FixedPoint

File: `C:\Users\Moses\math_ops\OperatorKernelO6\OperatorKernelO6\Meta\FixedPoint.lean`
Type: lean
Generated: 2025-08-05 03:41:07
Size: 1674 characters

Overview

Fixed point theorems and applications

Source Code

```
import OperatorKernelO6.Kernel
import OperatorKernelO6.Meta.Termination

open OperatorKernelO6 Trace

namespace OperatorKernelO6.Meta

-- Normalization function (placeholder - uses strong normalization)
def normalize (t : Trace) : Trace := by
  -- In a complete implementation, this would reduce t to normal form
  -- For now, we'll use a placeholder that relies on strong normalization
  sorry

-- Equivalence via normalization
def Equiv (x y : Trace) : Prop := normalize x = normalize y

-- Fixed point witness structure
structure FixpointWitness (F : Trace → Trace) where
  ψ     : Trace
  fixed : Equiv ψ (F ψ)

-- Constructor for fixed point witness
theorem mk_fixed {F} {ψ} (h : Equiv ψ (F ψ)) : FixpointWitness F :=
□ψ, h□

-- Idempotent functions have fixed points
theorem idemp_fixed {F : Trace → Trace}
  (h : ∀ t, Equiv (F t) (F (F t))) :
  FixpointWitness F :=
□F Trace.void, by
  have := h Trace.void
  exact this□

-- Fixed point theorem for continuous functions (diagonal construction)
theorem diagonal_fixed (F : Trace → Trace) : ∃ ψ, Equiv ψ (F ψ) := by
  -- This is the key theorem for Gödel's diagonal lemma
  let diag := λ x => F (recΔ x x x)  -- Self-application via recΔ
  let ψ := diag (delta void)         -- Apply to some base term
  use ψ
  sorry -- Detailed proof requires careful analysis of recΔ unfolding

-- Fixed point uniqueness under normalization
theorem fixed_unique {F : Trace → Trace} {ψ₁ ψ₂ : Trace}
  (h₁ : Equiv ψ₁ (F ψ₁)) (h₂ : Equiv ψ₂ (F ψ₂))
  (hF : ∀ x y, Equiv x y → Equiv (F x) (F y)) : -- F respects equivalence
  Equiv ψ₁ ψ₂ := by
  sorry -- Follows from confluence and uniqueness of normal forms

end OperatorKernelO6.Meta
```