pointers

The **complete** lemma catalogue, local bridges (including `opow_lt_opow_right`), μ-measure cookbook, and the do-not-use list are in `ordinal-toolkit.md`. Keep this section slim to avoid duplication.

---

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
```

CONSTRAINT BLOCKER

```
CONSTRAINT BLOCKER
Needed theorem: Ordinal.opow_le_opow_right (a := omega0) to lift ≤ through ω-powers.
Reason: bound head coefficient in μ-decrease proof. Import from §8.1.
```

---

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.