# Complement

## Overview

Complement operations and properties

## Source Code

```
import OperatorKernelO6.Kernel
import OperatorKernelO6.Meta.Confluence

open OperatorKernelO6 Trace

namespace OperatorKernelO6.Meta

-- Complement operation using integration
def complement (t : Trace) : Trace := integrate t

-- Negation is involutive via double integration cancellation
theorem complement_involution (t : Trace) :
  ∃ u, StepStar (complement (complement t)) u ∧ StepStar t u := by
  unfold complement
  cases t with
  | void =>
    use void
    constructor
    · apply stepstar_of_step; apply R_int_delta
    · apply StepStar.refl
  | delta s =>
    use void
    constructor
    · apply stepstar_of_step; apply R_int_delta
    · sorry -- Need to show delta s reduces somehow
  | integrate s =>
    use s
    constructor
    · apply stepstar_of_step; apply R_int_delta
    · apply StepStar.refl
  | merge a b =>
    sorry -- Complex case
  | recΔ b s n =>
    sorry -- Complex case
  | eqW a b =>
    sorry -- Complex case

-- Complement uniqueness via normal forms
theorem complement_unique {t u v : Trace}
  (h1 : StepStar (complement t) u) (h2 : StepStar (complement t) v)
  (hu : NormalForm u) (hv : NormalForm v) : u = v := by
  -- Use confluence to get common reduct, then use normal form uniqueness
  have ⟨w, hw1, hw2⟩ := confluence h1 h2
  have : u = w := nf_no_stepstar_forward hu hw1
  have : v = w := nf_no_stepstar_forward hv hw2
  rw [‹u = w›, ‹v = w›]

-- De Morgan laws
theorem demorgan1 (a b : Trace) :
  ∃ c d, StepStar (complement (merge a b)) c ∧
         StepStar (merge (complement a) (complement b)) d ∧
         ∃ e, StepStar c e ∧ StepStar d e := by
  sorry -- Requires detailed case analysis

theorem demorgan2 (a b : Trace) :
  ∃ c d, StepStar (merge (complement a) (complement b)) c ∧
         StepStar (complement (merge a b)) d ∧
         ∃ e, StepStar c e ∧ StepStar d e := by
  sorry -- Dual of demorgan1

end OperatorKernelO6.Meta
```