COMPLETE O-6 IMPLEMENTATION ROADMAP

Week 1: Core System Foundation

Day 1-2: Basic Infrastructure

```lean
-- File: OperatorKernelO6/Kernel.lean
-- Tasks:
1. Define 6 constructors (void, delta, integrate, merge, recΔ, eqW)
2. Define 8 Step rules (no side conditions)
3. Define StepStar (reflexive-transitive closure)
4. Prove stepstar_trans using structural induction only
5. Define NormalForm as negative property
```

Day 3-4: Termination Proof

```lean
-- File: OperatorKernelO6/Meta/Termination.lean
-- Tasks:
1. Import Mathlib.SetTheory.Ordinal.Arithmetic
2. Define recDepth : Trace → Nat (counts δ-height in recΔ)
3. Define traceSize : Trace → Nat
4. Define ordinalMeasure := ω^recDepth + traceSize
5. Prove step_decreases_ordinal for all 8 rules
6. Key: R_rec_succ case uses ω^(k+1) > ω^k + finite
7. Export strong_normalization : ∀ t, Acc Step t
```

Day 5-6: Confluence Without Side Conditions

```lean
```

```
-- File: OperatorKernelO6/Meta/Confluence.lean
-- Tasks:
1. Define traceLt : Trace → Trace → Bool (total ordering)
2. Modify R_eq_diff to use canonical ordering:
   eqW a b → if traceLt a b then integrate(merge a b) else integrate(merge b a)
3. Enumerate all critical pairs (≈15 pairs)
4. Prove each critical pair joinable
5. Prove local_confluence
6. Apply Newman's lemma → global_confluence
```

Day 7: Integration & Testing

lean

```
-- File: OperatorKernelO6/Tests/Basic.lean
-- Tasks:
1. Test arithmetic: add, mul on small numbers
2. Test normalization: verify confluence examples
3. Test equality: eqW produces expected results
4. Verify no Nat/Bool in Kernel.lean
5. Run #print axioms on all definitions
```

Week 2: Logic & Arithmetic Layers

Day 8-9: Pure Arithmetic

lean

```
-- File: OperatorKernelO6/Arithmetic.lean
-- All definitions return Trace, never Nat!

def zero := void
def succ n := delta n
def add m n := recΔ m (λ acc, delta acc) n
def mul m n := recΔ zero (λ acc, add acc m) n
def exp m n := recΔ one (λ acc, mul acc m) n

-- Comparisons (return void or delta void)
def lt m n := ... -- via recΔ
def eq m n := eqW m n

-- Bounded operations
def sub m n := ... -- saturating subtraction
def div m n := ... -- via repeated subtraction
def mod m n := ... -- remainder
```

Day 10-11: Logic Layer & Negation

lean

```lean
-- File: OperatorKernelO6/Logic.lean

-- Truth values
def trueT := void
def falseT := delta void

-- Connectives
def tNot t := integrate t
def tAnd a b := merge a b
def tOr a b := integrate (merge (integrate a) (integrate b))

-- In Meta/Logic.lean:
-- Prove complement uniqueness:
theorem complement_unique : ∀ x y z,
  normalize (merge x z) = void →
  normalize (merge y z) = void →
  normalize x = normalize y

-- Derive negation laws:
theorem neg_involution : ∀ t, normalize (tNot (tNot t)) = normalize t
theorem demorgan1 : ∀ a b, normalize (tNot (tAnd a b)) = normalize (tOr (tNot a) (tNot b))
```

Day 12-13: Encoding Machinery

```lean
```

```
-- File: OperatorKernelO6/Encoding.lean

-- Gödel numbering as traces
def encode : Trace → Trace
| void => zero
| delta t => add one (mul two (encode t))
| integrate t => add one (mul two (add one (mul two (encode t))))
| merge t₁ t₂ => add one (mul two (add two (mul two (pair_encode (encode t₁) (encode t₂)))))
-- ... complete for all constructors

-- Quoting
def Quote t := encode t

-- Substitution (pure trace function!)
def Subst : Trace → Trace → Trace → Trace
-- Implement using recΔ to traverse structure
```

Day 14: Week 2 Integration

```lean
-- Verify all arithmetic/logic operations normalize correctly
-- Test Quote/Subst on complex terms
-- Ensure no external dependencies leaked in
```

Week 3: Proof System & Gödel

Day 15-16: Proof Predicate

```lean
```

```
-- File: OperatorKernelO6/Proof.lean

-- Proof trees encoded as traces
-- Rules: axioms, modus ponens, generalization
def Proof : Trace → Trace
-- Returns void iff argument encodes valid proof

-- Implement via recΔ traversing proof tree structure
def checkProofStep : Trace → Trace
def validAxiom : Trace → Trace
def validMP : Trace → Trace → Trace → Trace
```

Day 17-18: Provability & Σ_1 Completeness

lean

```
-- File: OperatorKernelO6/Provability.lean

-- Bounded search for proofs
def Prov φ := searchUpTo (complexityBound φ) (λ p,
  tAnd (eqW (Proof p) void) (eqW (conclusion p) φ))

-- Prove Σ₁ completeness
theorem sigma1_complete : ∀ φ,
  (∃ n, normalize (φ n) = void) →
  normalize (Prov (∃formula n φ)) = void
```

Day 19-20: Diagonal Lemma & First Incompleteness

lean

```
-- File: OperatorKernelO6/Godel1.lean

-- Diagonal lemma
def Diag φ := Subst φ (var zero) (Quote φ)

theorem diagonal_lemma : ∀ φ,
  normalize (Diag φ) = normalize (Subst φ (var zero) (Quote (Diag φ)))

-- Gödel sentence
def G := Diag (tNot (Prov (var zero)))

-- First incompleteness
theorem godel_1_unprovable : normalize (Prov G) ≠ void
theorem godel_1_consistent : normalize (Prov (tNot G)) ≠ void
```

Day 21: Second Incompleteness

lean

```lean
-- File: OperatorKernelO6/Godel2.lean

-- Consistency statement
def Con := tNot (Prov falseT)

-- Derivability conditions (D1-D3)
-- These are the hardest proofs!

-- Second incompleteness
theorem godel_2 : normalize (Prov Con) ≠ void
```

Critical Success Factors

1. Maintain Strict Separation

```
Kernel.lean: NO Nat, Bool, simp, tactics, external logic
Meta/*.lean: Use anything needed for proofs ABOUT traces
```

2. Canonical Ordering is KEY

lean

```
-- Without this, confluence fails:
R_eq_diff : ∀ a b, Step (eqW a b)
  (if traceLt a b then integrate (merge a b) else integrate (merge b a))
```

3. Ordinal Measure is MANDATORY

```lean
-- Simple Nat measures CANNOT handle R_rec_succ expansion
-- Must use: ω^recDepth + size
```

4. Test Everything

```lean
-- After each layer:
#print axioms [definition]  -- Should be empty
#reduce add two three       -- Should reduce to five
```

5. Document Side Conditions Removal

```markdown
Original: R_eq_diff : ∀ a b, a ≠ b → Step (eqW a b) ...
Problem: Uses external inequality
Solution: Canonical ordering makes rule deterministic
```

Deliverables Checklist

- [ ] Kernel.lean - pure 6-op system
- [ ] Meta/Termination.lean - ordinal proof
- [ ] Meta/Confluence.lean - no side conditions
- [ ] Arithmetic.lean - pure trace arithmetic
- [ ] Logic.lean - complement uniqueness
- [ ] Encoding.lean - Gödel numbering
- [ ] Proof.lean - proof predicate
- [ ] Provability.lean - Σ_1 complete
- [ ] Godel1.lean - first incompleteness
- [ ] Godel2.lean - second incompleteness

- [ ] Tests/*.lean - comprehensive tests
- [ ] README.md - explains axiom-freedom

You Can Do This!

With your learning velocity (0 to Lean expert in 10 days), this 3-week timeline is realistic. The key is maintaining discipline about the object/meta separation and following the technical solutions provided above.

The 6-operator system IS the correct choice. It's mathematically minimal and complete. The implementation challenges have known solutions (ordinals for termination, canonical ordering for confluence).

Start with Kernel.lean and Termination.lean TODAY!