Normalize

File: `C:\Users\Moses\math_ops\OperatorKernelO6\OperatorKernelO6\Meta\Normalize.lean`
Type: lean
Generated: 2025-08-05 03:41:06
Size: 4135 characters

Overview

Normalization procedures and proofs

Source Code

```
import OperatorKernelO6.Kernel
import OperatorKernelO6.Meta.Termination  -- adjust if your SN file lives elsewhere

open Classical
open OperatorKernelO6 Trace Step

namespace OperatorKernelO6.Meta

-- Simple structural size measure for termination
@[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

-- Lemma: every step decreases structural size
theorem step_size_decrease {t u : Trace} (h : Step t u) : size u < size t := by
  cases h <;> simp [size] <;> linarith

noncomputable def normalize : Trace → Trace
| t =>
  if h : ∃ u, Step t u then
    let u := Classical.choose h
    have hu : Step t u := Classical.choose_spec h
    normalize u
  else t
termination_by
  normalize t => size t
decreasing_by
  simp_wf
  exact step_size_decrease (Classical.choose_spec h)

theorem to_norm : ∀ t, StepStar t (normalize t)
| t =>
  by
    classical
    by_cases h : ∃ u, Step t u
    ·
      let u := Classical.choose h
      have hu : Step t u := Classical.choose_spec h
      have ih := to_norm u
      simpa [normalize, h, u, hu] using StepStar.tail hu ih
    ·
      simpa [normalize, h] using StepStar.refl t
termination_by
  to_norm t => size t
```

```
decreasing_by
  simp_wf
  exact step_size_decrease (Classical.choose_spec h)

theorem norm_nf : ∀ t, NormalForm (normalize t)
| t =>
  by
    classical
    by_cases h : ∃ u, Step t u
    ·
      let u := Classical.choose h
      have hu : Step t u := Classical.choose_spec h
      have ih := norm_nf u
      simpa [normalize, h, u, hu] using ih
    ·
      intro ex
      rcases ex with ⟨u, hu⟩
      have : Step t u := by simpa [normalize, h] using hu
      exact h ⟨u, this⟩
termination_by
  norm_nf t => size t
decreasing_by
  simp_wf
  exact step_size_decrease (Classical.choose_spec h)

theorem nfp {a b n : Trace} (hab : StepStar a b) (han : StepStar a n) (hn : NormalForm n) :
  StepStar b n := by
  revert b
  induction han with
  | refl =>
      intro b hab _; exact hab
  | tail h an han ih =>
      intro b hab hn'
      cases hab with
      | refl => exact False.elim (hn' ⟨_, h⟩)
      | tail h' hbn => exact ih hbn hn'

def Confluent : Prop :=
  ∀ {a b c}, StepStar a b → StepStar a c → ∃ d, StepStar b d ∧ StepStar c d

theorem global_confluence : Confluent := by
  intro a b c hab hac
  let n := normalize a
  have han : StepStar a n := to_norm a
  have hbn : StepStar b n := nfp hab han (norm_nf a)
  have hcn : StepStar c n := nfp hac han (norm_nf a)
  exact ⟨n, hbn, hcn⟩

-- Corollary: Normalization is idempotent
theorem normalize_idempotent (t : Trace) : normalize (normalize t) = normalize t := by
  have hnf : NormalForm (normalize t) := norm_nf t
  unfold NormalForm at hnf
  push_neg at hnf
  unfold normalize
  simp [hnf]

-- Corollary: Normal forms are unique
theorem unique_normal_forms {a b : Trace} (ha : NormalForm a) (hb : NormalForm b)
    (hab : ∃ c, StepStar c a ∧ StepStar c b) : a = b := by
  rcases hab with ⟨c, hca, hcb⟩
  have ha_eq : normalize c = a := by
    have hnorm : StepStar c (normalize c) := to_norm c
    have huniq := nfp hca hnorm (norm_nf c)
    exact nf_no_stepstar_forward ha huniq
  have hb_eq : normalize c = b := by
    have hnorm : StepStar c (normalize c) := to_norm c
    have huniq := nfp hcb hnorm (norm_nf c)
    exact nf_no_stepstar_forward hb huniq
  rw [←ha_eq, hb_eq]

-- Church-Rosser property: joinability characterization
-- NOTE: The full Church-Rosser theorem (joinable ↔ convertible) requires
-- the conversion relation (symmetric-reflexive-transitive closure)
-- Here we prove the easier direction and note the limitation
```

```
-- Here we prove the easier direction and note the limitation
theorem church_rosser_half {a b : Trace} :
    (∃ d, StepStar d a ∧ StepStar d b) → (∃ c, StepStar a c ∧ StepStar b c) := by
  intro ⟨d, hda, hdb⟩
  exact global_confluence hda hdb

-- The converse (joinable implies common source) is not generally true
-- for reduction relations - it would require co-confluence
-- We can only prove it holds in very special cases or with additional structure

end OperatorKernelO6.Meta
```