ordinal-toolkit.md — OperatorKernel O6

MOSES RAHNAMA
Mina Analytics
GENERATED: 2025-08-05 05:20:24

Overview

ordinal-op-toolkit

DOCUMENT CONTENT

Version 2025-07-29 — authoritative, no placeholders; aligns with AGENT.md (same date)

---

0 SCOPE

This toolkit consolidates **all ordinal facts, imports, name-prefix rules, and μ-measure patterns** required by the OperatorKernelO6 meta pro (SN, confluence, arithmetic). It is the single source of truth for ordinal API usage and module locations. If a symbol is not listed here (o AGENT.md §8), carefully evaluate the guidelines for using **out of documents** lemmas and tactics.

---

1 IMPORT & LIBRARY AUDIT (AUTHORITATIVE)

> Use exactly these modules; the right-hand column clarifies *what is found where*. Generic ordered-monoid lemmas must **not** be used for ordi multiplication unless explicitly noted.

Area | Correct import | Contains / Notes ---------------------------- | -- | -------------------------- -- W F / A c `Init.WF` | `WellFounded`, `Acc`, `InvImage.wf`, `Subrelation.wf` Prod lex orders | `Mathlib.Data.Prod.Lex` | `Prod.Lex` lexicographic measures Ordinal basics | `Mathlib.SetTheory.Ordinal.Basic` | `omega0_pos`, `one_lt_omega0`, `lt_omega0` `nat_lt_omega0` Ordinal arithmetic | `Mathlib.SetTheory.Ordinal.Arithmetic` | `Ordinal.add_`, *`Ordinal.mul_`* `Ordinal.mul_lt_mul_of_pos_left`, `Ordinal.mul_le_mul_iff_left`, primed `mul_le_mul_left'` / `mul_le_mul_right` `le_mul_right` Ordinal exponentiation | `Mathlib.SetTheory.Ordinal.Exponential` | `opow`, `opow_add` `Ordinal.opow_le_opow_right`, `isNormal_opow` Successor helpers | `Mathlib.Algebra.Order.SuccPred` `Order.lt_add_one_iff`, `Order.add_one_le_of_lt` ℕ-casts (order bridges) | `Mathlib.Data.Nat.Cast.Order.Basic` `Nat.cast_le`, `Nat.cast_lt` Tactics | `Mathlib.Tactic.Linarith`, `Mathlib.Tactic.Ring` | `linarith`, `ring` (both whiteliste **Generic monoid inequality** | `Mathlib.Algebra.Order.Monoid.Defs` | **Generic** `mul_le_mul_left` — do **not** use it for ordinal products.

Qualification rule (must appear verbatim at call-sites):

- **Exponent (≤-mono):** call `Ordinal.opow_le_opow_right` (never the bare name).

- **Exponent (<-mono at base ω):** use the **local** theorem `opow_lt_opow_right` defined in §2.4 (since upstream remov `Ordinal.opow_lt_opow_right`).
- **Products:** prefer `Ordinal.mul_lt_mul_of_pos_left` and `Ordinal.mul_le_mul_iff_left` (`mul_le_mul_left'` / `mul_le_mul_right'`) — these are the **ordinal** APIs.
- **Successor bridge:** call `Order.lt_add_one_iff` / `Order.add_one_le_of_lt` with the `Order.` prefix.

---

2 TOOLKIT LEMMA CATALOGUE (NAMES, SIGNATURES, MODULES)

>All entries compile under Mathlib 4 (≥ v4.8) + this project's local bridges. Nothing here is hypothetical.

2.1 Basics & Positivity

- `omega0_pos : 0 < omega0` — *module:* `SetTheory.Ordinal.Basic`

- `one_lt_omega0 : 1 < omega0` — *module:* `SetTheory.Ordinal.Basic`
- `lt_omega0 : o < omega0 ↔ ∃ n : ℕ, o = n` — *module:* `SetTheory.Ordinal.Basic`
- `nat_lt_omega0 : ∀ n : ℕ, (n : Ordinal) < omega0` — *module:* `SetTheory.Ordinal.Basic`

2.2 Addition & Successor

- `add_lt_add_left : a < b → c + a < c + b` — *module:* `SetTheory.Ordinal.Arithmetic`

- `add_lt_add_right : a < b → a + c < b + c` — *module:* `SetTheory.Ordinal.Arithmetic`
- `add_le_add_left : a ≤ b → c + a ≤ c + b` — *module:* `SetTheory.Ordinal.Arithmetic`
- `add_le_add_right : a ≤ b → a + c ≤ b + c` — *module:* `SetTheory.Ordinal.Arithmetic`
- `Order.lt_add_one_iff : x < y + 1 ↔ x ≤ y` — *module:* `Algebra.Order.SuccPred`
- `Order.add_one_le_of_lt : x < y → x + 1 ≤ y` — *module:* `Algebra.Order.SuccPred`

Absorption on infinite right addends

- `Ordinal.one_add_of_omega_le : omega0 ≤ p → (1 : Ordinal) + p = p`

- `Ordinal.nat_add_of_omega_le : omega0 ≤ p → (n : Ordinal) + p = p`

traffic-ligh

Colour | Rule of thumb | Examples

--------- | -- | ---
- - - **Green** | Ordinal-specific or left-monotone lemmas | `add_lt_add_left`, `mul_lt_mul_of_pos_left`, `le_mul_righ` `opow_mul_lt_of_exp_lt` **Amber** | Generic lemmas that satisfy the 4-point rule | `mul_le_mul_left'`, `add_lt_add_of_lt_of_le` **Re** Breaks rule 2 (needs right-strict mono / commutativity) | `add_lt_add_right`, `mul_lt_mul_of_pos_right`

2.3 Multiplication (Ordinal-specific)

- `Ordinal.mul_lt_mul_of_pos_left : a < b → 0 < c → c a < c b`

- `Ordinal.mul_le_mul_iff_left : c a ≤ c b ↔ a ≤ b`
- Primed monotone helpers: `mul_le_mul_left'`, `mul_le_mul_right'` (convenient rewriting forms).
- `le_mul_right : 0 < b → a ≤ b * a`.
- `opow_mul_lt_of_exp_lt : β < α → 0 < γ → omega0 ^ β γ < omega0 ^ α` — module:* `SetTheory.Ordinal.Exponentia` — absorbs any positive right factor.

> **Note:** `mul_le_mul_left` without a trailing apostrophe comes from `Algebra.Order.Monoid.Defs` and is **generic** (ordered monoid Do **not** use it to reason about ordinal multiplication.

> **Q:** “ `library_search` **EXAMPLE SUGGESTEI** `le_mul_of_le_mul_left'` . Can I use it?” (IT CAN APPLY TO A MODULE YOU BELIEVE WILL HELP)

1. Check axioms → none found.

2. It uses only `OrderedRing` , which `Ordinal` instantiates. 3. Import adds 17 decls. □ 4. Proof is kernel-checked, no `meta` . Append one li to toolkit with a brief descrpition/justification sentence and commit.

2.4 Exponentiation (ω-powers & normality)

- `opow_add : a ^ (b + c) = a ^ b * a ^ c` — split exponents.

- `opow_pos : 0 < a → 0 < a ^ b` — positivity of powers.
- `Ordinal.opow_le_opow_right : 0 < a → b ≤ c → a ^ b ≤ a ^ c` — **use fully-qualified.**

Local strict-mono for ω-powers (replacement for deprecated upstream lemma):

```
/-- Strict-mono of ω-powers in the exponent (base  omega0 ). --/
```

`@[simp] theorem opow_lt_opow_right {b c : Ordinal} (h : b < c) : omega0 ^ b < omega0 ^ c := by simpa usi` `((Ordinal.isNormal_opow (a := omega0) one_lt_omega0).strictMono h)`

Why this is correct: `isNormal_opow` states that, for `a > 1` , the map `b ↦ a ^ b` is normal (continuous, strictly increasing). With `a` `omega0` and `one_lt_omega0` , `strictMono` yields exactly `<` from `<` in the exponent, which is what we need in μ-decrease proofs.

2.5 Cast bridges (ℕ ↔ Ordinal)

```
 @[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
```

2.6 Finite vs. infinite split helper

```
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
```

Absorption shorthands

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


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

2.7 Two-sided product monotonicity (derived helper)

```
/-- 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)
  have h₂' : c  b ≤ c  d := by
    simpa using (mul_le_mul_left'  h₂ c)
  exact le_trans h₁' h₂'
```

---

3 M-MEASURE PLAYBOOK (USED ACROSS ALL RULE PROOFS)

Goal form: for each kernel rule `Step t u`, show `mu u < mu t`. Typical shape reduces to chains like

```
ω^κ * (x + 1) ≤ ω^(x + κ')
```

Standard ladder (repeatable):

1 . **Assert base positivity:** `have ωpos : 0 < omega0 := omega0_pos`. 2. **Lift inequalities through exponents:** u `Ordinal.opow_le_opow_right ωpos h` for `≤`, and the local `opow_lt_opow_right` for `<`. 3. **Split exponents/products:** `[opow_add]` to turn exponent sums into products so product monotonicity applies cleanly. 4. **Move (≤) across products:** u `Ordinal.mul_le_mul_iff_left`, `mul_le_mul_left'`, `mul_le_mul_right'`; for `<` use `Ordinal.mul_lt_mul_of_pos_left` wit positive left factor. 5. **Absorb finite addends:** once `omega0 ≤ p`, rewrite `(n:Ordinal) + p = p` (or `1 + p = p`). 6. **Bridge success** convert `x < y + 1` ↔ `x ≤ y` via `Order.lt_add_one_iff`; introduce `x + 1 ≤ y` via `Order.add_one_le_of_lt` when chaining. **Clean arithmetic noise:** `simp` for associativity/neutral elements; `ring` or `linarith` only for integer-arithmetic side-conditions (both tactics a whitelisted).

Critical correction for `recΔ b s n` (μ-rules):

Do **not** try to relate `mu s` and `mu (delta n)`. They are **independent parameters**; the inequality `mu s ≤ mu (delta n)` is **false general**. A simple counterexample (compiles in this codebase):

```
def s : Trace := delta (delta void)      -- μ s begins with a higher ω-tower
def n : Trace := void                    -- μ (delta n) is strictly smaller
-- here: mu s > mu (delta n)
```

Structure μ-decrease proofs without assuming any structural relation between `s` and `n` beyond what the rule's right-hand side entails.

---

4 ORDER.SUCC VS `+ 1` (BRIDGE & HYGIENE)

Lean will often rewrite `p + 1` to `Order.succ p` in goals. Work with the `Order` lemmas:

- `Order.lt_add_one_iff : x < y + 1 ↔ x ≤ y`

- `Order.add_one_le_of_lt : x < y → x + 1 ≤ y`

  Keep the `Order.` prefix to avoid name resolution issues. Avoid inventing `succ_eq_add_one` —rely on these bridges instead.

  ## 5 DO-NOT-USE / DEPRECATED IN THIS PROJECT

  - **Generic** `mul_le_mul_left` (from `Algebra.Order.Monoid.Defs`) on ordinal goals. Use `Ordinal.mul_*` APIs instead.

- Old paths `Mathlib.Data.Ordinal.` — *replaced by* `Mathlib.SetTheory.Ordinal.`.
- `Ordinal.opow_lt_opow_right` (upstream removed). Use the **local** `opow_lt_opow_right` defined in §2.4.
- `le_of_not_lt` (deprecated) — use `le_of_not_gt`.

---

6 MINIMAL IMPORT PRELUDE (COPY-PASTE)

```
import Init.WF
```

`import Mathlib.Data.Prod.Lex import Mathlib.SetTheory.Ordinal.Basic import Mathlib.SetTheory.Ordinal.Arithmet import Mathlib.SetTheory.Ordinal.Exponential import Mathlib.Algebra.Order.SuccPred impo Mathlib.Data.Nat.Cast.Order.Basic import Mathlib.Tactic.Linarith import Mathlib.Tactic.Ring open Ordinal`

---

7 READY-MADE SNIPPETS

Nat-sized measure (optional helper):

```
@[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
```

WF via ordinal μ:

```
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
```

---

8 CROSS-FILE CONSISTENCY NOTES

- This toolkit and **AGENT.md (2025-07-29)** are **synchronized**: imports, prefixes, do-not-use list, and the μ-rule correction are identical. If you e one, mirror the change here.

- Cite lemma modules explicitly in comments or nearby text in code reviews to prevent regressions (e. "`Ordinal.mul_lt_mul_of_pos_left` — from `SetTheory.Ordinal.Arithmetic`").

---

9 CHECKLIST (BEFORE SENDING A PR)

- [ ] Imports ⊆ §6, no stray module paths.

- [ ] All exponent/product/`+1` lemmas called with **qualified** names as in §1.
- [ ] μ-proofs avoid any relation between `μ s` and `μ (δ n)` in `recΔ b s n`.
- [ ] Tactics limited to `simp`, `linarith`, `ring`.
- [ ] No generic `mul_le_mul_left` on ordinal goals; use `Ordinal.mul_*` API.
- [ ] SN proof provides μ-decrease on all 8 rules; WF via `InvImage.wf`.
- [ ] Normalize-join confluence skeleton compiles (`normalize`, `to_norm`, `norm_nf`, `nfp`).

End of file.