Untitled

```
import OperatorKernelO6.Kernel open OperatorKernelO6 Trace Step
namespace OperatorKernelO6.Meta  -- meta-layer may use Nat, tactics
def sz : Trace → Nat | void => 1 | delta t => sz t + 1 | integrate t => sz t + 1 | merge a b => sz a + sz b + 1 |
recΔ b s n => sz b + sz s + sz n + 1 | eqW a b => sz a + sz b + 1
theorem step_sz {a b} (h : Step a b) : sz b < sz a := by cases h <;> simp [sz, Nat.add_comm,
Nat.add_left_comm, Nat.add_assoc]
open WellFounded
theorem strong_norm : ∀ t : Trace, Acc Step t := by have wf : WellFounded (measure sz) := measure_wf sz
intro t refine (wf.apply t).induction ?_ intro x ih apply Acc.intro intro y hy have : measure sz y x := step_sz hy
exact ih _ this
end OperatorKernelO6.Meta
attribute [instance] OperatorKernelO6.Meta.strong_norm  -- for future use

Termination.lean:20:24
Application type mismatch: In the application
  WellFounded (measure sz)
the argument
  measure sz
has type
  WellFoundedRelation Trace : Type
but is expected to have type
  ?m.793 → ?m.793 → Prop : Sort (max 1 ?u.792)
Termination.lean:20:40
unknown identifier 'measure_wf'
Termination.lean:19:49
unsolved goals
⊢ ∀ (t : Trace), Acc Step t

CONTAINS SORRY AND ADMIT
import OperatorKernelO6.Kernel import Mathlib.Tactic.Linarith open OperatorKernelO6 Trace Step
namespace OperatorKernelO6.Meta
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 {a b : Trace} (h : Step a b) : size b < size a := by cases h <;> simp [size] <;> linarith
open WellFounded
theorem strong_norm : ∀ t : Trace, Acc Step t := by have wf : WellFounded (measure size) := measure_wf size
intro t refine (wf.apply t).induction ?_ intro x ih apply Acc.intro intro y hy have : measure size y x := step_size
```

hy exact ih _ this
partial def normalize (t : Trace) : Trace := match t with | void => void | delta u => delta (normalize u) | integrate u => match normalize u with | delta v => void | w => integrate w | merge a b => let na := normalize a let nb := normalize b if na = void then nb else if nb = void then na else if na = nb then na else merge na nb | recΔ b s n => match normalize n with | void => normalize b | delta n' => let nb := normalize b let ns := normalize s merge ns (normalize (recΔ nb ns n')) | n' => recΔ (normalize b) (normalize s) n' | eqW a b => let na := normalize a let nb := normalize b if na = nb then void else integrate (merge na nb) termination_by size t decreasing_by simp_wf_tac first | apply step_size | simp [size] ; linarith
theorem normalize_idempotent : ∀ t : Trace, normalize (normalize t) = normalize t := by intro t have : size (normalize t) ≤ size t := by have h := strong_norm t cases h with | intro _ go => have rec : ∀ x, Acc Step x → size (normalize x) ≤ size x := by intro x acc cases acc with | intro _ k => induction x with | void => simp [normalize, size] | delta x ih => have := rec x (k _ (Step.int_delta x))? sorry admit
end OperatorKernelO6.Meta

Termination.lean:16:30 linarith failed to find a contradiction case R_rec_succ.h b† s† n† : Trace a† : size b† + size s† + (size n† + 1) ≤ size s† + (size b† + size s† + size n† + 1) ⊢ False
failed Termination.lean:16:30 linarith failed to find a contradiction case R_eq_refl.h a†¹ : Trace a† : size a†¹ ≤ 0 ⊢ False
failed Termination.lean:16:30 linarith failed to find a contradiction case R_eq_diff a† b† : Trace ⊢ False
failed Termination.lean:21:24 Application type mismatch: In the application WellFounded (measure size) the argument measure size has type WellFoundedRelation Trace : Type but is expected to have type ?m.889 → ?m.889 → Prop : Sort (max 1 ?u.888) Termination.lean:21:42 unknown identifier 'measure_wf' Termination.lean:20:49 unsolved goals ⊢ ∀ (t : Trace), Acc Step t Termination.lean:60:5 unknown tactic Termination.lean:71:8 unexpected syntax failed to pretty print term (use 'set_option pp.rawOnError true' for raw representation) Termination.lean:65:85 unsolved goals t : Trace this : size (normalize t) ≤ size t ⊢ normalize (normalize t) = normalize t Termination.lean:71:13 unexpected rec

import OperatorKernelO6.Kernel namespace OperatorKernelO6.Meta -- meta-layer may use Nat, tactics
def sz : Trace → Nat | void => 1 | delta t => sz t + 1 | integrate t => sz t + 1 | merge a b => sz a + sz b + 1 | recΔ b s n => sz b + sz s + sz n + 1 | eqW a b => sz a + sz b + 1
theorem step_sz {a b} (h : Step a b) : sz b < sz a := by cases h <;> simp [sz, Nat.add_comm, Nat.add_left_comm, Nat.add_assoc]
open WellFounded
theorem strong_norm : ∀ t : Trace, Acc Step t := by have wf : WellFounded (measure sz) := measure_wf sz intro t refine (wf.apply t).induction ?_ intro x ih apply Acc.intro intro y hy have : measure sz y x := step_sz hy exact ih _ this
end OperatorKernelO6.Meta
attribute [instance] OperatorKernelO6.Meta.strong_norm -- for future use

All Messages (10) Termination.lean:5:2 Local variable 'void' resembles constructor 'OperatorKernelO6.Trace.void' - write '.void' (with a dot) or 'OperatorKernelO6.Trace.void' to use the constructor.
Note: This linter can be disabled with `set_option linter.constructorNameAsVariable false`
Termination.lean:6:2 Invalid pattern: Expected a constructor or constant marked with `[match_pattern]`
Hint: 'OperatorKernelO6.Trace.delta' is similar Termination.lean:13:20 declaration uses 'sorry'
Termination.lean:13:24 This simp argument is unused: Nat.add_comm
Hint: Omit it from the simp argument list. simp [sz, Nat.add_comm, Nat.add_left_comm, Nat.add_assoc]
Note: This linter can be disabled with `set_option linter.unusedSimpArgs false` Termination.lean:13:38
This simp argument is unused: Nat.add_left_comm
Hint: Omit it from the simp argument list. simp [sz, Nat.add_comm, Nat.add_left_comm, Nat.add_assoc]
Note: This linter can be disabled with `set_option linter.unusedSimpArgs false` Termination.lean:13:57
This simp argument is unused: Nat.add_assoc
Hint: Omit it from the simp argument list. simp [sz, Nat.add_comm, Nat.add_left_comm, Nat.add_assoc]
Note: This linter can be disabled with `set_option linter.unusedSimpArgs false` Termination.lean:12:54
unsolved goals case R_int_delta t† : Trace ⊢ False
case R_merge_void_left b : Trace ⊢ False
case R_merge_void_right b : Trace ⊢ False
case R_merge_cancel b : Trace ⊢ False
case R_rec_zero b s† : Trace ⊢ False
case R_rec_succ b† s† n† : Trace ⊢ False
case R_eq_refl a† : Trace ⊢ False
case R_eq_diff a† b† : Trace ⊢ False Termination.lean:18:24 Application type mismatch: In the application WellFounded (measure sz) the argument measure sz has type WellFoundedRelation Trace : Type but is expected to have type ?m.67 → ?m.67 → Prop : Sort (max 1 ?u.66) Termination.lean:18:40 unknown identifier 'measure_wf' Termination.lean:17:49 unsolved goals ⊢ ∀ (t : Trace), Acc Step t

——————————————— Arithmetic.lean ———————————————

```
import OperatorKernelO6.Kernel
open OperatorKernelO6 Trace
namespace OperatorKernelO6
def numeral : Nat → Trace
| 0    => void
| n + 1 => delta (numeral n)
def succ (t : Trace) : Trace := delta t
def add (m n : Trace) : Trace := recΔ n delta m
def mul (m n : Trace) : Trace := recΔ void (merge n) m
end OperatorKernelO6
```

All Messages (2)
Arithmetic.lean:12:40
Application type mismatch: In the application

```
  n.recΔ delta
the argument
  delta
has type
  Trace → Trace : Type
but is expected to have type
  Trace : Type
Arithmetic.lean:14:43
Application type mismatch: In the application
  void.recΔ n.merge
the argument
  n.merge
has type
  Trace → Trace : Type
but is expected to have type
  Trace : Type
```