Physlib Monthly Report

1–30 September 2025

Contributors: CasualPhysicsEnjoier, Joseph Tooby-Smith, Miyahara Kō, Shlok Vaibhav Singh

Reviewers: Joseph Tooby-Smith, Zhi Kai Pong

Generated 14 August 2026 from commit `9a8fe27`.

Abstract

Physlib is a community library of formalised physics for the Lean 4 proof assistant: definitions and results from across physics, stated in a formal language and checked by machine. This report is an automatically generated record of one month's additions to it.

It covers 1–30 September 2025 on branch `master` of `leanprover-community/physlib`, between commits `a439539` and `9a8fe27`. Over that period 4 contributors landed 41 commits, changing 19,825 lines across 146 files and adding 675 new Lean declarations, each catalogued with its statement in Section 2. The complete diff is available at GitHub compare.

Contents

1 Introduction **2**
1.1 Contributors . . . 2
1.2 Reviewers . . . 2
1.3 Modified subfolders . . . 2
1.4 New declarations by kind . . . 3

2 Details by file **3**
2.1 `PhysLean.ClassicalMechanics.HarmonicOscillator.Basic` . . . 3
2.2 `PhysLean.ClassicalMechanics.HarmonicOscillator.Solution` . . . 7
2.3 `PhysLean.ClassicalMechanics.RigidBody.Basic` . . . 9
2.4 `PhysLean.ClassicalMechanics.RigidBody.SolidSphere` . . . 10
2.5 `PhysLean.Electromagnetism.Electrostatics.ThreeDimension.FiniteCollection` 10
2.6 `PhysLean.Electromagnetism.Electrostatics.ThreeDimension.PointParticle` 11
2.7 `PhysLean.Electromagnetism.KineticTerm` . . . 16
2.8 `PhysLean.Electromagnetism.LorentzAction` . . . 17
2.9 `PhysLean.Electromagnetism.Potential` . . . 18
2.10 `PhysLean.Mathematics.Calculus.Divergence` . . . 22
2.11 `PhysLean.Mathematics.Distribution.Function.IsDistBounded` . . . 23
2.12 `PhysLean.Mathematics.Distribution.Function.OfFunction` . . . 26
2.13 `PhysLean.Mathematics.Trigonometry.Tanh` . . . 27
2.14 `PhysLean.Mathematics.VariationalCalculus.HasVarAdjDeriv` . . . 28
2.15 `PhysLean.Mathematics.VariationalCalculus.HasVarAdjoint` . . . 29
2.16 `PhysLean.Particles.StandardModel.Basic` . . . 29
2.17 `PhysLean.Particles.StandardModel.HiggsBoson.Basic` . . . 30
2.18 `PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.AllowsTerm` . . . 33
2.19 `PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.Basic` . . . 36

2.20 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.Completions . . . 40
2.21 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.Map 42
2.22 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.MinimallyAllowsTerm.Basic 45
2.23 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.MinimallyAllowsTerm.FinsetTerms 46
2.24 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.MinimallyAllowsTerm.OfFinset 47
2.25 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.MinimalSuperSet . 49
2.26 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.OfFieldLabel . . . 52
2.27 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.OfPotentialTerm . 53
2.28 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.PhenoClosed . . . 55
2.29 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.PhenoConstrained 57
2.30 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.Yukawa 59
2.31 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.ZMod 61
2.32 PhysLean.QFT.AnomalyCancellation.Basic 62
2.33 PhysLean.QuantumMechanics.OneDimension.ReflectionlessPotential.Basic 63
2.34 PhysLean.Relativity.LorentzGroup.Boosts.Basic 64
2.35 PhysLean.Relativity.Tensors.Basic 64
2.36 PhysLean.Relativity.Tensors.Product 64
2.37 PhysLean.Relativity.Tensors.RealTensor.CoVector.Basic 65
2.38 PhysLean.Relativity.Tensors.RealTensor.Vector.Basic 69
2.39 PhysLean.Relativity.Tensors.RealTensor.Vector.Pre.Basic 69
2.40 PhysLean.Relativity.Tensors.RealTensor.Vector.Pre.Modules 70
2.41 PhysLean.Relativity.Tensors.Tensorial 70
2.42 PhysLean.SpaceAndTime.Space.Basic 71
2.43 PhysLean.SpaceAndTime.Space.Distributions 71
2.44 PhysLean.SpaceAndTime.Space.Translations 72
2.45 PhysLean.SpaceAndTime.Space.VectorIdentities 74
2.46 PhysLean.SpaceAndTime.SpaceTime.Basic 74
2.47 PhysLean.SpaceAndTime.SpaceTime.Boosts 76
2.48 PhysLean.SpaceAndTime.Time.Basic 76
2.49 PhysLean.StringTheory.FTheory.SU5.Charges.AnomalyFree 76
2.50 PhysLean.StringTheory.FTheory.SU5.Charges.Viable 77
2.51 PhysLean.StringTheory.FTheory.SU5.Fluxes.Basic 81
2.52 PhysLean.StringTheory.FTheory.SU5.Quanta.Basic 81
2.53 PhysLean.Units.Basic 81
2.54 PhysLean.Units.Dimension 82
2.55 PhysLean.Units.Examples 84
2.56 PhysLean.Units.FDeriv 86
2.57 PhysLean.Units.Integral 87
2.58 PhysLean.Units.UnitDependent 87
2.59 PhysLean.Units.WithDim.Basic 89
2.60 PhysLean.Units.WithDim.Speed 90
2.61 scripts.MetaPrograms.module_doc_lint 91

1 Introduction

During Sep 2025, 4 contributors submitted 41 commits that changed 19,825 lines (+14,375 / -5,450) across 146 files spanning 22 top-level areas of the repository. Of those, 27 files were newly added and 10 were removed outright. This report catalogues 675 newly added Lean declarations: 512 lemmas, 105 defs, 49 instances, 3 structures, 2 abbrevs, 1 informal definition, 1 informal lemma, 1 theorem, and 1 inductive. We're delighted to recognize the following first-time contributors this month: Miyahara Kō and Shlok Vaibhav Singh.

See the full compare diff and the list of commits on GitHub for the raw source.

1.1 Contributors

The following individuals authored commits merged during this reporting period, listed alphabetically.

Contributor	Lines Changed	Commits
CasualPhysicsEnjoier	395	12
Joseph Tooby-Smith	19,046	26
Miyahara Kō	45	2
Shlok Vaibhav Singh	89	1

1.2 Reviewers

The following individuals approved pull requests during this reporting period, listed alphabetically.

Reviewer	PRs Reviewed
Joseph Tooby-Smith	6
Zhi Kai Pong	1

1.3 Modified subfolders

Subfolder	Files	New	Deleted	Additions	Deletions
`PhysLean/Particles`	39	+5	-8	+4,261	-2,310
`PhysLean/Electromagnetism`	9	+4	–	+2,792	-88
`PhysLean/Mathematics`	15	+3	-1	+1,450	-847
`PhysLean/Units`	8	+2	–	+1,210	-760
`PhysLean/ClassicalMechanics`	8	+4	–	+1,278	-331
`PhysLean/StringTheory`	9	–	-1	+679	-762
`PhysLean/Relativity`	18	+1	–	+970	-102
`scripts/MetaPrograms`	2	+2	–	+565	-0
`PhysLean/SpaceAndTime`	10	+2	–	+501	-46
`PhysLean/QFT`	5	–	–	+348	-30
`PhysLean/StatisticalMechanics`	4	–	–	+73	-88
`PhysLean/QuantumMechanics`	6	–	–	+91	-32
`scripts`	4	+2	–	+61	-2
`PhysLean.lean`	1	–	–	+30	-19
`lake-manifest.json`	1	–	–	+21	-21
`.github/workflows`	1	+1	–	+30	-0

PhysLean/CondensedMatter	1	–	–	+7	-7
lakefile.toml	1	–	–	+6	-2
README.md	1	–	–	+1	-1
lean-toolchain	1	–	–	+1	-1
PhysLean/Thermodynamics	1	–	–	+0	-1
docs	1	+1	–	+0	-0

1.4 New declarations by kind

Kind	Count
lemma	512
def	105
instance	49
structure	3
abbrev	2
informal_definition	1
informal_lemma	1
theorem	1
inductive	1

2 Details by file

2.1 PhysLean.ClassicalMechanics.HarmonicOscillator.Basic

PhysLean/ClassicalMechanics/HarmonicOscillator/Basic.lean on GitHub.

lemma kineticEnergy_eq [source]

```
lemma kineticEnergy_eq (xₜ : Time → Space 1) :
    kineticEnergy S xₜ = fun t => (1 / (2 : ℝ)) * S.m * ⟪∂ₜ xₜ t, ∂ₜ xₜ t⟫_ℝ
```

lemma potentialEnergy_eq [source]

```
lemma potentialEnergy_eq (x : Space 1) :
    potentialEnergy S x = (1 / (2 : ℝ)) • S.k • ⟪x, x⟫_ℝ
```

lemma energy_eq [source]

```
lemma energy_eq (xₜ : Time → Space 1) :
    energy S xₜ = fun t => kineticEnergy S xₜ t + potentialEnergy S (xₜ t)
```

lemma kineticEnergy_differentiable [source]

```
@[fun_prop]
lemma kineticEnergy_differentiable (xₜ : Time → Space 1) (hx : ContDiff ℝ ∞ xₜ) :
    Differentiable ℝ (kineticEnergy S xₜ)
```

lemma potentialEnergy_differentiable [source]

```
@[fun_prop]
lemma potentialEnergy_differentiable (xₜ : Time → Space 1) (hx : ContDiff ℝ ∞ xₜ) :
    Differentiable ℝ (fun t => potentialEnergy S (xₜ t))
```

lemma energy_differentiable [source]

```
@[fun_prop]
lemma energy_differentiable (xₜ : Time → Space 1) (hx : ContDiff ℝ ∞ xₜ) :
    Differentiable ℝ (energy S xₜ)
```

lemma kineticEnergy_deriv [source]

```
lemma kineticEnergy_deriv (xₜ : Time → Space 1) (hx : ContDiff ℝ ∞ xₜ) :
    ∂ₜ (kineticEnergy S xₜ) = fun t => ⟪∂ₜ xₜ t, S.m • ∂ₜ (∂ₜ xₜ) t⟫_ℝ
```

lemma potentialEnergy_deriv [source]

```
lemma potentialEnergy_deriv (xₜ : Time → Space 1) (hx : ContDiff ℝ ∞ xₜ) :
    ∂ₜ (fun t => potentialEnergy S (xₜ t)) = fun t => ⟪∂ₜ xₜ t, S.k • xₜ t⟫_ℝ
```

lemma energy_deriv [source]

```
lemma energy_deriv (xₜ : Time → Space 1) (hx : ContDiff ℝ ∞ xₜ) :
    ∂ₜ (energy S xₜ) = fun t => ⟪∂ₜ xₜ t, S.m • ∂ₜ (∂ₜ xₜ) t + S.k • xₜ t⟫_ℝ
```

lemma lagrangian_eq [source]

```
@[nolint unusedArguments]
lemma lagrangian_eq : lagrangian S = fun t x v =>
    1 / (2 : ℝ) * S.m * ⟪v, v⟫_ℝ - 1 / (2 : ℝ) * S.k * ⟪x, x⟫_ℝ
```

lemma lagrangian_eq_kineticEnergy_sub_potentialEnergy [source]

```
lemma lagrangian_eq_kineticEnergy_sub_potentialEnergy (t : Time) (xₜ : Time → Space 1) :
    lagrangian S t (xₜ t) (∂ₜ xₜ t) = kineticEnergy S xₜ t - potentialEnergy S (xₜ t)
```

lemma contDiff_lagrangian [source]

```
@[fun_prop]
lemma contDiff_lagrangian (n : WithTop ℕ∞) : ContDiff ℝ n ↿S.lagrangian
```

lemma gradient_lagrangian_position_eq [source]

```
lemma gradient_lagrangian_position_eq (t : Time) (x : Space 1) (v : EuclideanSpace ℝ (Fin 1)) :
    gradient (fun x => lagrangian S t x v) x = - S.k • x
```

lemma gradient_lagrangian_velocity_eq [source]

```
lemma gradient_lagrangian_velocity_eq (t : Time) (x : Space 1) (v : EuclideanSpace ℝ (Fin 1)) :
    gradient (lagrangian S t x) v = S.m • v
```

def eulerLagrangeOp [source]

The Euler-Lagrange operator for the classical harmonic osscilator.

```
noncomputable def eulerLagrangeOp (xₜ : Time → Space 1) : Time → Space 1
```

lemma eulerLagrangeOp_eq [source]

```
lemma eulerLagrangeOp_eq (xₜ : Time → Space 1) :
    eulerLagrangeOp S xₜ = fun t => gradient (S.lagrangian t · (∂ₜ xₜ t)) (xₜ t)
    - ∂ₜ (fun t' => gradient (S.lagrangian t' (xₜ t') ·) (∂ₜ xₜ t')) t
```

lemma variational_gradient_action [source]

```
lemma variational_gradient_action (xₜ : Time → Space 1) (hq : ContDiff ℝ ∞ xₜ) :
    (δ (q':=xₜ), ∫ t, lagrangian S t (q' t) (fderiv ℝ q' t 1)) = eulerLagrangeOp S xₜ
```

lemma equationOfMotion_iff_eulerLagrangeOp [source]

```
lemma equationOfMotion_iff_eulerLagrangeOp (xₜ : Time → Space 1) :
    S.EquationOfMotion xₜ ↔ eulerLagrangeOp S xₜ = 0
```

lemma eulerLagrangeOp_eq_force [source]

The Euler lagrange operator corresponds to Newton's second law.

```
lemma eulerLagrangeOp_eq_force (xₜ : Time → Space 1) (hx : ContDiff ℝ ∞ xₜ) :
    eulerLagrangeOp S xₜ = fun t => force S (xₜ t) - S.m • ∂ₜ (∂ₜ xₜ) t
```

lemma equationOfMotion_iff_newtons_2nd_law [source]

```
lemma equationOfMotion_iff_newtons_2nd_law (xₜ : Time → Space 1) (hx : ContDiff ℝ ∞ xₜ) :
    S.EquationOfMotion xₜ ↔
    (∀ t, S.m • ∂ₜ (∂ₜ xₜ) t = force S (xₜ t))
```

lemma energy_conservation_of_equationOfMotion [source]

```
lemma energy_conservation_of_equationOfMotion (xₜ : Time → Space 1) (hx : ContDiff ℝ ∞ xₜ)
    (h : S.EquationOfMotion xₜ) : ∂ₜ (S.energy xₜ) = 0
```

lemma energy_conservation_of_equationOfMotion' [source]

```
lemma energy_conservation_of_equationOfMotion' (xₜ : Time → Space 1) (hx : ContDiff ℝ ∞ xₜ)
    (h : S.EquationOfMotion xₜ) (t : Time) : S.energy xₜ t = S.energy xₜ 0
```

def toCanonicalMomentum [source]

The equivalence between velocity and canonical momentum.

```
noncomputable def toCanonicalMomentum (t : Time) (x : Space 1) :
    EuclideanSpace ℝ (Fin 1) ≃ₗ[ℝ] EuclideanSpace ℝ (Fin 1) where
```

lemma toCanonicalMomentum_eq [source]

```
lemma toCanonicalMomentum_eq (t : Time) (x : Space 1) (v : EuclideanSpace ℝ (Fin 1)) :
    toCanonicalMomentum S t x v = S.m • v
```

def hamiltonian [source]

The hamiltonian as a function of time, momentum and position.

```
noncomputable def hamiltonian (t : Time) (p : EuclideanSpace ℝ (Fin 1)) (x : Space 1) : ℝ
```

lemma hamiltonian_eq [source]

```
lemma hamiltonian_eq :
    hamiltonian S = fun _ p x => (1 / (2 : ℝ)) * (1 / S.m) * ⟪p, p⟫_ℝ +
      (1 / (2 : ℝ)) * S.k * ⟪x, x⟫_ℝ
```

lemma hamiltonian_contDiff [source]

```
@[fun_prop]
lemma hamiltonian_contDiff (n : WithTop ℕ∞) : ContDiff ℝ n ↿S.hamiltonian
```

lemma gradient_hamiltonian_position_eq [source]

```
lemma gradient_hamiltonian_position_eq (t : Time) (x : Space 1) (p : EuclideanSpace ℝ (Fin 1))
    :
    gradient (hamiltonian S t p) x = S.k • x
```

lemma gradient_hamiltonian_momentum_eq [source]

```
lemma gradient_hamiltonian_momentum_eq (t : Time) (x : Space 1) (p : EuclideanSpace ℝ (Fin 1))
    :
    gradient (hamiltonian S t · x) p = (1 / S.m) • p
```

lemma hamiltonian_eq_energy [source]

```
lemma hamiltonian_eq_energy (xₜ : Time → Space 1) :
    (fun t => hamiltonian S t (toCanonicalMomentum S t (xₜ t) (∂ₜ xₜ t)) (xₜ t)) = energy S xₜ
```

def hamiltonEqOp [source]

The operator on the momentum-position phase-space whose vanishing is equivalent to the hamilton's equations between the momentum and position.

```
noncomputable def hamiltonEqOp (p : Time → EuclideanSpace ℝ (Fin 1)) (q : Time → Space 1)
```

lemma equationOfMotion_iff_hamiltonEqOp_eq_zero [source]

```
lemma equationOfMotion_iff_hamiltonEqOp_eq_zero (xₜ : Time → Space 1) (hx : ContDiff ℝ ∞ xₜ) :
    S.EquationOfMotion xₜ ↔
    hamiltonEqOp S (fun t => S.toCanonicalMomentum t (xₜ t) (∂ₜ xₜ t)) xₜ = 0
```

lemma equationOfMotion_tfae [source]

```
lemma equationOfMotion_tfae (xₜ : Time → Space 1) (hx : ContDiff ℝ ∞ xₜ) :
    List.TFAE [S.EquationOfMotion xₜ,
      (∀ t, S.m • ∂ₜ (∂ₜ xₜ) t = force S (xₜ t)),
      hamiltonEqOp S (fun t => S.toCanonicalMomentum t (xₜ t) (∂ₜ xₜ t)) xₜ = 0,
      (δ (q':=xₜ), ∫ t, lagrangian S t (q' t) (fderiv ℝ q' t 1)) = 0,
      (δ (pq':= fun t => (S.toCanonicalMomentum t (xₜ t) (∂ₜ xₜ t), xₜ t)),
        ∫ t, ⟪(pq' t).1, ∂ₜ (Prod.snd ∘ pq') t⟫_ℝ - S.hamiltonian t (pq' t).1 (pq' t).2) = 0]
```

2.2 PhysLean.ClassicalMechanics.HarmonicOscillator.Solution

PhysLean/ClassicalMechanics/HarmonicOscillator/Solution.lean on GitHub.

instance Zero InitialConditions [source]

The zero initial condition.

```
instance : Zero InitialConditions
```

lemma x₀_zero [source]

The zero initial condition has zero starting point.

```
@[simp]
lemma x₀_zero : x₀ 0 = 0
```

lemma v₀_zero [source]

The zero initial condition has zero starting velocity.

```
@[simp]
lemma v₀_zero : v₀ 0 = 0
```

def trajectory [source]

Given initial conditions, the solution to the classical harmonic oscillator.

```
noncomputable def trajectory (IC : InitialConditions) : Time → Space 1
```

lemma trajectory_eq [source]

```
lemma trajectory_eq (IC : InitialConditions) :
    IC.trajectory S = fun t : Time => cos (S.ω * t) • IC.x₀ + (sin (S.ω * t)/S.ω) • IC.v₀
```

lemma trajectory_zero [source]

For zero initial conditions, the trajectory is zero.

```
@[simp]
lemma trajectory_zero : trajectory S 0 = fun _ => 0
```

lemma trajectory_contDiff [source]

```
@[fun_prop]
lemma trajectory_contDiff (S : HarmonicOscillator) (IC : InitialConditions) {n : WithTop ℕ∞} :
    ContDiff ℝ n (IC.trajectory S)
```

lemma trajectory_velocity [source]

```
lemma trajectory_velocity (IC : InitialConditions) : ∂ₜ (IC.trajectory S) =
    fun t : Time => - S.ω • sin (S.ω * t.val) • IC.x₀ + cos (S.ω * t.val) • IC.v₀
```

lemma trajectory_acceleration [source]

```
lemma trajectory_acceleration (IC : InitialConditions) : ∂ₜ (∂ₜ (IC.trajectory S)) =
    fun t : Time => - S.ω^2 • cos (S.ω * t.val) • IC.x₀ - S.ω • sin (S.ω * t.val) • IC.v₀
```

lemma trajectory_position_at_zero [source]

For a set of initial conditions `IC` the position of the solution at time `0` is `IC.x₀`.

```
@[simp]
lemma trajectory_position_at_zero (IC : InitialConditions) : IC.trajectory S 0 = IC.x₀
```

lemma trajectory_velocity_at_zero [source]

```
@[simp]
lemma trajectory_velocity_at_zero (IC : InitialConditions) : ∂ₜ (IC.trajectory S) 0 = IC.v₀
```

lemma trajectory_equationOfMotion [source]

```
lemma trajectory_equationOfMotion (IC : InitialConditions) :
    EquationOfMotion S (IC.trajectory S)
```

lemma trajectories_unique [source]

The trajectories to the equation of motion for a given set of initial conditions are unique.

Semiformal implmentation: - One may needed the added condition of smoothness on `x` here. - `EquationOfMotion` needs defining before this can be proved.

```
@[sorryful]
lemma trajectories_unique (IC : InitialConditions) (x : Time → Space 1) :
    S.EquationOfMotion x ∧ x 0 = IC.x₀ ∧ ∂ₜ x 0 = IC.v₀ →
    x = IC.trajectory S
```

lemma trajectory_energy [source]

```
lemma trajectory_energy (IC : InitialConditions) : S.energy (IC.trajectory S) =
    fun _ => 1/2 * (S.m * ‖IC.v₀‖ ^2 + S.k * ‖IC.x₀‖ ^ 2)
```

lemma tan_time_eq_of_trajectory_velocity_eq_zero [source]

```
lemma tan_time_eq_of_trajectory_velocity_eq_zero (IC : InitialConditions) (t : Time)
    (h : ∂ₜ (IC.trajectory S) t = 0) (hx : IC.x₀ ≠ 0 ∨ IC.v₀ ≠ 0) :
    tan (S.ω * t) = IC.v₀ 0 / (S.ω * IC.x₀ 0)
```

lemma trajectory_velocity_eq_zero_at_arctan [source]

```
lemma trajectory_velocity_eq_zero_at_arctan (IC : InitialConditions) (hx : IC.x₀ ≠ 0) :
    (∂ₜ (IC.trajectory S)) (arctan (IC.v₀ 0 / (S.ω * IC.x₀ 0)) / S.ω) = 0
```

lemma trajectory_velocity_eq_zero_iff [source]

```
@[sorryful]
lemma trajectory_velocity_eq_zero_iff (IC : InitialConditions) (t : Time)
    (hx : IC.x₀ ≠ 0 ∨ IC.v₀ ≠ 0) :
    ∂ₜ (IC.trajectory S) t = 0 ↔
    ‖(IC.trajectory S) t‖ = √(‖IC.x₀‖^2 + (‖IC.v₀‖/S.ω)^2)
```

2.3 PhysLean.ClassicalMechanics.RigidBody.Basic

PhysLean/ClassicalMechanics/RigidBody/Basic.lean on GitHub.

structure RigidBody [source]

A Rigid body defined by its mass distribution.

```
structure RigidBody (d : ℕ) where
  ρ : C^⊤ ⟮𝓘(ℝ, Space d), Space d; 𝓘(ℝ, ℝ), ℝ⟯ →ₗ[ℝ] ℝ
```

def mass [source]

The total mass of the rigid body.

```
def mass {d : ℕ} (R : RigidBody d) : ℝ
```

def centerOfMass [source]

The center of mass of the rigid body.

```
noncomputable def centerOfMass {d : ℕ} (R : RigidBody d) : Space d
```

def inertiaTensor [source]

The inertia tensor of the rigid body.

```
noncomputable def inertiaTensor {d : ℕ} (R : RigidBody d) :
    Matrix (Fin d) (Fin d) ℝ
```

lemma inertiaTensor_symmetric [source]

```
lemma inertiaTensor_symmetric {d : ℕ} (R : RigidBody d) (i j : Fin d) :
    R.inertiaTensor i j = R.inertiaTensor j i
```

informal_definition kineticEnergy [source]

The kinetic energy of a rigid body.

```
informal_definition kineticEnergy where
  tag := "MEYBM"
  deps := [``RigidBody]
```

2.4 PhysLean.ClassicalMechanics.RigidBody.SolidSphere

PhysLean/ClassicalMechanics/RigidBody/SolidSphere.lean on GitHub.

def solidSphere [source]

The solid sphere as a rigid body.

```
noncomputable def solidSphere (d : ℕ) (m R : ℝ≥0) : RigidBody d where
```

lemma solidSphere_mass [source]

```
lemma solidSphere_mass {d : ℕ} (m R : ℝ≥0) (hr : R ≠ 0) : (solidSphere d.succ m R).mass = m
```

lemma solidSphere_centerOfMass [source]

The center of mass of a solid sphere located at the origin is `0`.

```
@[sorryful]
lemma solidSphere_centerOfMass {d : ℕ} (m R : ℝ≥0) (hr : R ≠ 0) :
    (solidSphere d.succ m R).centerOfMass = 0
```

lemma solidSphere_inertiaTensor [source]

The moment of inertia tensor of a solid sphere through its center of mass is `2/5 m R^2 * I`.

```
@[sorryful]
lemma solidSphere_inertiaTensor (m R : ℝ≥0) (hr : R ≠ 0) :
    (solidSphere 3 m R).inertiaTensor = (2/5 * m.1 * R.1^2) • (1 : Matrix _ _ _)
```

2.5 PhysLean.Electromagnetism.Electrostatics.ThreeDimension.FiniteCollection

PhysLean/Electromagnetism/Electrostatics/ThreeDimension/FiniteCollection.lean on GitHub.

def chargeDistribution [source]

The charge distribution for a finite collection of point particles.

```
def chargeDistribution {n : ℕ} (q : Fin n → ℝ) (r₀ : Fin n → Space) : ChargeDistribution 3
```

def electricPotential [source]

The electrostatic potential of a finite collection of point particles.

```
def electricPotential {n : ℕ} (q : Fin n → ℝ) (ε : ℝ) (r₀ : Fin n → Space) :
    StaticElectricPotential 3
```

def electricField [source]

The electric field of a finite collection of point particles.

```
def electricField {n : ℕ} (q : Fin n → ℝ) (ε : ℝ) (r₀ : Fin n → Space) : StaticElectricField 3
```

lemma electricField_eq_neg_gradD_electricPotential [source]

```
lemma electricField_eq_neg_gradD_electricPotential {n : ℕ} (q : Fin n → ℝ) (ε : ℝ)
    (r₀ : Fin n → Space) :
    electricField q ε r₀ = - Space.gradD (electricPotential q ε r₀)
```

lemma electricField_eq_ofPotential_electricPotential [source]

```
lemma electricField_eq_ofPotential_electricPotential {n : ℕ} (q : Fin n → ℝ) (ε : ℝ)
    (r₀ : Fin n → Space) :
    electricField q ε r₀ = ofPotential (electricPotential q ε r₀)
```

lemma gaussLaw [source]

Gauss's law for a finite collection of point particles.

```
lemma gaussLaw {n : ℕ} (q : Fin n → ℝ) (ε : ℝ)
    (r₀ : Fin n → Space) :
    (electricField q ε r₀).GaussLaw ε (chargeDistribution q r₀)
```

lemma faradaysLaw [source]

Faraday's law for a finite collection of point particles.

```
lemma faradaysLaw {n : ℕ} (q : Fin n → ℝ) (ε : ℝ)
    (r₀ : Fin n → Space) :
    (electricField q ε r₀).FaradaysLaw
```

2.6 PhysLean.Electromagnetism.Electrostatics.ThreeDimension.PointParticle

PhysLean/Electromagnetism/Electrostatics/ThreeDimension/PointParticle.lean on GitHub.

def electricPotential [source]

The electric potential of a point particle of charge `q` in 3d space sitting at the `r₀`. In physics notation this corresponds to the 'function' `(q/(4 * π * ε)) • ‖r - r₀‖⁻¹`. Here it is defined as the distribution corresponing to that function.

```
def electricPotential (q ε : ℝ) (r₀ : Space) : StaticElectricPotential 3
```

lemma chargeDistribution_eq_zero_of_charge_eq_zero [source]

```
lemma chargeDistribution_eq_zero_of_charge_eq_zero (r₀ : Space) :
    chargeDistribution 0 r₀ = 0
```

lemma electricPotential_eq_zero_of_charge_eq_zero [source]

```
lemma electricPotential_eq_zero_of_charge_eq_zero {ε : ℝ} (r₀ : Space) :
    electricPotential 0 ε r₀ = 0
```

lemma electricField_eq_zero_of_charge_eq_zero [source]

```
lemma electricField_eq_zero_of_charge_eq_zero {ε : ℝ} (r₀ : Space) :
    electricField 0 ε r₀ = 0
```

lemma chargeDistribution_eq_translateD [source]

```
lemma chargeDistribution_eq_translateD (q : ℝ) (r₀ : Space) :
    chargeDistribution q r₀ = Space.translateD r₀
      (chargeDistribution q 0)
```

lemma electricPotential_eq_translateD [source]

```
lemma electricPotential_eq_translateD (q ε : ℝ) (r₀ : Space) :
    electricPotential q ε r₀ = Space.translateD r₀ (electricPotential q ε 0)
```

lemma electricField_eq_translateD [source]

```
lemma electricField_eq_translateD (q ε : ℝ) (r₀ : Space) :
    electricField q ε r₀ = Space.translateD r₀ (electricField q ε 0)
```

lemma gradD_electricPotential_eq_electricField_of_integral_eq_zero [source]

The relation `E = -∇φ` holds for the point particle if the integral `∫ x, d_y η x * ‖x‖⁻¹ + η x * -⟪(‖x‖ ^ 3)⁻¹ • x, y⟫_ℝ = 0` is zero.

```
lemma gradD_electricPotential_eq_electricField_of_integral_eq_zero (q ε : ℝ)
    (h_integral : ∀ η : 𝒮(EuclideanSpace ℝ (Fin 3), ℝ), ∀ y : EuclideanSpace ℝ (Fin 3),
    ∫ (a : EuclideanSpace ℝ (Fin 3)), (fderivCLM ℝ η a y * ‖a‖⁻¹ +
    η a * - ⟪(‖a‖ ^ 3)⁻¹ • a, y⟫_ℝ) = 0) :
    - Space.gradD (electricPotential q ε 0) = electricField q ε 0
```

def potentialLimitSeries [source]

A series of functions whose limit is the `‖x‖⁻¹` and for which each function is differentiable everywhere.

```
def potentialLimitSeries : ℕ → EuclideanSpace ℝ (Fin 3) → ℝ
```

lemma potentialLimitSeries_eq [source]

```
lemma potentialLimitSeries_eq (n : ℕ) :
    potentialLimitSeries n = fun x => (‖x‖ ^ 2 + 1/(n + 1))^ (-1/2 : ℝ)
```

lemma potentialLimitSeries_tendsto [source]

```
lemma potentialLimitSeries_tendsto (x : EuclideanSpace ℝ (Fin 3)) (hx : x ≠ 0) :
    Filter.Tendsto (fun n => potentialLimitSeries n x) Filter.atTop (𝓝 (‖x‖⁻¹))
```

lemma potentialLimitSeries_differentiable [source]

```
lemma potentialLimitSeries_differentiable (n : ℕ) :
    Differentiable ℝ (potentialLimitSeries n)
```

lemma potentialLimitSeries_fderiv [source]

```
lemma potentialLimitSeries_fderiv (x y : EuclideanSpace ℝ (Fin 3)) (n : ℕ) :
    fderiv ℝ (potentialLimitSeries n) x y =
    - ((‖x‖ ^ 2 + (1 + (n : ℝ))⁻¹) ^ (- 1 /2 : ℝ)) ^ 3 * ⟪x, y⟫_ℝ
```

lemma potentialLimitSeries_fderiv_eq_potentialLimitseries_mul [source]

```
lemma potentialLimitSeries_fderiv_eq_potentialLimitseries_mul
    (x y : EuclideanSpace ℝ (Fin 3)) (n : ℕ) :
    fderiv ℝ (potentialLimitSeries n) x y = - (potentialLimitSeries n x) ^ 3 * ⟪x, y⟫_ℝ
```

lemma potentialLimitSeries_fderiv_tendsto [source]

```
lemma potentialLimitSeries_fderiv_tendsto (x y : EuclideanSpace ℝ (Fin 3)) (hx : x ≠ 0) :
    Filter.Tendsto (fun n => fderiv ℝ (potentialLimitSeries n) x y) Filter.atTop
      (𝓝 (-⟪(‖x‖ ^ 3)⁻¹ • x, y⟫_ℝ))
```

lemma potentialLimitSeries_aeStronglyMeasurable [source]

```
@[fun_prop]
lemma potentialLimitSeries_aeStronglyMeasurable (n : ℕ) :
    AEStronglyMeasurable (potentialLimitSeries n)
```

lemma potentialLimitSeries_fderiv_aeStronglyMeasurable [source]

```
@[fun_prop]
lemma potentialLimitSeries_fderiv_aeStronglyMeasurable (n : ℕ) (y : EuclideanSpace ℝ (Fin 3)) :
    AEStronglyMeasurable (fun x => fderiv ℝ (potentialLimitSeries n) x y)
```

lemma potentialLimitSeries_eq_sqrt_inv [source]

```
lemma potentialLimitSeries_eq_sqrt_inv (n : ℕ) :
    potentialLimitSeries n = fun x => √(‖x‖ ^ 2 + 1/(n + 1))⁻¹
```

lemma potentialLimitSeries_nonneg [source]

```
lemma potentialLimitSeries_nonneg (n : ℕ) (x : EuclideanSpace ℝ (Fin 3)) :
    0 ≤ potentialLimitSeries n x
```

lemma potentialLimitSeries_bounded_neq_zero [source]

```
lemma potentialLimitSeries_bounded_neq_zero (n : ℕ) (x : EuclideanSpace ℝ (Fin 3)) (hx : x ≠ 0)
     :
    ‖potentialLimitSeries n x‖ ≤ ‖x‖⁻¹
```

lemma potentialLimitSeries_bounded [source]

```
lemma potentialLimitSeries_bounded (n : ℕ) (x : EuclideanSpace ℝ (Fin 3)) :
    ‖potentialLimitSeries n x‖ ≤ ‖x‖⁻¹ + √(n + 1)
```

lemma potentialLimitSeries_isDistBounded [source]

```
lemma potentialLimitSeries_isDistBounded (n : ℕ) :
    IsDistBounded (potentialLimitSeries n)
```

lemma potentialLimitSeries_fderiv_bounded [source]

```
lemma potentialLimitSeries_fderiv_bounded (n : ℕ)
    (x y : EuclideanSpace ℝ (Fin 3)) :
    ‖fderiv ℝ (potentialLimitSeries n) x y‖ ≤ (‖x‖⁻¹) ^ 2 * ‖y‖
```

lemma potentialLimitSeries_fderiv_isDistBounded [source]

```
lemma potentialLimitSeries_fderiv_isDistBounded (n : ℕ) (y : EuclideanSpace ℝ (Fin 3)) :
    IsDistBounded (fun x => fderiv ℝ (potentialLimitSeries n) x y)
```

def potentialLimitSeriesFDerivSchwartz [source]

A series of functions of the form `fderiv ℝ (fun x => η x * potentialLimitSeries n x) x y`.

```
def potentialLimitSeriesFDerivSchwartz
    (y : EuclideanSpace ℝ (Fin 3)) (η : 𝓢(EuclideanSpace ℝ (Fin 3), ℝ)) (n : ℕ)
    (x : EuclideanSpace ℝ (Fin 3)) : ℝ
```

lemma potentialLimitSeriesFDerivSchwartz_eq [source]

```
lemma potentialLimitSeriesFDerivSchwartz_eq
    (y : EuclideanSpace ℝ (Fin 3)) (η : 𝓢(EuclideanSpace ℝ (Fin 3), ℝ)) (n : ℕ)
    (x : EuclideanSpace ℝ (Fin 3)) :
    potentialLimitSeriesFDerivSchwartz y η n x=
      fderiv ℝ η x y * potentialLimitSeries n x + η x * fderiv ℝ (potentialLimitSeries n) x y
```

lemma potentialLimitSeriesFDerivSchwartz_tendsto [source]

```
lemma potentialLimitSeriesFDerivSchwartz_tendsto
    (y : EuclideanSpace ℝ (Fin 3)) (η : 𝓢(EuclideanSpace ℝ (Fin 3), ℝ)) :
    ∀ᵐ (a : EuclideanSpace ℝ (Fin 3)) ∂(volume),
    Filter.Tendsto (fun n => potentialLimitSeriesFDerivSchwartz y η n a)
      Filter.atTop (𝓝 (fderiv ℝ η a y * ‖a‖⁻¹ + η a * -⟪(‖a‖ ^ 3)⁻¹ • a, y⟫_ℝ))
```

lemma potentialLimitSeriesFDerivSchwartz_aeStronglyMeasurable [source]

```
lemma potentialLimitSeriesFDerivSchwartz_aeStronglyMeasurable
    (y : EuclideanSpace ℝ (Fin 3)) (η : 𝓢(EuclideanSpace ℝ (Fin 3), ℝ)) (n : ℕ) :
    AEStronglyMeasurable (fun x => potentialLimitSeriesFDerivSchwartz y η n x)
```

lemma potentialLimitSeriesFDerivSchwartz_integral_tendsto_eq_integral [source]

```
lemma potentialLimitSeriesFDerivSchwartz_integral_tendsto_eq_integral
    (y : EuclideanSpace ℝ (Fin 3)) (η : 𝓢(EuclideanSpace ℝ (Fin 3), ℝ)) :
    Filter.Tendsto (fun n => ∫ (x : EuclideanSpace ℝ (Fin 3)),
      potentialLimitSeriesFDerivSchwartz y η n x) Filter.atTop
      (𝓝 (∫ (x : EuclideanSpace ℝ (Fin 3)), fderiv ℝ η x y * ‖x‖⁻¹ +
        η x * -⟪(‖x‖ ^ 3)⁻¹ • x, y⟫_ℝ))
```

lemma potentialLimitSeriesFDerivSchwartz_integral_eq_zero [source]

```
lemma potentialLimitSeriesFDerivSchwartz_integral_eq_zero
    (y : EuclideanSpace ℝ (Fin 3)) (η : 𝓢(EuclideanSpace ℝ (Fin 3), ℝ)) (n : ℕ) :
    ∫ (x : EuclideanSpace ℝ (Fin 3)), potentialLimitSeriesFDerivSchwartz y η n x = 0
```

lemma potentialLimitSeriesFDerivSchwartz_integral_tendsto_eq_zero [source]

```
lemma potentialLimitSeriesFDerivSchwartz_integral_tendsto_eq_zero
    (y : EuclideanSpace ℝ (Fin 3)) (η : 𝓢(EuclideanSpace ℝ (Fin 3), ℝ)) :
    Filter.Tendsto (fun n => ∫ (x : EuclideanSpace ℝ (Fin 3)),
      potentialLimitSeriesFDerivSchwartz y η n x) Filter.atTop (𝓝 (0))
```

lemma electricField_eq_neg_gradD_electricPotential_origin [source]

```
lemma electricField_eq_neg_gradD_electricPotential_origin (q ε : ℝ) :
    electricField q ε 0 = - Space.gradD (electricPotential q ε 0)
```

lemma electricField_eq_neg_gradD_electricPotential [source]

```
lemma electricField_eq_neg_gradD_electricPotential (q ε : ℝ) (r₀ : EuclideanSpace ℝ (Fin 3)) :
    electricField q ε r₀ = - Space.gradD (electricPotential q ε r₀)
```

lemma electricField_eq_ofPotential_electricPotential [source]

```
lemma electricField_eq_ofPotential_electricPotential (q ε : ℝ) (r₀ : EuclideanSpace ℝ (Fin 3))
    :
    electricField q ε r₀ = ofPotential (electricPotential q ε r₀)
```

lemma faradaysLaw [source]

```
lemma faradaysLaw (q ε : ℝ) (r₀ : Space) : (electricField q ε r₀).FaradaysLaw
```

lemma gaussLaw_origin [source]

Guass' law for a point particle in 3-dimensions at the origin, that is this theorem states that the divergence of `(q/(4 * π * ε)) • ‖r‖⁻¹ ^ 3 • r` is equal to `q • δ(r)`.

```
lemma gaussLaw_origin (q ε : ℝ) : (electricField q ε 0).GaussLaw ε (chargeDistribution q 0)
```

informal_lemma electricField_rotationally_invariant [source]

The electrostatic field of a point particle is rotationally invariant.

```
informal_lemma electricField_rotationally_invariant where
  deps := [``electricField]
  tag := "L7NXF"
```

2.7 PhysLean.Electromagnetism.KineticTerm

PhysLean/Electromagnetism/KineticTerm.lean on GitHub.

def kineticTerm [source]

The kinetic energy from an electromagnetic potential.

```
noncomputable def kineticTerm (A : ElectromagneticPotential) : SpaceTime → ℝ
```

lemma kineticTerm_equivariant [source]

```
lemma kineticTerm_equivariant (A : ElectromagneticPotential) (Λ : LorentzGroup 3)
    (hf : Differentiable ℝ A) (x : SpaceTime) :
    kineticTerm (fun x => Λ • A (Λ⁻¹ • x)) x = kineticTerm A (Λ⁻¹ • x)
```

lemma kineticTerm_eq_sum [source]

```
lemma kineticTerm_eq_sum (A : ElectromagneticPotential) (x : SpaceTime) :
    A.kineticTerm x =
    - 1/4 * ∑ μ, ∑ ν, ∑ μ', ∑ ν', η μ μ' * η ν ν' *
      (Lorentz.CoVector.basis.tensorProduct Lorentz.Vector.basis).repr (A.toFieldStrength x) (
    μ, ν)
      * (Lorentz.CoVector.basis.tensorProduct Lorentz.Vector.basis).repr
        (A.toFieldStrength x) (μ', ν')
```

lemma kineticTerm_eq_sum_potential [source]

```
lemma kineticTerm_eq_sum_potential (A : ElectromagneticPotential) (x : SpaceTime) :
    A.kineticTerm x = - 1 / 2 * ∑ μ, ∑ ν,
        (η μ μ * η ν ν * (∂_ μ A x ν) ^ 2 - ∂_ μ A x ν * ∂_ ν A x μ)
```

lemma kineticTerm_eq_electric_magnetic [source]

```
lemma kineticTerm_eq_electric_magnetic (A : ElectromagneticPotential) (t : Time)
    (x : Space) (hA : Differentiable ℝ A) :
    A.kineticTerm (SpaceTime.toTimeAndSpace.symm (t, x)) =
    1/2 * (‖A.electricField t x‖ ^ 2 - ‖A.magneticField t x‖ ^ 2)
```

instance InnerProductSpace ℝ SpaceTime [source]

A local instance of an inner product structure on `SpaceTime`.

```
noncomputable local instance : InnerProductSpace ℝ SpaceTime
```

def gradKineticTerm [source]

The variational gradient of the kinetic term of an electromagnetic potential.

```
noncomputable def gradKineticTerm (A : ElectromagneticPotential) : SpaceTime → Lorentz.Vector
```

lemma gradKineticTerm_eq_sum_fderiv [source]

```
lemma gradKineticTerm_eq_sum_fderiv (A : ElectromagneticPotential)
    (hA : ContDiff ℝ ∞ A) :
    let F' : (Fin 1 ⊕ Fin 3) × (Fin 1 ⊕ Fin 3) → (SpaceTime → ℝ) →
    SpaceTime → Lorentz.Vector
```

lemma gradKineticTerm_eq_sum_sum [source]

```
lemma gradKineticTerm_eq_sum_sum (A : ElectromagneticPotential) (x : SpaceTime)
    (ha : ContDiff ℝ ∞ A) :
    A.gradKineticTerm x = ∑ (ν : (Fin 1 ⊕ Fin 3)), ∑ (μ : (Fin 1 ⊕ Fin 3)),
        (η μ μ * η ν ν * ∂_ μ (fun x' => ∂_ μ A x' ν) x -
        ∂_ μ (fun x' => ∂_ ν A x' μ) x) • Lorentz.Vector.basis ν
```

lemma gradKineticTerm_eq_fieldStrength [source]

```
lemma gradKineticTerm_eq_fieldStrength (A : ElectromagneticPotential)
    (x : SpaceTime) (ha : ContDiff ℝ ∞ A) :
    A.gradKineticTerm x = ∑ (ν : (Fin 1 ⊕ Fin 3)), η ν ν •
    (∑ (μ : (Fin 1 ⊕ Fin 3)), (∂_ μ (A.fieldStrengthMatrix · (μ, ν)) x))
    • Lorentz.Vector.basis ν
```

lemma gradKineticTerm_eq_electric_magnetic [source]

```
lemma gradKineticTerm_eq_electric_magnetic (A : ElectromagneticPotential)
    (x : SpaceTime) (ha : ContDiff ℝ ∞ A) :
    A.gradKineticTerm x =
    Space.div (A.electricField (toTimeAndSpace x).1) (toTimeAndSpace x).2 •
      Lorentz.Vector.basis (Sum.inl 0) +
    ∑ i, (∂ₜ (fun t => A.electricField t (toTimeAndSpace x).2) (toTimeAndSpace x).1 i-
          Space.curl (A.magneticField (toTimeAndSpace x).1) (toTimeAndSpace x).2 i) •
          Lorentz.Vector.basis (Sum.inr i)
```

2.8 PhysLean.Electromagnetism.LorentzAction

PhysLean/Electromagnetism/LorentzAction.lean on GitHub.

lemma electricField_apply_x_boost [source]

```
lemma electricField_apply_x_boost (β : ℝ) (hβ : |β| < 1)
    (A : ElectromagneticPotential) (hA : Differentiable ℝ A) (t : Time) (x : Space) :
    let Λ
```

lemma magneticField_apply_x_boost [source]

```
lemma magneticField_apply_x_boost (β : ℝ) (hβ : |β| < 1)
    (A : ElectromagneticPotential) (hA : Differentiable ℝ A) (t : Time) (x : Space) :
    let Λ
```

2.9 PhysLean.Electromagnetism.Potential

PhysLean/Electromagnetism/Potential.lean on GitHub.

abbrev ElectromagneticPotential [source]

The electricomagnetic potential is a tensor `A^μ`.

```
noncomputable abbrev ElectromagneticPotential
```

lemma spaceTime_deriv_action_eq_sum [source]

```
lemma spaceTime_deriv_action_eq_sum {μ ν : Fin 1 ⊕ Fin 3}
    (Λ : LorentzGroup 3) (A : ElectromagneticPotential) (hA : Differentiable ℝ A) :
    ∂_ μ (fun x => Λ • A (Λ⁻¹ • x)) x ν =
    ∑ κ, ∑ ρ, (Λ.1 ν κ * Λ⁻¹.1 ρ μ) * ∂_ ρ A (Λ⁻¹ • x) κ
```

lemma differentiable_component [source]

```
lemma differentiable_component
    (A : ElectromagneticPotential) (hA : Differentiable ℝ A) (μ : Fin 1 ⊕ Fin 3) :
    Differentiable ℝ (fun x => A x μ)
```

instance InnerProductSpace ℝ SpaceTime [source]

A local instance of the inner product structure on `SpaceTime`.

```
noncomputable local instance : InnerProductSpace ℝ SpaceTime
```

lemma hasVarAdjDerivAt_component [source]

```
lemma hasVarAdjDerivAt_component (μ : Fin 1 ⊕ Fin 3) (A : SpaceTime → Lorentz.Vector)
    (hA : ContDiff ℝ ∞ A) :
        HasVarAdjDerivAt (fun (A' : SpaceTime → Lorentz.Vector) x => A' x μ)
          (fun (A' : SpaceTime → ℝ) x => A' x • Lorentz.Vector.basis μ) A
```

lemma deriv_hasVarAdjDerivAt [source]

```
lemma deriv_hasVarAdjDerivAt (μ ν : Fin 1 ⊕ Fin 3) (A : SpaceTime → Lorentz.Vector)
    (hA : ContDiff ℝ ∞ A) :
    HasVarAdjDerivAt (fun (A : SpaceTime → Lorentz.Vector) x => ∂_ μ A x ν)
      (fun ψ x => - (fderiv ℝ ψ x) (Lorentz.Vector.basis μ) • Lorentz.Vector.basis ν) A
```

def deriv [source]

The derivative of the electric potential, `∂_μ A^ν`.

```
noncomputable def deriv (A : ElectromagneticPotential) :
    SpaceTime → Lorentz.CoVector ⊗[ℝ] Lorentz.Vector
```

lemma deriv_equivariant [source]

```
lemma deriv_equivariant {x : SpaceTime} (A : ElectromagneticPotential) (Λ : LorentzGroup 3)
    (hf : Differentiable ℝ A) : deriv (fun x => Λ • A (Λ⁻¹ • x)) x = Λ • (deriv A (Λ⁻¹ • x))
```

lemma deriv_basis_repr_apply [source]

```
@[simp]
lemma deriv_basis_repr_apply {μν : (Fin 1 ⊕ Fin 3) × (Fin 1 ⊕ Fin 3)} (A :
    ElectromagneticPotential)
    (x : SpaceTime) :
    (Lorentz.CoVector.basis.tensorProduct Lorentz.Vector.basis).repr (deriv A x) μν =
    ∂_ μν.1 A x μν.2
```

lemma toTensor_deriv_basis_repr_apply [source]

```
lemma toTensor_deriv_basis_repr_apply (A : ElectromagneticPotential)
    (x : SpaceTime) (b : ComponentIdx (S := realLorentzTensor)
      (Sum.elim ![Color.down] ![Color.up] ∘ ⇑finSumFinEquiv.symm)) :
    (Tensor.basis _).repr (Tensorial.toTensor (deriv A x)) b =
    ∂_ (finSumFinEquiv.symm (b 0)) A x (finSumFinEquiv.symm (b 1))
```

def toFieldStrength [source]

The field strength from an electromagnetic potential, as a tensor `F_μ^ν`.

```
noncomputable def toFieldStrength (A : ElectromagneticPotential) :
    SpaceTime → Lorentz.Vector ⊗[ℝ] Lorentz.Vector
```

lemma toFieldStrength_eq_add [source]

```
lemma toFieldStrength_eq_add (A : ElectromagneticPotential) (x : SpaceTime) :
    toFieldStrength A x =
    Tensorial.toTensor.symm (permT id (PermCond.auto) {(η 3 | μ μ' ⊗ A.deriv x | μ' ν)}ᵀ)
    - Tensorial.toTensor.symm (permT ![1, 0] (PermCond.auto)
      {(η 3 | μ μ' ⊗ A.deriv x | μ' ν)}ᵀ)
```

lemma toTensor_toFieldStrength [source]

```
lemma toTensor_toFieldStrength (A : ElectromagneticPotential) (x : SpaceTime) :
    Tensorial.toTensor (toFieldStrength A x) =
    (permT id (PermCond.auto) {(η 3 | μ μ' ⊗ A.deriv x | μ' ν)}ᵀ)
    - (permT ![1, 0] (PermCond.auto) {(η 3 | μ μ' ⊗ A.deriv x | μ' ν)}ᵀ)
```

lemma toTensor_toFieldStrength_basis_repr [source]

```
lemma toTensor_toFieldStrength_basis_repr (A : ElectromagneticPotential) (x : SpaceTime)
    (b : ComponentIdx (S := realLorentzTensor)
      (Sum.elim ![Color.up] ![Color.up] ∘ ⇑finSumFinEquiv.symm)) :
    (Tensor.basis _).repr (Tensorial.toTensor (toFieldStrength A x)) b =
    ∑ κ, (η (finSumFinEquiv.symm (b 0)) κ * ∂_ κ A x (finSumFinEquiv.symm (b 1)) -
      η (finSumFinEquiv.symm (b 1)) κ * ∂_ κ A x (finSumFinEquiv.symm (b 0)))
```

lemma toFieldStrength_tensor_basis_eq_basis [source]

```
lemma toFieldStrength_tensor_basis_eq_basis (A : ElectromagneticPotential) (x : SpaceTime)
    (b : ComponentIdx (S := realLorentzTensor)
      (Sum.elim ![Color.up] ![Color.up] ∘ ⇑finSumFinEquiv.symm)) :
    (Tensor.basis _).repr (Tensorial.toTensor (toFieldStrength A x)) b =
```

```
    (Lorentz.Vector.basis.tensorProduct Lorentz.Vector.basis).repr (toFieldStrength A x)
      (finSumFinEquiv.symm (b 0), finSumFinEquiv.symm (b 1))
```

lemma toFieldStrength_basis_repr_apply [source]

```
lemma toFieldStrength_basis_repr_apply {μν : (Fin 1 ⊕ Fin 3) × (Fin 1 ⊕ Fin 3)}
    (A : ElectromagneticPotential) (x : SpaceTime) :
    (Lorentz.CoVector.basis.tensorProduct Lorentz.Vector.basis).repr (A.toFieldStrength x) μν =
    ∑ κ, ((η μν.1 κ * ∂_ κ A x μν.2) - η μν.2 κ * ∂_ κ A x μν.1)
```

lemma toFieldStrength_basis_repr_apply_eq_single [source]

```
lemma toFieldStrength_basis_repr_apply_eq_single {μν : (Fin 1 ⊕ Fin 3) × (Fin 1 ⊕ Fin 3)}
    (A : ElectromagneticPotential) (x : SpaceTime) :
    (Lorentz.CoVector.basis.tensorProduct Lorentz.Vector.basis).repr (A.toFieldStrength x) μν =
    ((η μν.1 μν.1 * ∂_ μν.1 A x μν.2) - η μν.2 μν.2 * ∂_ μν.2 A x μν.1)
```

abbrev fieldStrengthMatrix [source]

The matrix corresponding to the field strength in the standard basis.

```
noncomputable abbrev fieldStrengthMatrix (A : ElectromagneticPotential) (x : SpaceTime)
```

lemma fieldStrengthMatrix_eq_tensor_basis_repr [source]

```
lemma fieldStrengthMatrix_eq_tensor_basis_repr (A : ElectromagneticPotential) (x : SpaceTime)
    (μ ν : (Fin 1 ⊕ Fin 3)) :
    A.fieldStrengthMatrix x (μ, ν) =
    (Tensor.basis _).repr (Tensorial.toTensor (toFieldStrength A x))
    (fun | 0 => finSumFinEquiv μ | 1 => finSumFinEquiv ν)
```

lemma fieldStrengthMatrix_differentiable [source]

```
lemma fieldStrengthMatrix_differentiable {A : ElectromagneticPotential}
    {μν} (hA : ContDiff ℝ 2 A) :
    Differentiable ℝ (A.fieldStrengthMatrix · μν)
```

lemma toFieldStrength_antisymmetric [source]

```
lemma toFieldStrength_antisymmetric (A : ElectromagneticPotential) (x : SpaceTime) :
    {A.toFieldStrength x | μ ν = - (A.toFieldStrength x | ν μ)}^T
```

lemma fieldStrengthMatrix_antisymm [source]

```
lemma fieldStrengthMatrix_antisymm (A : ElectromagneticPotential) (x : SpaceTime)
    (μ ν : Fin 1 ⊕ Fin 3) :
    A.fieldStrengthMatrix x (μ, ν) = - A.fieldStrengthMatrix x (ν, μ)
```

lemma fieldStrengthMatrix_diag_eq_zero [source]

```
@[simp]
lemma fieldStrengthMatrix_diag_eq_zero (A : ElectromagneticPotential) (x : SpaceTime)
```

```
    (μ : Fin 1 ⊕ Fin 3) :
    A.fieldStrengthMatrix x (μ, μ) = 0
```

lemma toFieldStrength_equivariant [source]

```
lemma toFieldStrength_equivariant (A : ElectromagneticPotential) (Λ : LorentzGroup 3)
    (hf : Differentiable ℝ A) (x : SpaceTime) :
    toFieldStrength (fun x => Λ • A (Λ⁻¹ • x)) x =
      Λ • toFieldStrength A (Λ⁻¹ • x)
```

lemma fieldStrengthMatrix_equivariant [source]

```
lemma fieldStrengthMatrix_equivariant (A : ElectromagneticPotential)
    (Λ : LorentzGroup 3) (hf : Differentiable ℝ A) (x : SpaceTime)
    (μ : (Fin 1 ⊕ Fin 3)) (ν : Fin 1 ⊕ Fin 3) :
    fieldStrengthMatrix (fun x => Λ • A (Λ⁻¹ • x)) x (μ, ν) =
    ∑ κ, ∑ ρ, (Λ.1 μ κ * Λ.1 ν ρ) * A.fieldStrengthMatrix (Λ⁻¹ • x) (κ, ρ)
```

def scalarPotential [source]

The scalar potential from the electromagnetic potential.

```
noncomputable def scalarPotential (A : ElectromagneticPotential) : Time → Space → ℝ
```

def vectorPotential [source]

The vector potential from the electromagnetic potential.

```
noncomputable def vectorPotential (A : ElectromagneticPotential) :
    Time → Space → EuclideanSpace ℝ (Fin 3)
```

def electricField [source]

The electric field from the electromagnetic potential.

```
noncomputable def electricField (A : ElectromagneticPotential) : ElectricField
```

lemma electricField_eq_fieldStrengthMatrix [source]

```
lemma electricField_eq_fieldStrengthMatrix (A : ElectromagneticPotential) (t : Time)
    (x : Space) (i : Fin 3) (hA : Differentiable ℝ A) :
    A.electricField t x i = -
    A.fieldStrengthMatrix (SpaceTime.toTimeAndSpace.symm (t, x)) (Sum.inl 0, Sum.inr i)
```

lemma electricField_differentiable [source]

```
lemma electricField_differentiable {A : ElectromagneticPotential}
    (hA : ContDiff ℝ 2 A) : Differentiable ℝ (₁A.electricField)
```

def magneticField [source]

The magnetic field from the electromagnetic potential.

```
noncomputable def magneticField (A : ElectromagneticPotential) : MagneticField
```

lemma magneticField_fst_eq_fieldStrengthMatrix [source]

```
lemma magneticField_fst_eq_fieldStrengthMatrix (A : ElectromagneticPotential) (t : Time)
    (x : Space) (hA : Differentiable ℝ A) :
    A.magneticField t x 0 =
    - A.fieldStrengthMatrix (SpaceTime.toTimeAndSpace.symm (t, x)) (Sum.inr 1, Sum.inr 2)
```

lemma magneticField_snd_eq_fieldStrengthMatrix [source]

```
lemma magneticField_snd_eq_fieldStrengthMatrix (A : ElectromagneticPotential) (t : Time)
    (x : Space) (hA : Differentiable ℝ A) :
    A.magneticField t x 1 = A.fieldStrengthMatrix (SpaceTime.toTimeAndSpace.symm (t, x))
      (Sum.inr 0, Sum.inr 2)
```

lemma magneticField_thd_eq_fieldStrengthMatrix [source]

```
lemma magneticField_thd_eq_fieldStrengthMatrix (A : ElectromagneticPotential) (t : Time)
    (x : Space) (hA : Differentiable ℝ A) :
    A.magneticField t x 2 =
    - A.fieldStrengthMatrix (SpaceTime.toTimeAndSpace.symm (t, x)) (Sum.inr 0, Sum.inr 1)
```

lemma fieldStrengthMatrix_eq_electric_magnetic [source]

```
lemma fieldStrengthMatrix_eq_electric_magnetic (A : ElectromagneticPotential) (t : Time)
    (x : Space) (hA : Differentiable ℝ A) (μ ν : Fin 1 ⊕ Fin 3) :
    A.fieldStrengthMatrix (SpaceTime.toTimeAndSpace.symm (t, x)) (μ, ν) =
    match μ, ν with
    | Sum.inl 0, Sum.inl 0 => 0
    | Sum.inl 0, Sum.inr i => - A.electricField t x i
    | Sum.inr i, Sum.inl 0 => A.electricField t x i
    | Sum.inr i, Sum.inr j =>
    match i, j with
    | 0, 0 => 0
    | 0, 1 => - A.magneticField t x 2
    | 0, 2 => A.magneticField t x 1
    | 1, 0 => A.magneticField t x 2
    | 1, 1 => 0
    | 1, 2 => - A.magneticField t x 0
    | 2, 0 => - A.magneticField t x 1
    | 2, 1 => A.magneticField t x 0
    | 2, 2 => 0
```

lemma fieldStrengthMatrix_eq_electric_magnetic_of_spaceTime [source]

```
lemma fieldStrengthMatrix_eq_electric_magnetic_of_spaceTime (A : ElectromagneticPotential)
    (x : SpaceTime) (hA : Differentiable ℝ A) (μ ν : Fin 1 ⊕ Fin 3) :
    let tx
```

2.10 PhysLean.Mathematics.Calculus.Divergence

PhysLean/Mathematics/Calculus/Divergence.lean on GitHub.

lemma toMatrix_smulRight [source]

```
lemma LinearMap.toMatrix_smulRight {R M M₁ m n : Type*} [CommSemiring R] [AddCommMonoid M]
    [AddCommMonoid M₁] [Module R M] [Module R M₁] [Finite m] [Fintype n] [DecidableEq n]
    (f : M₁ →ₗ[R] R) (x : M) (v₁ : Module.Basis n R M₁) (v₂ : Module.Basis m R M) :
    toMatrix v₁ v₂ (f.smulRight x) = Matrix.vecMulVec (v₂.repr x) (⇑f ∘ ⇑v₁)
```

theorem trace_vecMulVec [source]

```
@[simp]
theorem Matrix.trace_vecMulVec {R n : Type*} [Fintype n] [NonUnitalNonAssocSemiring R]
    (a b : n → R) : trace (vecMulVec a b) = a ⬝ᵥ b
```

lemma trace_smulRight [source]

```
@[simp]
lemma LinearMap.trace_smulRight {R M : Type*} [CommSemiring R] [AddCommMonoid M]
    [Module R M] [Module.Free R M] [Module.Finite R M] (f : M →ₗ[R] R) (x : M) :
    trace R M (f.smulRight x) = f x
```

lemma smulRight_toLinearMap [source]

```
@[simp]
lemma ContinuousLinearMap.smulRight_toLinearMap {M₁ : Type*} [TopologicalSpace M₁]
    [AddCommMonoid M₁] {M₂ : Type*} [TopologicalSpace M₂] [AddCommMonoid M₂] {R : Type*} {S :
    Type*}
    [Semiring R] [Semiring S] [Module R M₁] [Module R M₂] [Module R S] [Module S M₂]
    [IsScalarTower R S M₂] [TopologicalSpace S] [ContinuousSMul S M₂] (c : M₁ →L[R] S) (f : M₂)
     :
    (↑(ContinuousLinearMap.smulRight c f) : M₁ →ₗ[R] M₂) =
      LinearMap.smulRight (↑c : M₁ →ₗ[R] S) f
```

2.11 PhysLean.Mathematics.Distribution.Function.IsDistBounded

PhysLean/Mathematics/Distribution/Function/IsDistBounded.lean on GitHub.

def IsDistBounded [source]

The boundedness condition on a function `EuclideanSpace ℝ (Fin dm1.succ) → F` for it to form a distribution.

```
@[fun_prop]
def IsDistBounded {dm1 : ℕ} (f : EuclideanSpace ℝ (Fin dm1.succ) → F) : Prop
```

lemma add [source]

```
@[fun_prop]
lemma IsDistBounded.add {dm1 : ℕ} {f g : EuclideanSpace ℝ (Fin dm1.succ) → F}
    (hf : IsDistBounded f) (hg : IsDistBounded g) : IsDistBounded (f + g)
```

lemma const_smul [source]

```
@[fun_prop]
```

```
lemma IsDistBounded.const_smul {dm1 : ℕ} [NormedSpace ℝ F] {f : EuclideanSpace ℝ (Fin dm1.succ)
     → F}
    (hf : IsDistBounded f) (c : ℝ) : IsDistBounded (c • f)
```

lemma pi_comp [source]

```
lemma IsDistBounded.pi_comp {dm1 n : ℕ}
    {f : EuclideanSpace ℝ (Fin dm1.succ) → EuclideanSpace ℝ (Fin n)}
    (hf : IsDistBounded f) (j : Fin n) : IsDistBounded (fun x => f x j)
```

lemma comp_add_right [source]

```
lemma IsDistBounded.comp_add_right {dm1 : ℕ} {f : EuclideanSpace ℝ (Fin dm1.succ) → F}
    (hf : IsDistBounded f) (c : EuclideanSpace ℝ (Fin dm1.succ)) :
    IsDistBounded (fun x => f (x + c))
```

lemma const_mul_fun [source]

```
@[fun_prop]
lemma IsDistBounded.const_mul_fun {dm1 : ℕ}
    {f : EuclideanSpace ℝ (Fin dm1.succ) → ℝ}
    (hf : IsDistBounded f) (c : ℝ) : IsDistBounded (fun x => c * f x)
```

lemma congr [source]

```
lemma IsDistBounded.congr {dm1 : ℕ} {f : EuclideanSpace ℝ (Fin dm1.succ) → F}
    {g : EuclideanSpace ℝ (Fin dm1.succ) → F'}
    (hf : IsDistBounded f) (hfg : ∀ x, ‖g x‖ = ‖f x‖) : IsDistBounded g
```

lemma mono [source]

```
lemma IsDistBounded.mono {dm1 : ℕ} {f : EuclideanSpace ℝ (Fin dm1.succ) → F}
    {g : EuclideanSpace ℝ (Fin dm1.succ) → F'}
    (hf : IsDistBounded f) (hfg : ∀ x, ‖g x‖ ≤ ‖f x‖) : IsDistBounded g
```

lemma inner_left [source]

```
@[fun_prop]
lemma IsDistBounded.inner_left {dm1 n : ℕ}
    {f : EuclideanSpace ℝ (Fin dm1.succ) → EuclideanSpace ℝ (Fin n) }
    (hf : IsDistBounded f) (y : EuclideanSpace ℝ (Fin n)) :
    IsDistBounded (fun x => ⟪f x, y⟫_ℝ)
```

lemma const [source]

```
@[fun_prop]
lemma IsDistBounded.const {dm1 : ℕ} (f : F) :
    IsDistBounded (dm1 := dm1) (fun _ : EuclideanSpace ℝ (Fin dm1.succ) => f)
```

lemma pow [source]

```
@[fun_prop]
lemma IsDistBounded.pow {dm1 : ℕ} (n : ℤ) (hn : - dm1 ≤ n) :
    IsDistBounded (dm1 := dm1) (fun x => ‖x‖ ^ n)
```

lemma pow_shift [source]

```
@[fun_prop]
lemma IsDistBounded.pow_shift {dm1 : ℕ} (n : ℤ)
    (g : EuclideanSpace ℝ (Fin dm1.succ)) (hn : - dm1 ≤ n) :
    IsDistBounded (dm1 := dm1) (fun x => ‖x - g‖ ^ n)
```

lemma inv [source]

```
@[fun_prop]
lemma IsDistBounded.inv {n : ℕ} :
    IsDistBounded (dm1 := n.succ) (fun x => ‖x‖⁻¹)
```

lemma schwartzMap_mul_pow_integrable [source]

```
private lemma schwartzMap_mul_pow_integrable {dm1 : ℕ} (η : 𝓢(EuclideanSpace ℝ (Fin dm1.succ),
    ℝ))
    (p : ℤ) (hp : - (dm1 : ℤ) ≤ p) :
    Integrable (fun x => ‖η x‖ * ‖x‖ ^ p) volume
```

lemma schwartzMap_mul_integrable_norm [source]

```
lemma IsDistBounded.schwartzMap_mul_integrable_norm {dm1 : ℕ}
    {η : 𝓢(EuclideanSpace ℝ (Fin dm1.succ), ℝ)}
    {f : EuclideanSpace ℝ (Fin dm1.succ) → F}
    (hf : IsDistBounded f)
    (hae: AEStronglyMeasurable (fun x => f x) volume) :
    Integrable (fun x => ‖η x‖ * ‖f x‖) volume
```

lemma schwartzMap_smul_integrable [source]

```
lemma IsDistBounded.schwartzMap_smul_integrable {dm1 : ℕ}
    {η : 𝓢(EuclideanSpace ℝ (Fin dm1.succ), ℝ)}
    {f : EuclideanSpace ℝ (Fin dm1.succ) → F}
    (hf : IsDistBounded f) [NormedSpace ℝ F]
    (hae: AEStronglyMeasurable (fun x => f x) volume) :
    Integrable (fun x => η x • f x) volume
```

lemma schwartzMap_mul_integrable [source]

```
@[fun_prop]
lemma IsDistBounded.schwartzMap_mul_integrable {dm1 : ℕ} (f : EuclideanSpace ℝ (Fin dm1.succ)
    → ℝ)
    (hf : IsDistBounded f)
    (hae: AEStronglyMeasurable (fun x => f x) volume)
    (η : 𝓢(EuclideanSpace ℝ (Fin dm1.succ), ℝ)) :
    Integrable (fun x : EuclideanSpace ℝ (Fin dm1.succ) => η x * f x)
```

lemma integrable_fderiv_schwartzMap_mul [source]

```
@[fun_prop]
lemma IsDistBounded.integrable_fderiv_schwartzMap_mul {dm1 : ℕ}
    (f : EuclideanSpace ℝ (Fin dm1.succ) → ℝ) (hf : IsDistBounded f)
    (hae: AEStronglyMeasurable (fun x => f x) volume)
    (η : 𝓢(EuclideanSpace ℝ (Fin dm1.succ), ℝ)) (y : EuclideanSpace ℝ (Fin dm1.succ)) :
    Integrable (fun x : EuclideanSpace ℝ (Fin dm1.succ) => fderiv ℝ η x y * f x)
```

lemma intergrable_pow [source]

```
lemma intergrable_pow {dm1 : ℕ} (p: ℤ) (r : ℕ) (p_bound : -dm1 ≤ p)
    (r_bound : (p + ↑dm1).toNat + (invPowMeasure (dm1 := dm1)).integrablePower ≤ r)
    (v : EuclideanSpace ℝ (Fin dm1.succ)) :
    Integrable (fun x => ‖x + v‖ ^ p * ‖((1 + ‖x‖) ^ r)⁻¹‖) volume
```

lemma norm_inv_mul_exists_pow_integrable [source]

```
lemma IsDistBounded.norm_inv_mul_exists_pow_integrable {dm1 : ℕ}
    (f : EuclideanSpace ℝ (Fin dm1.succ) → F) (hf : IsDistBounded f)
    (hae: AEStronglyMeasurable (fun x => f x) volume) :
    ∃ r, Integrable (fun x => ‖f x‖ * ‖((1 + ‖x‖) ^ r)⁻¹‖) volume
```

2.12 PhysLean.Mathematics.Distribution.Function.OfFunction

PhysLean/Mathematics/Distribution/Function/OfFunction.lean on GitHub.

def ofFunction [source]

A distribution `(EuclideanSpace ℝ (Fin 3)) →d[ℝ] F` from a function `f : EuclideanSpace ℝ (Fin 3) → F` bounded by `c1 * ‖x‖ ^ (-2 : ℝ) + c2 * ‖x‖ ^ n`.

```
def ofFunction {dm1 : ℕ} (f : EuclideanSpace ℝ (Fin dm1.succ) → F)
    (hf : IsDistBounded f)
    (hae: AEStronglyMeasurable (fun x => f x) volume) :
    (EuclideanSpace ℝ (Fin dm1.succ)) →d[ℝ] F
```

lemma ofFunction_apply [source]

```
lemma ofFunction_apply {dm1 : ℕ} (f : EuclideanSpace ℝ (Fin dm1.succ) → F)
    (hf : IsDistBounded f)
    (hae: AEStronglyMeasurable (fun x => f x) volume) (η : 𝓢(EuclideanSpace ℝ (Fin dm1.succ),
    ℝ)) :
    ofFunction f hf hae η = ∫ x, η x • f x
```

lemma ofFunction_zero_eq_zero [source]

```
@[simp]
lemma ofFunction_zero_eq_zero {dm1 : ℕ} :
    ofFunction (fun _ : EuclideanSpace ℝ (Fin (dm1 + 1)) => (0 : F))
      ⟨0, 0, 0, 0, by simp⟩ (by fun_prop) = 0
```

lemma ofFunction_smul [source]

```
lemma ofFunction_smul {dm1 : ℕ} (f : EuclideanSpace ℝ (Fin dm1.succ) → F)
    (hf : IsDistBounded f)
    (hae: AEStronglyMeasurable (fun x => f x) volume) (c : ℝ) :
    ofFunction (c • f) (by fun_prop) (by fun_prop) = c • ofFunction f hf hae
```

lemma ofFunction_smul_fun [source]

```
lemma ofFunction_smul_fun {dm1 : ℕ} (f : EuclideanSpace ℝ (Fin dm1.succ) → F)
    (hf : IsDistBounded f)
    (hae: AEStronglyMeasurable (fun x => f x) volume) (c : ℝ) :
    ofFunction (fun x => c • f x) (by
      change IsDistBounded (c • f)
      fun_prop) (by fun_prop) = c • ofFunction f hf hae
```

lemma ofFunction_inner [source]

```
lemma ofFunction_inner {dm1 n : ℕ} (f : EuclideanSpace ℝ (Fin dm1.succ) → EuclideanSpace ℝ (Fin
     n))
    (hf : IsDistBounded f)
    (hae: AEStronglyMeasurable (fun x => f x) volume)
    (η : 𝓢(EuclideanSpace ℝ (Fin dm1.succ), ℝ)) (y : EuclideanSpace ℝ (Fin n)) :
    ⟪ofFunction f hf hae η, y⟫_ℝ = ∫ x, η x * ⟪f x, y⟫_ℝ
```

2.13 PhysLean.Mathematics.Trigonometry.Tanh

PhysLean/Mathematics/Trigonometry/Tanh.lean on GitHub.

lemma polynomial_bounded_on_interval [source]

For a polynomial P, show it's bounded on any bounded interval

```
lemma polynomial_bounded_on_interval (P : Polynomial ℝ) (a b : ℝ) :
    ∃ M : ℝ, ∀ x : ℝ, x ∈ Set.Icc a b → |P.eval x| ≤ M
```

lemma polynomial_tanh_bounded [source]

For a polynomial P, show that P (tanh x) is bounded on the real line

```
lemma polynomial_tanh_bounded (P : Polynomial ℝ) :
    ∃ C : ℝ, ∀ x : ℝ, |P.eval (Real.tanh x)| ≤ C
```

lemma iteratedDeriv_tanh_bounded [source]

The nth derivative of tanh is bounded on the real line

```
lemma iteratedDeriv_tanh_bounded (n : ℕ) :
    ∃ C : ℝ, ∀ x : ℝ, |iteratedDeriv n Real.tanh x| ≤ C
```

lemma contDiff_top_tanh [source]

tanh is infinitely differentiable

```
lemma contDiff_top_tanh : ContDiff ℝ ∞ Real.tanh
```

lemma tanh_hasTemperateGrowth [source]

tanh has temperate growth

```
lemma tanh_hasTemperateGrowth : Function.HasTemperateGrowth Real.tanh
```

lemma iteratedDeriv_tanh_differentiable [source]

Iterated derivative for scaled tanh is differentiable

```
lemma iteratedDeriv_tanh_differentiable (n : ℕ) : Differentiable ℝ (iteratedDeriv n tanh)
```

lemma tanh_const_mul_iteratedDeriv_norm_eq_iteratedFDeriv_norm [source]

Norm of Iterated derivative for scaled tanh is equal to the norm of its Fderiv

```
lemma tanh_const_mul_iteratedDeriv_norm_eq_iteratedFDeriv_norm (n : ℕ) (x : ℝ) :
    ‖iteratedFDeriv ℝ n (fun x => tanh (κ * x)) x‖
    = |iteratedDeriv n (fun x => tanh (κ * x)) x|
```

lemma iteratedDeriv_tanh_const_mul [source]

Iterated derivative for scaled tanh

```
lemma iteratedDeriv_tanh_const_mul (n : ℕ) (κ : ℝ) : ∀ x : ℝ,
    iteratedDeriv n (fun y => Real.tanh (κ * y)) x = κ^n * (iteratedDeriv n Real.tanh) (κ * x)
```

lemma tanh_const_mul_hasTemperateGrowth [source]

tanh(κx) has temperate growth

```
lemma tanh_const_mul_hasTemperateGrowth (κ : ℝ) :
    Function.HasTemperateGrowth (fun x => Real.tanh (κ * x))
```

2.14 PhysLean.Mathematics.VariationalCalculus.HasVarAdjDeriv

PhysLean/Mathematics/VariationalCalculus/HasVarAdjDeriv.lean on GitHub.

lemma congr [source]

```
lemma congr {F G : (X → U) → (Y → V)} {F' } {u : X → U}
    (hF : HasVarAdjDerivAt F F' u) (h : ∀ φ, ContDiff ℝ ∞ φ → F φ = G φ) :
    HasVarAdjDerivAt G F' u where
```

lemma sum [source]

```
lemma sum {ι : Type} [Fintype ι]
    (F : ι → (X → U) → (X → V)) (F' : ι → (X → V) → X → U) (u)
    (hu : ContDiff ℝ ∞ u)
    (hF : ∀ i, HasVarAdjDerivAt (F i) (F' i) u) :
    HasVarAdjDerivAt (fun φ x => ∑ i, F i φ x) (fun ψ x => ∑ i, F' i ψ x) u
```

lemma const_mul [source]

```
lemma const_mul
    (F : (X → U) → (X → ℝ)) (F') (u)
    (hF : HasVarAdjDerivAt F F' u) (c : ℝ) :
    HasVarAdjDerivAt (fun φ x => c * F φ x) (fun ψ x => F' (fun x' => c * ψ x') x) u
```

2.15 PhysLean.Mathematics.VariationalCalculus.HasVarAdjoint

PhysLean/Mathematics/VariationalCalculus/HasVarAdjoint.lean on GitHub.

lemma sum [source]

```
lemma sum {ι : Type} [Fintype ι] {F : ι → (X → U) → (X → V)} {F' : ι → (X → V) → (X → U)}
    (hF : ∀ i, HasVarAdjoint (F i) (F' i)) :
    HasVarAdjoint (fun φ x => ∑ i, F i φ x) (fun φ x => ∑ i, F' i φ x)
```

2.16 PhysLean.Particles.StandardModel.Basic

PhysLean/Particles/StandardModel/Basic.lean on GitHub.

def toSU3 [source]

The underlying element of `SU(3)` of an element in `GaugeGroupI`.

```
def toSU3 : GaugeGroupI →* specialUnitaryGroup (Fin 3) ℂ where
```

def toSU2 [source]

The underlying element of `SU(2)` of an element in `GaugeGroupI`.

```
def toSU2 : GaugeGroupI →* specialUnitaryGroup (Fin 2) ℂ where
```

def toU1 [source]

The underlying element of `U(1)` of an element in `GaugeGroupI`.

```
def toU1 : GaugeGroupI →* unitary ℂ where
```

lemma ext [source]

```
@[ext]
lemma ext {g g' : GaugeGroupI} (hSU3 : toSU3 g = toSU3 g')
    (hSU2 : toSU2 g = toSU2 g') (hU1 : toU1 g = toU1 g') : g = g'
```

instance Star GaugeGroupI [source]

```
instance : Star GaugeGroupI where
```

lemma star_eq [source]

```
lemma star_eq (g : GaugeGroupI) : star g = (star g.1, star g.2.1, star g.2.2)
```

lemma star_toSU3 [source]

```
@[simp]
lemma star_toSU3 (g : GaugeGroupI) : toSU3 (star g) = star (toSU3 g)
```

lemma star_toSU2 [source]

```
@[simp]
lemma star_toSU2 (g : GaugeGroupI) : toSU2 (star g) = star (toSU2 g)
```

lemma star_toU1 [source]

```
@[simp]
lemma star_toU1 (g : GaugeGroupI) : toU1 (star g) = star (toU1 g)
```

instance InvolutiveStar GaugeGroupI [source]

```
instance : InvolutiveStar GaugeGroupI where
```

2.17 PhysLean.Particles.StandardModel.HiggsBoson.Basic

PhysLean/Particles/StandardModel/HiggsBoson/Basic.lean on GitHub.

instance SMul StandardModel.GaugeGroupI HiggsVec [source]

```
instance : SMul StandardModel.GaugeGroupI HiggsVec where
```

lemma gaugeGroupI_smul_eq [source]

```
lemma gaugeGroupI_smul_eq (g : StandardModel.GaugeGroupI) (φ : HiggsVec) :
    g • φ = g.toU1 ^ 3 • (g.toSU2.1 *ᵥ φ)
```

lemma gaugeGroupI_smul_eq_U1_mul_SU2 [source]

```
lemma gaugeGroupI_smul_eq_U1_mul_SU2 (g : StandardModel.GaugeGroupI) (φ : HiggsVec) :
    g • φ = g.toSU2.1 *ᵥ (g.toU1 ^ 3 • φ)
```

instance MulAction StandardModel.GaugeGroupI HiggsVec [source]

```
instance : MulAction StandardModel.GaugeGroupI HiggsVec where
```

lemma gaugeGroupI_smul_inner [source]

```
@[simp]
lemma gaugeGroupI_smul_inner (g : StandardModel.GaugeGroupI) (φ ψ : HiggsVec) :
    ⟪g • φ, g • ψ⟫_ℂ = ⟪φ, ψ⟫_ℂ
```

lemma gaugeGroupI_smul_norm [source]

```
@[simp]
lemma gaugeGroupI_smul_norm (g : StandardModel.GaugeGroupI) (φ : HiggsVec) :
    ‖g • φ‖ = ‖φ‖
```

def toRealGroupElem [source]

Given a Higgs vector, a rotation matrix which puts the second component of the vector to zero, and the first component to a real

```
def toRealGroupElem (φ : HiggsVec) : GaugeGroupI
```

lemma toRealGroupElem_smul_self [source]

```
lemma toRealGroupElem_smul_self (φ : HiggsVec) :
    (toRealGroupElem φ) • φ = ofReal (‖φ‖ ^ 2)
```

lemma mem_orbit_gaugeGroupI_iff [source]

```
lemma mem_orbit_gaugeGroupI_iff (φ : HiggsVec) (ψ : HiggsVec) :
    ψ ∈ MulAction.orbit GaugeGroupI φ ↔ ‖ψ‖ = ‖φ‖
```

def const [source]

Given a vector in `HiggsVec` the constant Higgs field with value equal to that section.

```
def const : HiggsVec →ₗ[ℝ] HiggsField where
```

lemma const_apply [source]

For all spacetime points, the constant Higgs field defined by a Higgs vector, returns that Higgs Vector.

```
@[simp]
lemma const_apply (φ : HiggsVec) (x : SpaceTime) : const φ x = φ
```

lemma const_toHiggsVec_apply [source]

```
lemma const_toHiggsVec_apply (φ : HiggsField) (x : SpaceTime) :
    const (φ.toHiggsVec x) x = φ x
```

instance Inner (SpaceTime → ℂ) (HiggsField) [source]

```
instance : Inner (SpaceTime → ℂ) (HiggsField) where
```

lemma inner_apply [source]

```
lemma inner_apply (φ1 φ2 : HiggsField) (x : SpaceTime) :
    ⟪φ1, φ2⟫_(SpaceTime → ℂ) x = ⟪φ1 x, φ2 x⟫_ℂ
```

lemma inner_eq_expand [source]

```
lemma inner_eq_expand (φ1 φ2 : HiggsField) :
    ⟪φ1, φ2⟫_(SpaceTime → ℂ) = fun x => equivRealProdCLM.symm (((φ1 x 0).re * (φ2 x 0).re
    + (φ1 x 1).re * (φ2 x 1).re+ (φ1 x 0).im * (φ2 x 0).im + (φ1 x 1).im * (φ2 x 1).im),
    ((φ1 x 0).re * (φ2 x 0).im + (φ1 x 1).re * (φ2 x 1).im
    - (φ1 x 0).im * (φ2 x 0).re - (φ1 x 1).im * (φ2 x 1).re))
```

lemma inner_expand_conj [source]

Expands the inner product on Higgs fields in terms of complex components of the Higgs fields.

```
lemma inner_expand_conj (φ1 φ2 : HiggsField) (x : SpaceTime) :
    ⟪φ1, φ2⟫_(SpaceTime → ℂ) x = conj (φ1 x 0) * φ2 x 0 + conj (φ1 x 1) * φ2 x 1
```

lemma inner_symm [source]

```
lemma inner_symm (φ1 φ2 : HiggsField) :
    conj ⟪φ2, φ1⟫_(SpaceTime → ℂ) = ⟪φ1, φ2⟫_(SpaceTime → ℂ)
```

lemma inner_add_left [source]

```
lemma inner_add_left (φ1 φ2 φ3 : HiggsField) :
    ⟪φ1 + φ2, φ3⟫_(SpaceTime → ℂ) = ⟪φ1, φ3⟫_(SpaceTime → ℂ) + ⟪φ2, φ3⟫_(SpaceTime → ℂ)
```

lemma inner_add_right [source]

```
lemma inner_add_right (φ1 φ2 φ3 : HiggsField) :
    ⟪φ1, φ2 + φ3⟫_(SpaceTime → ℂ) = ⟪φ1, φ2⟫_(SpaceTime → ℂ) + ⟪φ1, φ3⟫_(SpaceTime → ℂ)
```

lemma inner_zero_left [source]

```
@[simp]
lemma inner_zero_left (φ : HiggsField) :
    ⟪0, φ⟫_(SpaceTime → ℂ) = 0
```

lemma inner_zero_right [source]

```
@[simp]
lemma inner_zero_right (φ : HiggsField) :
    ⟪φ, 0⟫_(SpaceTime → ℂ) = 0
```

lemma inner_neg_left [source]

```
lemma inner_neg_left (φ1 φ2 : HiggsField) :
    ⟪-φ1, φ2⟫_(SpaceTime → ℂ) = -⟪φ1, φ2⟫_(SpaceTime → ℂ)
```

lemma inner_neg_right [source]

```
lemma inner_neg_right (φ1 φ2 : HiggsField) :
    ⟪φ1, -φ2⟫_(SpaceTime → ℂ) = -⟪φ1, φ2⟫_(SpaceTime → ℂ)
```

lemma inner_smooth [source]

```
lemma inner_smooth (φ1 φ2 : HiggsField) : ContMDiff 𝓘(ℝ, SpaceTime) 𝓘(ℝ, ℂ) ⊤
    ⟪φ1, φ2⟫_(SpaceTime → ℂ)
```

lemma inner_self_eq_normSq [source]

```
lemma inner_self_eq_normSq (φ : HiggsField) (x : SpaceTime) :
    ⟪φ, φ⟫_(SpaceTime → ℂ) x = ‖φ‖_H^2 x
```

lemma normSq_eq_inner_self_re [source]

```
lemma normSq_eq_inner_self_re (φ : HiggsField) (x : SpaceTime) :
    φ.normSq x = (⟪φ, φ⟫_(SpaceTime → ℂ) x).re
```

lemma const_normSq [source]

```
@[simp]
lemma const_normSq (φ : HiggsVec) (x : SpaceTime) :
    ‖const φ‖_H^2 x = ‖φ‖ ^ 2
```

2.18 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.AllowsTerm

PhysLean/Particles/SuperSymmetry/SU5/ChargeSpectrum/AllowsTerm.lean on GitHub.

def AllowsTerm [source]

The charges of representations `x : Charges` allow a potential term `T : PotentialTerm` if the zero charge is in the set of charges associated with that potential term.

```
def AllowsTerm (x : ChargeSpectrum 𝒵) (T : PotentialTerm) : Prop
```

lemma allowsTerm_iff_zero_mem_ofPotentialTerm' [source]

```
lemma allowsTerm_iff_zero_mem_ofPotentialTerm' [DecidableEq 𝒵]
    {x : ChargeSpectrum 𝒵} {T : PotentialTerm} :
    x.AllowsTerm T ↔ 0 ∈ x.ofPotentialTerm' T
```

instance Decidable (x.AllowsTerm T) [source]

```
instance [DecidableEq 𝒵] (x : ChargeSpectrum 𝒵) (T : PotentialTerm) : Decidable (x.AllowsTerm
    T)
```

lemma allowsTerm_mono [source]

```
lemma allowsTerm_mono {T : PotentialTerm} {y x : ChargeSpectrum 𝒵}
    (h : y ⊆ x) (hy : y.AllowsTerm T) : x.AllowsTerm T
```

def allowsTermForm [source]

A element of `Charges` from three intergers `a b c : ℤ` for a given potential term `T`. Defined such that `allowsTermForm a b c T` always allows the potential term `T`, and if any over charge `x` allows `T` then it is due to a subset of the form `allowsTermForm a b c T`.

```
def allowsTermForm (a b c : 𝒵) : (T : PotentialTerm) → ChargeSpectrum 𝒵
  | .μ => ⟨some a, some a, ∅, ∅⟩
  | .β => ⟨none, some a, {a}, ∅⟩
```

```
  | .Λ => ⟨none, none, {a, b}, {- a - b}⟩
  | .W1 => ⟨none, none, {- a - b - c}, {a, b, c}⟩
  | .W2 => ⟨some (- a - b - c), none, ∅, {a, b, c}⟩
  | .W3 => ⟨none, some (- a), {b, - b - 2 • a}, ∅⟩
  | .W4 => ⟨some (- c - 2 • b), some (- b), {c}, ∅⟩
  | .K1 => ⟨none, none, {-a}, {b, - a - b}⟩
  | .K2 => ⟨some a, some b, ∅, {- a - b}⟩
  | .topYukawa => ⟨none, some (-a), ∅, {b, - a - b}⟩
  | .bottomYukawa => ⟨some a, none, {b}, {- a - b}⟩
```

lemma allowsTermForm_allowsTerm [source]

The charge spectrum `allowsTermForm a b c T` allows the potential term `T`.

```
lemma allowsTermForm_allowsTerm {a b c : ℤ} {T : PotentialTerm} :
    (allowsTermForm a b c T).AllowsTerm T
```

lemma allowsTerm_of_eq_allowsTermForm [source]

```
lemma allowsTerm_of_eq_allowsTermForm {T : PotentialTerm}
    (x : ChargeSpectrum ℤ) (h : ∃ a b c, x = allowsTermForm a b c T) :
    x.AllowsTerm T
```

lemma allowsTermForm_eq_of_subset [source]

```
lemma allowsTermForm_eq_of_subset {a b c a' b' c' : ℤ} {T : PotentialTerm}
    (h : allowsTermForm a b c T ⊆ allowsTermForm a' b' c' T) (hT : T ≠ W1 ∧ T ≠ W2) :
    allowsTermForm a b c T = allowsTermForm a' b' c' T
```

lemma allowsTermForm_card_le_degree [source]

```
lemma allowsTermForm_card_le_degree {a b c : ℤ} {T : PotentialTerm} :
    (allowsTermForm a b c T).card ≤ T.degree
```

lemma allowsTermForm_subset_allowsTerm_of_allowsTerm [source]

```
lemma allowsTermForm_subset_allowsTerm_of_allowsTerm {T : PotentialTerm} {x : ChargeSpectrum ℤ}

    (h : x.AllowsTerm T) :
    ∃ a b c, allowsTermForm a b c T ⊆ x ∧ (allowsTermForm a b c T).AllowsTerm T
```

lemma allowsTerm_iff_subset_allowsTermForm [source]

```
lemma allowsTerm_iff_subset_allowsTermForm {T : PotentialTerm} {x : ChargeSpectrum ℤ} :
    x.AllowsTerm T ↔ ∃ a b c, allowsTermForm a b c T ⊆ x
```

lemma subset_card_le_degree_allowsTerm_of_allowsTerm [source]

```
lemma subset_card_le_degree_allowsTerm_of_allowsTerm {T : PotentialTerm} {x : ChargeSpectrum ℤ}

    (h : x.AllowsTerm T) : ∃ y ∈ x.powerset, y.card ≤ T.degree ∧ y.AllowsTerm T
```

def AllowsTermQ5 [source]

The proposition for which says, given a charge `x` adding a charge `q5` permits the existence of a potential term `T` due to the addition of that charge.

```
def AllowsTermQ5 [DecidableEq ℤ] (x : ChargeSpectrum ℤ) (q5 : ℤ) (T : PotentialTerm) : Prop
```

instance Decidable (AllowsTermQ5 x q5 T) [source]

```
instance [DecidableEq ℤ] (x : ChargeSpectrum ℤ) (q5 : ℤ) (T : PotentialTerm) :
    Decidable (AllowsTermQ5 x q5 T)
```

lemma allowsTermQ5_or_allowsTerm_of_allowsTerm_insertQ5 [source]

```
lemma allowsTermQ5_or_allowsTerm_of_allowsTerm_insertQ5 {qHd qHu : Option ℤ}
    {Q5 Q10: Finset ℤ} {q5 : ℤ} (T : PotentialTerm)
    (h : AllowsTerm (qHd, qHu, insert q5 Q5, Q10) T) :
    AllowsTermQ5 (qHd, qHu, Q5, Q10) q5 T ∨
    AllowsTerm (qHd, qHu, Q5, Q10) T
```

lemma allowsTerm_insertQ5_of_allowsTermQ5 [source]

```
lemma allowsTerm_insertQ5_of_allowsTermQ5 {qHd qHu : Option ℤ}
    {Q5 Q10: Finset ℤ} {q5 : ℤ} (T : PotentialTerm)
    (h : AllowsTermQ5 (qHd, qHu, Q5, Q10) q5 T) :
    AllowsTerm (qHd, qHu, insert q5 Q5, Q10) T
```

lemma allowsTerm_insertQ5_iff_allowsTermQ5 [source]

```
lemma allowsTerm_insertQ5_iff_allowsTermQ5 {qHd qHu : Option ℤ}
    {Q5 Q10: Finset ℤ} {q5 : ℤ} (T : PotentialTerm) :
    AllowsTerm (qHd, qHu, insert q5 Q5, Q10) T ↔
    AllowsTermQ5 (qHd, qHu, Q5, Q10) q5 T ∨
    AllowsTerm (qHd, qHu, Q5, Q10) T
```

def AllowsTermQ10 [source]

The proposition for which says, given a charge `x` adding a charge `q5` permits the existence of a potential term `T` due to the addition of that charge.

```
def AllowsTermQ10 [DecidableEq ℤ] (x : ChargeSpectrum ℤ) (q10 : ℤ) (T : PotentialTerm) : Prop
```

instance Decidable (AllowsTermQ10 x q10 T) [source]

```
instance [DecidableEq ℤ] (x : ChargeSpectrum ℤ) (q10 : ℤ) (T : PotentialTerm) :
    Decidable (AllowsTermQ10 x q10 T)
```

lemma allowsTermQ10_or_allowsTerm_of_allowsTerm_insertQ10 [source]

```
lemma allowsTermQ10_or_allowsTerm_of_allowsTerm_insertQ10 {qHd qHu : Option ℤ}
    {Q5 Q10: Finset ℤ} {q10 : ℤ} (T : PotentialTerm)
    (h : AllowsTerm (qHd, qHu, Q5, insert q10 Q10) T) :
    AllowsTermQ10 (qHd, qHu, Q5, Q10) q10 T ∨
```

```
    AllowsTerm ⟨qHd, qHu, Q5, Q10⟩ T
```

lemma allowsTerm_insertQ10_of_allowsTermQ10 [source]

```
lemma allowsTerm_insertQ10_of_allowsTermQ10 {qHd qHu : Option ℤ}
    {Q5 Q10: Finset ℤ} {q10 : ℤ} (T : PotentialTerm)
    (h : AllowsTermQ10 ⟨qHd, qHu, Q5, Q10⟩ q10 T) :
    AllowsTerm ⟨qHd, qHu, Q5, insert q10 Q10⟩ T
```

lemma allowsTerm_insertQ10_iff_allowsTermQ10 [source]

```
lemma allowsTerm_insertQ10_iff_allowsTermQ10 {qHd qHu : Option ℤ}
    {Q5 Q10: Finset ℤ} {q10 : ℤ} (T : PotentialTerm) :
    AllowsTerm ⟨qHd, qHu, Q5, insert q10 Q10⟩ T ↔
    AllowsTermQ10 ⟨qHd, qHu, Q5, Q10⟩ q10 T ∨
    AllowsTerm ⟨qHd, qHu, Q5, Q10⟩ T
```

2.19 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.Basic

PhysLean/Particles/SuperSymmetry/SU5/ChargeSpectrum/Basic.lean on GitHub.

structure ChargeSpectrum [source]

The type such that an element corresponds to the collection of charges associated with the matter content of the theory. The order of charges is implicitly taken to be `qHd`, `qHu`, `Q5`, `Q10`.

The `Q5` and `Q10` charges are represented by `Finset` rather than `Multiset`, so multiplicity is not included.

This is defined for a general type $\mathcal{Z}$, which could be e.g. - $\mathbb{Z}$ in the case of `U(1)`, - $\mathbb{Z} \times \mathbb{Z}$ in the case of `U(1) × U(1)`, - `Fin 2` in the case of $\mathbb{Z}_2$ etc.

```
structure ChargeSpectrum (ℤ : Type := ℤ) where
  qHd : Option ℤ
  qHu : Option ℤ
  Q5 : Finset ℤ
  Q10 : Finset ℤ
```

lemma eq_of_parts [source]

```
lemma eq_of_parts {x y : ChargeSpectrum ℤ} (h1 : x.qHd = y.qHd) (h2 : x.qHu = y.qHu)
    (h3 : x.Q5 = y.Q5) (h4 : x.Q10 = y.Q10) : x = y
```

lemma eq_iff [source]

```
lemma eq_iff {x y : ChargeSpectrum ℤ} :
    x = y ↔ x.qHd = y.qHd ∧ x.qHu = y.qHu ∧ x.Q5 = y.Q5 ∧ x.Q10 = y.Q10
```

instance DecidableEq (ChargeSpectrum ℤ) [source]

```
instance [DecidableEq ℤ] : DecidableEq (ChargeSpectrum ℤ)
```

def toProd [source]

The explicit casting of a term of type `Charges 𝓩` to a term of `Option 𝓩 × Option 𝓩 × Finset 𝓩 × Finset 𝓩`.

```
def toProd : ChargeSpectrum 𝓩 ≃ Option 𝓩 × Option 𝓩 × Finset 𝓩 × Finset 𝓩 where
```

instance Repr (ChargeSpectrum 𝓩) [source]

```
unsafe instance [Repr 𝓩] : Repr (ChargeSpectrum 𝓩) where
```

instance hasSubset [source]

```
instance hasSubset : HasSubset (ChargeSpectrum 𝓩) where
```

instance hasSSubset [source]

```
instance hasSSubset : HasSSubset (ChargeSpectrum 𝓩) where
```

instance subsetDecidable [source]

```
instance subsetDecidable [DecidableEq 𝓩] (x y : ChargeSpectrum 𝓩) : Decidable (x ⊆ y)
```

lemma subset_def [source]

```
lemma subset_def {x y : ChargeSpectrum 𝓩} : x ⊆ y ↔ x.qHd.toFinset ⊆ y.qHd.toFinset ∧
    x.qHu.toFinset ⊆ y.qHu.toFinset ∧ x.Q5 ⊆ y.Q5 ∧ x.Q10 ⊆ y.Q10
```

lemma subset_refl [source]

```
@[simp, refl]
lemma subset_refl (x : ChargeSpectrum 𝓩) : x ⊆ x
```

lemma subset_trans [source]

```
lemma subset_trans {x y z : ChargeSpectrum 𝓩} (hxy : x ⊆ y) (hyz : y ⊆ z) : x ⊆ z
```

lemma subset_antisymm [source]

```
lemma subset_antisymm {x y : ChargeSpectrum 𝓩} (hxy : x ⊆ y) (hyx : y ⊆ x) : x = y
```

instance emptyInst [source]

```
instance emptyInst : EmptyCollection (ChargeSpectrum 𝓩) where
```

lemma empty_subset [source]

```
@[simp]
lemma empty_subset (x : ChargeSpectrum 𝓩) : ∅ ⊆ x
```

lemma subset_of_empty_iff_empty [source]

```
@[simp]
lemma subset_of_empty_iff_empty {x : ChargeSpectrum 𝒵} :
    x ⊆ ∅ ↔ x = ∅
```

lemma empty_qHd [source]

```
@[simp]
lemma empty_qHd : (∅ : ChargeSpectrum 𝒵).qHd = none
```

lemma empty_qHu [source]

```
@[simp]
lemma empty_qHu : (∅ : ChargeSpectrum 𝒵).qHu = none
```

lemma empty_Q5 [source]

```
@[simp]
lemma empty_Q5 : (∅ : ChargeSpectrum 𝒵).Q5 = ∅
```

lemma empty_Q10 [source]

```
@[simp]
lemma empty_Q10 : (∅ : ChargeSpectrum 𝒵).Q10 = ∅
```

def card [source]

The cardinality of a `Charges` is defined to be the sum of the cardinalities of each of the underlying finite sets of charges, with `Option ℤ` turned to finsets.

```
def card (x : ChargeSpectrum 𝒵) : Nat
```

lemma card_empty [source]

```
@[simp]
lemma card_empty : card (∅ : ChargeSpectrum 𝒵) = 0
```

lemma card_mono [source]

```
lemma card_mono {x y : ChargeSpectrum 𝒵} (h : x ⊆ y) : card x ≤ card y
```

lemma eq_of_subset_card [source]

```
lemma eq_of_subset_card {x y : ChargeSpectrum 𝒵} (h : x ⊆ y) (hcard : card x = card y) : x = y
```

def powerset [source]

The powerset of a charge . Given a charge `x : Charges` it's powerset is the finite set of all `Charges` which are subsets of `x`.

```
def powerset (x : ChargeSpectrum 𝒵) : Finset (ChargeSpectrum 𝒵)
```

lemma mem_powerset_iff [source]

```
lemma mem_powerset_iff {x y : ChargeSpectrum 𝒵} :
    x ∈ powerset y ↔
    x.qHd ∈ y.qHd.powerset ∧
    x.qHu ∈ y.qHu.powerset ∧
    x.Q5 ∈ y.Q5.powerset ∧
    x.Q10 ∈ y.Q10.powerset
```

lemma mem_powerset_iff_subset [source]

```
@[simp]
lemma mem_powerset_iff_subset {x y : ChargeSpectrum 𝒵} :
    x ∈ powerset y ↔ x ⊆ y
```

lemma self_mem_powerset [source]

```
lemma self_mem_powerset (x : ChargeSpectrum 𝒵) :
    x ∈ powerset x
```

lemma empty_mem_powerset [source]

```
lemma empty_mem_powerset (x : ChargeSpectrum 𝒵) :
    ∅ ∈ powerset x
```

lemma powerset_of_empty [source]

```
@[simp]
lemma powerset_of_empty :
    powerset (∅ : ChargeSpectrum 𝒵) = {∅}
```

lemma powerset_mono [source]

```
lemma powerset_mono {x y : ChargeSpectrum 𝒵} :
    powerset x ⊆ powerset y ↔ x ⊆ y
```

lemma min_exists_inductive [source]

```
lemma min_exists_inductive (S : Finset (ChargeSpectrum 𝒵)) (hS : S ≠ ∅) :
    (n : ℕ) → (hn : S.card = n) →
    ∃ y ∈ S, powerset y ∩ S = {y}
  | 0, h => by simp_all
  | 1, h => by
    rw [Finset.card_eq_one] at h
    obtain ⟨y, rfl⟩
```

lemma min_exists [source]

```
lemma min_exists (S : Finset (ChargeSpectrum 𝒵)) (hS : S ≠ ∅) :
    ∃ y ∈ S, powerset y ∩ S = {y}
```

def ofFinset [source]

Given `S5 S10 : Finset ℤ` the finite set of charges associated with for which the 5-bar representation charges sit in `S5` and the 10d representation charges sit in `S10`.

```
def ofFinset (S5 S10 : Finset ℤ) : Finset (ChargeSpectrum ℤ)
```

lemma mem_ofFinset_iff [source]

```
lemma mem_ofFinset_iff {S5 S10 : Finset ℤ} {x : ChargeSpectrum ℤ} :
    x ∈ ofFinset S5 S10 ↔ x.qHd.toFinset ⊆ S5 ∧ x.qHu.toFinset ⊆ S5 ∧
      x.Q5 ⊆ S5 ∧ x.Q10 ⊆ S10
```

lemma mem_ofFinset_antitone [source]

```
lemma mem_ofFinset_antitone (S5 S10 : Finset ℤ)
    {x y : ChargeSpectrum ℤ} (h : x ⊆ y) (hy : y ∈ ofFinset S5 S10) :
    x ∈ ofFinset S5 S10
```

lemma ofFinset_subset_of_subset [source]

```
lemma ofFinset_subset_of_subset {S5 S5' S10 S10' : Finset ℤ}
    (h5 : S5 ⊆ S5') (h10 : S10 ⊆ S10') :
    ofFinset S5 S10 ⊆ ofFinset S5' S10'
```

lemma ofFinset_univ [source]

```
lemma ofFinset_univ [Fintype ℤ] (x : ChargeSpectrum ℤ) :
    x ∈ ofFinset (Finset.univ : Finset ℤ) (Finset.univ : Finset ℤ)
```

2.20 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.Completions

PhysLean/Particles/SuperSymmetry/SU5/ChargeSpectrum/Completions.lean on GitHub.

def IsComplete [source]

A charge spectrum is complete if it has all types of fields.

```
def IsComplete (x : ChargeSpectrum ℤ) : Prop
```

instance Decidable (IsComplete x) [source]

```
instance [DecidableEq ℤ] (x : ChargeSpectrum ℤ) : Decidable (IsComplete x)
```

lemma not_isComplete_empty [source]

```
@[simp]
lemma not_isComplete_empty : ¬ IsComplete (∅ : ChargeSpectrum ℤ)
```

lemma isComplete_mono [source]

```
lemma isComplete_mono {x y : ChargeSpectrum 𝒵} (h : x ⊆ y) (hx : IsComplete x) :
    IsComplete y
```

def completions [source]

Given a collection of charges `x` in `ofFinset S5 S10`, the minimimal charges `y` in `ofFinset S5 S10` which are a super sets of `x` and are complete.

```
def completions (S5 S10 : Finset 𝒵) (x : ChargeSpectrum 𝒵) : Multiset (ChargeSpectrum 𝒵)
```

lemma mem_completions_iff [source]

```
lemma mem_completions_iff {S5 S10 : Finset 𝒵} {x y : ChargeSpectrum 𝒵} :
    y ∈ completions S5 S10 x ↔
    y.qHd ∈ (if x.qHd.isSome then {x.qHd} else S5.val.map fun y => some y) ∧
    y.qHu ∈ (if x.qHu.isSome then {x.qHu} else S5.val.map fun y => some y) ∧
    y.Q5 ∈ (if x.Q5 ≠ ∅ then {x.Q5} else S5.val.map fun y => {y}) ∧
    y.Q10 ∈ (if x.Q10 ≠ ∅ then {x.Q10} else S10.val.map fun y => {y})
```

lemma completions_nodup [source]

For speed we define `completions` to be a multiset, but in fact it has no duplicates, so it could be defined as a finite set.

```
lemma completions_nodup (S5 S10 : Finset 𝒵) (x : ChargeSpectrum 𝒵) :
    (completions S5 S10 x).Nodup
```

lemma completions_eq_singleton_of_complete [source]

The completions of a complete charge spectrum is just the singleton containing itself.

```
lemma completions_eq_singleton_of_complete {S5 S10 : Finset 𝒵} (x : ChargeSpectrum 𝒵)
    (hcomplete : IsComplete x) :
    completions S5 S10 x = {x}
```

lemma self_mem_completions_iff_isComplete [source]

If a charge spectrum `x` is a member of its own completion then it is complete, and vice versa.

```
@[simp]
lemma self_mem_completions_iff_isComplete {S5 S10 : Finset 𝒵} (x : ChargeSpectrum 𝒵) :
    x ∈ completions S5 S10 x ↔ IsComplete x
```

lemma mem_completions_isComplete [source]

A charge spectrum which is a member of the completions of another charge spectrum is complete.

```
lemma mem_completions_isComplete {S5 S10 : Finset 𝒵} {x y : ChargeSpectrum 𝒵}
    (hx : y ∈ completions S5 S10 x) : IsComplete y
```

lemma self_subset_mem_completions [source]

If `y` is in the completions of `x` then `x ⊆ y`.

```
lemma self_subset_mem_completions (S5 S10 : Finset 𝒵) (x y : ChargeSpectrum 𝒵)
    (hy : y ∈ completions S5 S10 x) : x ⊆ y
```

lemma exist_completions_subset_of_complete [source]

```
lemma exist_completions_subset_of_complete (S5 S10 : Finset 𝒵) (x y : ChargeSpectrum 𝒵)
    (hsubset : x ⊆ y) (hy : y ∈ ofFinset S5 S10) (hycomplete : IsComplete y) :
    ∃ z ∈ completions S5 S10 x, z ⊆ y
```

def completionsTopYukawa [source]

A fast version of `completions` for an `x` which is in `minimallyAllowsTermsOfFinset S5 S10 .topYukawa`.

```
def completionsTopYukawa (S5 : Finset 𝒵) (x : ChargeSpectrum 𝒵) :
    Multiset (ChargeSpectrum 𝒵)
```

lemma completionsTopYukawa_nodup [source]

The multiset `completionsTopYukawa S5 x` has no duplicates.

```
lemma completionsTopYukawa_nodup {S5 : Finset 𝒵} (x : ChargeSpectrum 𝒵) :
    (completionsTopYukawa S5 x).Nodup
```

lemma completions_eq_completionsTopYukawa_of_mem_minimallyAllowsTermsOfFinset [source]

The multisets `completions S5 S10 x` and `completionsTopYukawa S5 x` are equivalent if `x` minimally allows the top Yukawa.

```
lemma completions_eq_completionsTopYukawa_of_mem_minimallyAllowsTermsOfFinset [AddCommGroup 𝒵]
    {S5 S10 : Finset 𝒵} (x : ChargeSpectrum 𝒵)
    (hx : x ∈ minimallyAllowsTermsOfFinset S5 S10 .topYukawa) :
    completions S5 S10 x = completionsTopYukawa S5 x
```

2.21 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.Map

PhysLean/Particles/SuperSymmetry/SU5/ChargeSpectrum/Map.lean on GitHub.

def map [source]

Given an additive monoid homomorphisms `f : 𝒵 →+ 𝒵1`, for a charge `x : Charges 𝒵`, `x.map f` is the charge of `Charges 𝒵1` obtained by mapping the elements of `x` by `f`.

```
def map (f : 𝒵 →+ 𝒵1) (x : ChargeSpectrum 𝒵) : ChargeSpectrum 𝒵1 where
```

lemma map_empty [source]

```
@[simp]
lemma map_empty (f : 𝒵 →+ 𝒵1) : map f (∅ : ChargeSpectrum 𝒵) = ∅
```

lemma map_map [source]

```
lemma map_map (f : 𝓩 →+ 𝓩1) (g : 𝓩1 →+ 𝓩2) (x : ChargeSpectrum 𝓩) :
    map g (map f x) = map (g.comp f) x
```

lemma map_id [source]

```
@[simp]
lemma map_id [DecidableEq 𝓩] (x : ChargeSpectrum 𝓩) : map (AddMonoidHom.id 𝓩) x = x
```

lemma map_ofFieldLabel [source]

```
lemma map_ofFieldLabel (f : 𝓩 →+ 𝓩1) (x : ChargeSpectrum 𝓩) (F : FieldLabel) :
    ofFieldLabel (map f x) F = (ofFieldLabel x F).image f
```

lemma map_subset [source]

```
lemma map_subset {f : 𝓩 →+ 𝓩1} {x y : ChargeSpectrum 𝓩} (h : x ⊆ y) :
    map f x ⊆ map f y
```

lemma map_ofPotentialTerm_toFinset [source]

```
lemma map_ofPotentialTerm_toFinset [DecidableEq 𝓩]
    (f : 𝓩 →+ 𝓩1) (x : ChargeSpectrum 𝓩) (T : PotentialTerm) :
    (ofPotentialTerm (map f x) T).toFinset = (ofPotentialTerm x T).toFinset.image f
```

lemma mem_map_ofPotentialTerm_iff [source]

```
lemma mem_map_ofPotentialTerm_iff [DecidableEq 𝓩]
    (f : 𝓩 →+ 𝓩1) (x : ChargeSpectrum 𝓩) (T : PotentialTerm) :
    i ∈ (ofPotentialTerm (map f x) T) ↔ i ∈ (ofPotentialTerm x T).map f
```

lemma mem_map_ofPotentialTerm'_iff [source]

```
lemma mem_map_ofPotentialTerm'_iff[DecidableEq 𝓩]
    (f : 𝓩 →+ 𝓩1) (x : ChargeSpectrum 𝓩) (T : PotentialTerm) :
    i ∈ (ofPotentialTerm' (map f x) T) ↔ i ∈ (ofPotentialTerm' x T).map f
```

lemma map_ofPotentialTerm'_toFinset [source]

```
lemma map_ofPotentialTerm'_toFinset [DecidableEq 𝓩]
    (f : 𝓩 →+ 𝓩1) (x : ChargeSpectrum 𝓩) (T : PotentialTerm) :
    (ofPotentialTerm' (map f x) T).toFinset = (ofPotentialTerm' x T).toFinset.image f
```

lemma allowsTermForm_map [source]

```
lemma allowsTermForm_map {T} {f : 𝓩 →+ 𝓩1} {a b c : 𝓩} :
    (allowsTermForm a b c T).map f = allowsTermForm (f a) (f b) (f c) T
```

lemma map_allowsTerm [source]

```
lemma map_allowsTerm {f : 𝒵 →+ 𝒵1} {x : ChargeSpectrum 𝒵} {T : PotentialTerm}
    (h : x.AllowsTerm T) : (map f x).AllowsTerm T
```

lemma map_isPhenoConstrained [source]

```
lemma map_isPhenoConstrained (f : 𝒵 →+ 𝒵1) {x : ChargeSpectrum 𝒵}
    (h : x.IsPhenoConstrained) : (map f x).IsPhenoConstrained
```

lemma not_isPhenoConstrained_of_map [source]

```
lemma not_isPhenoConstrained_of_map {f : 𝒵 →+ 𝒵1} {x : ChargeSpectrum 𝒵}
    (h : ¬ (map f x).IsPhenoConstrained) : ¬ x.IsPhenoConstrained
```

lemma map_isComplete_iff [source]

```
lemma map_isComplete_iff {f : 𝒵 →+ 𝒵1} {x : ChargeSpectrum 𝒵} :
    (map f x).IsComplete ↔ x.IsComplete
```

lemma map_ofYukawaTerms_toFinset [source]

```
lemma map_ofYukawaTerms_toFinset {f : 𝒵 →+ 𝒵1} {x : ChargeSpectrum 𝒵} :
    (map f x).ofYukawaTerms.toFinset = x.ofYukawaTerms.toFinset.image f
```

lemma mem_map_ofYukawaTerms_iff [source]

```
lemma mem_map_ofYukawaTerms_iff {f : 𝒵 →+ 𝒵1} {x : ChargeSpectrum 𝒵} {i} :
    i ∈ (map f x).ofYukawaTerms ↔ i ∈ x.ofYukawaTerms.map f
```

lemma map_ofYukawaTermsNSum_toFinset [source]

```
lemma map_ofYukawaTermsNSum_toFinset {f : 𝒵 →+ 𝒵1} {x : ChargeSpectrum 𝒵} {n : ℕ}:
    ((map f x).ofYukawaTermsNSum n).toFinset = (x.ofYukawaTermsNSum n).toFinset.image f
```

lemma mem_map_ofYukawaTermsNSum_iff [source]

```
lemma mem_map_ofYukawaTermsNSum_iff {f : 𝒵 →+ 𝒵1} {x : ChargeSpectrum 𝒵} {n i} :
    i ∈ (map f x).ofYukawaTermsNSum n ↔ i ∈ (x.ofYukawaTermsNSum n).map f
```

lemma map_phenoConstrainingChargesSP_toFinset [source]

```
lemma map_phenoConstrainingChargesSP_toFinset {f : 𝒵 →+ 𝒵1} {x : ChargeSpectrum 𝒵} :
    (map f x).phenoConstrainingChargesSP.toFinset =
    x.phenoConstrainingChargesSP.toFinset.image f
```

lemma map_yukawaGeneratesDangerousAtLevel [source]

```
lemma map_yukawaGeneratesDangerousAtLevel (f : 𝒵 →+ 𝒵1) {x : ChargeSpectrum 𝒵} (n : ℕ)
    (h : x.YukawaGeneratesDangerousAtLevel n) : (map f x).YukawaGeneratesDangerousAtLevel n
```

lemma not_yukawaGeneratesDangerousAtLevel_of_map [source]

```
lemma not_yukawaGeneratesDangerousAtLevel_of_map {f : 𝓩 →+ 𝓩1} {x : ChargeSpectrum 𝓩}
    (n : ℕ) (h : ¬ (map f x).YukawaGeneratesDangerousAtLevel n) :
    ¬ x.YukawaGeneratesDangerousAtLevel n
```

def preimageOfFinset [source]

The preimage of a charge `Charges 𝓩1` in `ofFinset S5 S10 ⊆ Charges 𝓩` under mapping charges through `f : 𝓩 →+ 𝓩1`.

```
def preimageOfFinset (S5 S10 : Finset 𝓩) (f : 𝓩 →+ 𝓩1)
    (x : ChargeSpectrum 𝓩1) : Finset (ChargeSpectrum 𝓩)
```

lemma preimageOfFinset_eq [source]

```
lemma preimageOfFinset_eq (S5 S10 : Finset 𝓩) (f : 𝓩 →+ 𝓩1) (x : ChargeSpectrum 𝓩1) :
    preimageOfFinset S5 S10 f x = {y : ChargeSpectrum 𝓩 | y.map f = x ∧ y ∈ ofFinset S5 S10}
```

def preimageOfFinsetCard [source]

The cardiniality of the preimage of a charge `Charges 𝓩1` in `ofFinset S5 S10 ⊆ Charges 𝓩` under mapping charges through `f : 𝓩 →+ 𝓩1`.

```
def preimageOfFinsetCard (S5 S10 : Finset 𝓩) (f : 𝓩 →+ 𝓩1) (x : ChargeSpectrum 𝓩1) : ℕ
```

lemma preimageOfFinset_card_eq [source]

```
lemma preimageOfFinset_card_eq (S5 S10 : Finset 𝓩) (f : 𝓩 →+ 𝓩1) (x : ChargeSpectrum 𝓩1) :
    preimageOfFinsetCard S5 S10 f x =
    (preimageOfFinset S5 S10 f x).card
```

2.22 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.MinimallyAllowsTerm.Basic

PhysLean/Particles/SuperSymmetry/SU5/ChargeSpectrum/MinimallyAllowsTerm/Basic.lean on GitHub.

def MinimallyAllowsTerm [source]

A collection of charges `x : Charges` is said to minimally allow the potential term `T` if it allows `T` and no strict subset of it allows `T`.

```
def MinimallyAllowsTerm (x : ChargeSpectrum 𝓩) (T : PotentialTerm) : Prop
```

instance Decidable (x.MinimallyAllowsTerm T) [source]

```
instance (x : ChargeSpectrum 𝓩) (T : PotentialTerm) : Decidable (x.MinimallyAllowsTerm T)
```

lemma allowsTerm_of_minimallyAllowsTerm [source]

```
lemma allowsTerm_of_minimallyAllowsTerm (h : x.MinimallyAllowsTerm T) : x.AllowsTerm T
```

lemma allowsTerm_of_has_minimallyAllowsTerm_subset [source]

```
lemma allowsTerm_of_has_minimallyAllowsTerm_subset
    (hx : ∃ y ∈ powerset x, y.MinimallyAllowsTerm T) : x.AllowsTerm T
```

lemma minimallyAllowsTerm_iff_powerset_filter_eq [source]

```
lemma minimallyAllowsTerm_iff_powerset_filter_eq :
    x.MinimallyAllowsTerm T ↔ x.powerset.filter (fun y => y.AllowsTerm T) = {x}
```

lemma minimallyAllowsTerm_iff_powerset_countP_eq_one [source]

```
lemma minimallyAllowsTerm_iff_powerset_countP_eq_one :
    x.MinimallyAllowsTerm T ↔ x.powerset.val.countP (fun y => y.AllowsTerm T) = 1
```

lemma subset_minimallyAllowsTerm_of_allowsTerm [source]

```
lemma subset_minimallyAllowsTerm_of_allowsTerm
    (hx : x.AllowsTerm T) : ∃ y ∈ powerset x, y.MinimallyAllowsTerm T
```

lemma allowsTerm_iff_subset_minimallyAllowsTerm [source]

```
lemma allowsTerm_iff_subset_minimallyAllowsTerm :
    x.AllowsTerm T ↔ ∃ y ∈ powerset x, y.MinimallyAllowsTerm T
```

lemma card_le_degree_of_minimallyAllowsTerm [source]

```
lemma card_le_degree_of_minimallyAllowsTerm (h : x.MinimallyAllowsTerm T) :
    x.card ≤ T.degree
```

lemma eq_allowsTermForm_of_minimallyAllowsTerm [source]

```
lemma eq_allowsTermForm_of_minimallyAllowsTerm (h1 : x.MinimallyAllowsTerm T) :
    ∃ a b c, x = allowsTermForm a b c T
```

lemma allowsTermForm_minimallyAllowsTerm [source]

```
lemma allowsTermForm_minimallyAllowsTerm {a b c : ℤ} {T : PotentialTerm}
    (hT : T ≠ W1 ∧ T ≠ W2) :
    MinimallyAllowsTerm (allowsTermForm a b c T) T
```

2.23 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.MinimallyAllowsTerm.FinsetTerms

PhysLean/Particles/SuperSymmetry/SU5/ChargeSpectrum/MinimallyAllowsTerm/FinsetTerms.lean on GitHub.

def MinimallyAllowsFinsetTerms [source]

A collection of charge spectra is said to minimally allow a finite set of potential terms `Ts` if it allows all terms in `Ts` and no strict subset of it allows all terms in `Ts`.

```
def MinimallyAllowsFinsetTerms (x : ChargeSpectrum ℤ) (Ts : Finset PotentialTerm) : Prop
```

instance Decidable (x.MinimallyAllowsFinsetTerms Ts) [source]

```
instance (x : ChargeSpectrum ℤ) (Ts : Finset PotentialTerm) :
    Decidable (x.MinimallyAllowsFinsetTerms Ts)
```

lemma allowsTerm_of_minimallyAllowsFinsetTerms [source]

```
lemma allowsTerm_of_minimallyAllowsFinsetTerms {T : PotentialTerm}
    (h : x.MinimallyAllowsFinsetTerms Ts) (hT : T ∈ Ts) : x.AllowsTerm T
```

lemma minimallyAllowsFinsetTerms_singleton [source]

```
@[simp]
lemma minimallyAllowsFinsetTerms_singleton {T : PotentialTerm} :
    x.MinimallyAllowsFinsetTerms {T} ↔ x.MinimallyAllowsTerm T
```

def minTopBottom [source]

The set of charges of the form `(qHd, qHu, {q5}, {-qHd-q5, q10, qHu - q10})` This includes every charge which minimally allows for the top and bottom Yukawas.

```
def minTopBottom (S5 S10 : Finset ℤ) : Multiset (ChargeSpectrum ℤ)
```

lemma allowsTerm_topYukawa_of_mem_minTopBottom [source]

```
lemma allowsTerm_topYukawa_of_mem_minTopBottom {S5 S10 : Finset ℤ}
    {x : ChargeSpectrum ℤ} (h : x ∈ minTopBottom S5 S10) :
    x.AllowsTerm topYukawa
```

lemma allowsTerm_bottomYukawa_of_mem_minTopBottom [source]

```
lemma allowsTerm_bottomYukawa_of_mem_minTopBottom {S5 S10 : Finset ℤ}
    {x : ChargeSpectrum ℤ} (h : x ∈ minTopBottom S5 S10) :
    x.AllowsTerm bottomYukawa
```

lemma mem_minTopBottom_of_minimallyAllowsFinsetTerms [source]

```
lemma mem_minTopBottom_of_minimallyAllowsFinsetTerms
    {x : ChargeSpectrum ℤ} {S5 S10 : Finset ℤ}
    (h : x.MinimallyAllowsFinsetTerms {topYukawa, bottomYukawa})
    (hx : x ∈ ofFinset S5 S10) :
    x ∈ minTopBottom S5 S10
```

2.24 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.MinimallyAllowsTerm.OfFinset

PhysLean/Particles/SuperSymmetry/SU5/ChargeSpectrum/MinimallyAllowsTerm/OfFinset.lean on GitHub.

def toMultisetsOne [source]

The multisets of cardinality `1` containing elements from a finite set `s`.

```
def toMultisetsOne (s : Finset ℤ) : Multiset (Multiset ℤ)
```

lemma mem_toMultisetsOne_iff [source]

```
@[simp]
lemma mem_toMultisetsOne_iff [DecidableEq ℤ] {s : Finset ℤ} (X : Multiset ℤ) :
    X ∈ toMultisetsOne s ↔ X.toFinset ⊆ s ∧ X.card = 1
```

def toMultisetsTwo [source]

The multisets of cardinality `2` containing elements from a finite set `s`.

```
def toMultisetsTwo (s : Finset ℤ) : Multiset (Multiset ℤ)
```

lemma mem_toMultisetsTwo_iff [source]

```
@[simp]
lemma mem_toMultisetsTwo_iff [DecidableEq ℤ] {s : Finset ℤ} (X : Multiset ℤ) :
    X ∈ toMultisetsTwo s ↔ X.toFinset ⊆ s ∧ X.card = 2
```

def toMultisetsThree [source]

The multisets of cardinality `3` containing elements from a finite set `s`.

```
def toMultisetsThree [DecidableEq ℤ] (s : Finset ℤ) : Multiset (Multiset ℤ)
```

lemma mem_toMultisetsThree_iff [source]

```
@[simp]
lemma mem_toMultisetsThree_iff [DecidableEq ℤ] {s : Finset ℤ} (X : Multiset ℤ) :
    X ∈ toMultisetsThree s ↔ X.toFinset ⊆ s ∧ X.card = 3
```

def minimallyAllowsTermsOfFinset [source]

The multiset of all charges within `ofFinset S5 S10` which minimally allow the potential term `T`.

```
def minimallyAllowsTermsOfFinset (S5 S10 : Finset ℤ) :
    (T : PotentialTerm) → Multiset (ChargeSpectrum ℤ)
  | μ =>
    let SqHd
```

lemma mem_ofFinset_of_mem_minimallyAllowsTermOfFinset [source]

```
lemma mem_ofFinset_of_mem_minimallyAllowsTermOfFinset {S5 S10 : Finset ℤ} {T : PotentialTerm}
    {x : ChargeSpectrum ℤ} (hx : x ∈ minimallyAllowsTermsOfFinset S5 S10 T) :
    x ∈ ofFinset S5 S10
```

lemma minimallyAllowsTermOfFinset_subset_ofFinset [source]

```
lemma minimallyAllowsTermOfFinset_subset_ofFinset {S5 S10 : Finset ℤ} {T : PotentialTerm} :
    minimallyAllowsTermsOfFinset S5 S10 T ⊆ (ofFinset S5 S10).val
```

lemma eq_allowsTermForm_of_mem_minimallyAllowsTermOfFinset [source]

```
lemma eq_allowsTermForm_of_mem_minimallyAllowsTermOfFinset {S5 S10 : Finset ℤ} {T :
    PotentialTerm}
    {x : ChargeSpectrum ℤ} (hx : x ∈ minimallyAllowsTermsOfFinset S5 S10 T) :
    ∃ a b c, x = allowsTermForm a b c T
```

lemma allowsTerm_of_mem_minimallyAllowsTermOfFinset [source]

```
lemma allowsTerm_of_mem_minimallyAllowsTermOfFinset {S5 S10 : Finset ℤ} {T : PotentialTerm}
    {x : ChargeSpectrum ℤ} (hx : x ∈ minimallyAllowsTermsOfFinset S5 S10 T) :
    x.AllowsTerm T
```

lemma minimallyAllowsTerm_of_mem_minimallyAllowsTermOfFinset [source]

```
lemma minimallyAllowsTerm_of_mem_minimallyAllowsTermOfFinset {S5 S10 : Finset ℤ}
    {T : PotentialTerm} {x : ChargeSpectrum ℤ}
    (hx : x ∈ minimallyAllowsTermsOfFinset S5 S10 T) :
    x.MinimallyAllowsTerm T
```

lemma mem_minimallyAllowsTermOfFinset_of_minimallyAllowsTerm [source]

```
lemma mem_minimallyAllowsTermOfFinset_of_minimallyAllowsTerm {S5 S10 : Finset ℤ}
    {T : PotentialTerm} (x : ChargeSpectrum ℤ) (h : x.MinimallyAllowsTerm T)
    (hx : x ∈ ofFinset S5 S10) :
    x ∈ minimallyAllowsTermsOfFinset S5 S10 T
```

lemma minimallyAllowsTerm_iff_mem_minimallyAllowsTermOfFinset [source]

```
lemma minimallyAllowsTerm_iff_mem_minimallyAllowsTermOfFinset
    {S5 S10 : Finset ℤ} {T : PotentialTerm}
    {x : ChargeSpectrum ℤ} (hx : x ∈ ofFinset S5 S10) :
    x.MinimallyAllowsTerm T ↔ x ∈ minimallyAllowsTermsOfFinset S5 S10 T
```

lemma minimallyAllowsTermOfFinset_subset_of_subset [source]

```
lemma minimallyAllowsTermOfFinset_subset_of_subset {S5 S5' S10 S10' : Finset ℤ} {T :
    PotentialTerm}
    (h5 : S5' ⊆ S5) (h10 : S10' ⊆ S10) :
    minimallyAllowsTermsOfFinset S5' S10' T ⊆ minimallyAllowsTermsOfFinset S5 S10 T
```

lemma not_isPhenoConstrained_of_minimallyAllowsTermsOfFinset_topYukawa [source]

```
lemma not_isPhenoConstrained_of_minimallyAllowsTermsOfFinset_topYukawa
    {S5 S10 : Finset ℤ} {x : ChargeSpectrum ℤ}
    (hx : x ∈ minimallyAllowsTermsOfFinset S5 S10 topYukawa) :
    ¬ x.IsPhenoConstrained
```

2.25 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.MinimalSuperSet

PhysLean/Particles/SuperSymmetry/SU5/ChargeSpectrum/MinimalSuperSet.lean on GitHub.

def minimalSuperSet [source]

Given a collection of charges `x` in `ofFinset S5 S10`, the minimimal charges `y` in `ofFinset S5 S10` which are a super sets of `x`.

```
def minimalSuperSet (S5 S10 : Finset ℤ) (x : ChargeSpectrum ℤ) : Finset (ChargeSpectrum ℤ)
```

lemma self_subset_mem_minimalSuperSet [source]

```
lemma self_subset_mem_minimalSuperSet (S5 S10 : Finset ℤ) (x y : ChargeSpectrum ℤ)
    (hy : y ∈ minimalSuperSet S5 S10 x) : x ⊆ y
```

lemma self_not_mem_minimalSuperSet [source]

```
@[simp]
lemma self_not_mem_minimalSuperSet (S5 S10 : Finset ℤ) (x : ChargeSpectrum ℤ) :
    x ∉ minimalSuperSet S5 S10 x
```

lemma self_neq_mem_minimalSuperSet [source]

```
lemma self_neq_mem_minimalSuperSet (S5 S10 : Finset ℤ) (x y : ChargeSpectrum ℤ)
    (hy : y ∈ minimalSuperSet S5 S10 x) : x ≠ y
```

lemma card_of_mem_minimalSuperSet [source]

```
lemma card_of_mem_minimalSuperSet {S5 S10 : Finset ℤ} {x : ChargeSpectrum ℤ}
    (y : ChargeSpectrum ℤ) (hy : y ∈ minimalSuperSet S5 S10 x) :
    card y = card x + 1
```

lemma insert_Q5_mem_minimalSuperSet [source]

```
lemma insert_Q5_mem_minimalSuperSet {S5 S10 : Finset ℤ} {x : ChargeSpectrum ℤ}
    (z : ℤ) (hz : z ∈ S5) (hznot : z ∉ x.Q5) :
    ⟨x.qHd, x.qHu, insert z x.Q5, x.Q10⟩ ∈ minimalSuperSet S5 S10 x
```

lemma insert_Q10_mem_minimalSuperSet [source]

```
lemma insert_Q10_mem_minimalSuperSet {S5 S10 : Finset ℤ} {x : ChargeSpectrum ℤ}
    (z : ℤ) (hz : z ∈ S10) (hznot : z ∉ x.Q10) :
    ⟨x.qHd, x.qHu, x.Q5, insert z x.Q10⟩ ∈ minimalSuperSet S5 S10 x
```

lemma some_qHd_mem_minimalSuperSet_of_none [source]

```
lemma some_qHd_mem_minimalSuperSet_of_none {S5 S10 : Finset ℤ}
    {x2 : Option ℤ × Finset ℤ × Finset ℤ} (z : ℤ) (hz : z ∈ S5) :
    ⟨some z, x2.1, x2.2.1, x2.2.2⟩ ∈ minimalSuperSet S5 S10 ⟨none, x2.1, x2.2.1, x2.2.2⟩
```

lemma some_qHu_mem_minimalSuperSet_of_none [source]

```
lemma some_qHu_mem_minimalSuperSet_of_none {S5 S10 : Finset ℤ}
    {x1 : Option ℤ} {x2 : Finset ℤ × Finset ℤ} (z : ℤ) (hz : z ∈ S5) :
    ⟨x1, some z, x2.1,x2.2⟩ ∈ minimalSuperSet S5 S10 ⟨x1, none, x2.1, x2.2⟩
```

lemma exists_minimalSuperSet [source]

```
lemma exists_minimalSuperSet (S5 S10 : Finset 𝒵) {x y : ChargeSpectrum 𝒵}
    (hy : y ∈ ofFinset S5 S10) (hsubset : x ⊆ y)
    (hxneqy : x ≠ y) : ∃ z ∈ minimalSuperSet S5 S10 x, z ⊆ y
```

lemma minimalSuperSet_induction_on_inductive [source]

```
lemma minimalSuperSet_induction_on_inductive {S5 S10 : Finset 𝒵}
    (p : ChargeSpectrum 𝒵 → Prop)
    (hp : (x : ChargeSpectrum 𝒵) → p x → ∀ y ∈ minimalSuperSet S5 S10 x, p y)
    (x : ChargeSpectrum 𝒵) (hbase : p x)
    (y : ChargeSpectrum 𝒵) (hy : y ∈ ofFinset S5 S10) (hsubset : x ⊆ y) :
    (n : ℕ) → (hn : n = y.card - x.card) → p y
  | 0, hn => by
    have hxy : x = y
```

lemma insert_filter_card_zero [source]

```
lemma insert_filter_card_zero
    (T : Multiset (ChargeSpectrum 𝒵)) (S5 S10 : Finset 𝒵)
    (p : ChargeSpectrum 𝒵 → Prop) [DecidablePred p]
    (hComplet : ∀ x ∈ T, IsComplete x)
    (h10 : ∀ q10 : S10, ((T.map fun x => ⟨x.qHd, x.qHu, x.Q5, insert q10.1 x.Q10⟩).filter
      fun y => (y ∉ T ∧ p y)) = ∅)
    (h5 : ∀ q5 : S5, ((T.map fun x => ⟨x.qHd, x.qHu, insert q5.1 x.Q5, x.Q10⟩).filter
      fun y => (y ∉ T ∧ p y)) = ∅) :
    ∀ x ∈ T, ∀ y ∈ minimalSuperSet S5 S10 x, y ∉ T → ¬ p y
```

lemma subset_insert_filter_card_zero_inductive [source]

```
lemma subset_insert_filter_card_zero_inductive
    (T : Multiset (ChargeSpectrum 𝒵))
    (S5 S10 : Finset 𝒵)
    (p : ChargeSpectrum 𝒵 → Prop) [DecidablePred p]
    (hnotSubset : ∀ (x y : ChargeSpectrum 𝒵), x ⊆ y → ¬ p x → ¬ p y)
    (hComplet : ∀ x ∈ T, IsComplete x)
    (x : ChargeSpectrum 𝒵)
    (hx : x ∈ T) (y : ChargeSpectrum 𝒵) (hsubset : x ⊆ y)
    (hy : y ∈ ofFinset S5 S10)
    (h10 : ∀ q10 : S10, ((T.map fun x => ⟨x.qHd, x.qHu, x.Q5, insert q10.1 x.Q10⟩).filter
      fun y => (y ∉ T ∧ p y)) = ∅)
    (h5 : ∀ q5 : S5, ((T.map fun x => ⟨x.qHd, x.qHu, insert q5.1 x.Q5, x.Q10⟩).filter
      fun y => (y ∉ T ∧ p y)) = ∅) :
    (n : ℕ) → (hn : n = y.card - x.card) → y ∉ T → ¬ p y
  | 0, hn, hnot_in_T => by
    have hxy : x = y
```

lemma subset_insert_filter_card_zero [source]

For a proposition `p` *if* `(T.uniqueMap4 (insert q10.1)).toMultiset.filter p` *and* `(T.uniqueMap3 (insert q5.1)).toMultiset.filter p` *for all* `q5 ∈ S5` *and* `q10 ∈ S10` *then if* `x ∈ T` *and* `x ⊆ y` *if* `y ∉ T` *then* `¬ p y`*. This assumes that all charges in* `T` *are complete, and that* `p` *satisfies* `x ⊆ y → ¬ p x → ¬ p y`*.*

```
lemma subset_insert_filter_card_zero
    (T : Multiset (ChargeSpectrum ℤ))
    (S5 S10 : Finset ℤ)
    (p : ChargeSpectrum ℤ → Prop) [DecidablePred p]
    (hnotSubset : ∀ (x y : ChargeSpectrum ℤ), x ⊆ y → ¬ p x → ¬ p y)
    (hComplet : ∀ x ∈ T, IsComplete x)
    (x : ChargeSpectrum ℤ)
    (hx : x ∈ T) (y : ChargeSpectrum ℤ) (hsubset : x ⊆ y)
    (hy : y ∈ ofFinset S5 S10)
    (h10 : ∀ q10 : S10, ((T.map fun x => (x.qHd, x.qHu, x.Q5, insert q10.1 x.Q10)).filter
      fun y => (y ∉ T ∧ p y)) = ∅)
    (h5 : ∀ q5 : S5, ((T.map fun x => (x.qHd, x.qHu, insert q5.1 x.Q5, x.Q10)).filter
      fun y => (y ∉ T ∧ p y)) = ∅) :
      y ∉ T → ¬ p y
```

2.26 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.OfFieldLabel

PhysLean/Particles/SuperSymmetry/SU5/ChargeSpectrum/OfFieldLabel.lean on GitHub.

def ofFieldLabel [source]

Given an `x : Charges`, the charges associated with a given `FieldLabel`.

```
def ofFieldLabel (x : ChargeSpectrum ℤ) : FieldLabel → Finset ℤ
  | .fiveBarHd => x.qHd.toFinset
  | .fiveBarHu => x.qHu.toFinset
  | .fiveBarMatter => x.Q5
  | .tenMatter => x.Q10
  | .fiveHd => x.qHd.toFinset.map ⟨Neg.neg, neg_injective⟩
  | .fiveHu => x.qHu.toFinset.map ⟨Neg.neg, neg_injective⟩
  | .fiveMatter => x.Q5.map ⟨Neg.neg, neg_injective⟩
```

lemma ofFieldLabel_empty [source]

`ofFieldLabel ∅ F` is empty for any field label `F`.

```
@[simp]
lemma ofFieldLabel_empty (F : FieldLabel) :
    ofFieldLabel (∅ : ChargeSpectrum ℤ) F = ∅
```

lemma ofFieldLabel_mono [source]

The function `ofFieldLabel` is monotone in the charge spectrum.

```
lemma ofFieldLabel_mono {x y : ChargeSpectrum ℤ} (h : x ⊆ y) (F : FieldLabel) :
    x.ofFieldLabel F ⊆ y.ofFieldLabel F
```

lemma mem_ofFieldLabel_fiveHd [source]

```
@[simp]
lemma mem_ofFieldLabel_fiveHd (x : ℤ) (y : ChargeSpectrum ℤ) :
    x ∈ y.ofFieldLabel FieldLabel.fiveHd ↔ -x ∈ y.ofFieldLabel .fiveBarHd
```

lemma mem_ofFieldLabel_fiveHu [source]

```
@[simp]
lemma mem_ofFieldLabel_fiveHu (x : ℤ) (y : ChargeSpectrum ℤ) :
    x ∈ y.ofFieldLabel FieldLabel.fiveHu ↔ -x ∈ y.ofFieldLabel .fiveBarHu
```

lemma mem_ofFieldLabel_fiveMatter [source]

```
@[simp]
lemma mem_ofFieldLabel_fiveMatter (x : ℤ) (y : ChargeSpectrum ℤ) :
    x ∈ y.ofFieldLabel FieldLabel.fiveMatter ↔ -x ∈ y.ofFieldLabel .fiveBarMatter
```

lemma ext_ofFieldLabel [source]

Two charges are equal if they are equal on all field labels.

```
lemma ext_ofFieldLabel {x y : ChargeSpectrum ℤ} (h : ∀ F, x.ofFieldLabel F = y.ofFieldLabel F)
    :
    x = y
```

2.27 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.OfPotentialTerm

PhysLean/Particles/SuperSymmetry/SU5/ChargeSpectrum/OfPotentialTerm.lean on GitHub.

def ofPotentialTerm [source]

Given a charges `x : Charges` associated to the representations, and a potential term `T`, the charges associated with instances of that potential term.

```
def ofPotentialTerm (x : ChargeSpectrum ℤ) (T : PotentialTerm) : Multiset ℤ
```

lemma ofPotentialTerm_mono [source]

```
lemma ofPotentialTerm_mono {x y : ChargeSpectrum ℤ} (h : x ⊆ y) (T : PotentialTerm) :
    x.ofPotentialTerm T ⊆ y.ofPotentialTerm T
```

lemma ofPotentialTerm_empty [source]

```
@[simp]
lemma ofPotentialTerm_empty (T : PotentialTerm) :
    ofPotentialTerm (∅ : ChargeSpectrum ℤ) T = ∅
```

def ofPotentialTerm' [source]

Given a charges `x : ChargeSpectrum` associated to the representations, and a potential term `T`, the charges associated with instances of that potential term.

This is a more explicit form of `PotentialTerm`, which has the benifit that it is quick with `decide`, but it is not defined based on more fundamental concepts, like `ofPotentialTerm` is.

```
def ofPotentialTerm' (y : ChargeSpectrum ℤ) (T : PotentialTerm) : Multiset ℤ
```

lemma ofPotentialTerm'_μ_finset [source]

```
lemma ofPotentialTerm'_μ_finset {x : ChargeSpectrum 𝒵} :
    x.ofPotentialTerm' μ =
    (x.qHd.toFinset.product <| x.qHu.toFinset).val.map (fun x => x.1 - x.2)
```

lemma ofPotentialTerm'_β_finset [source]

```
lemma ofPotentialTerm'_β_finset {x : ChargeSpectrum 𝒵} :
    x.ofPotentialTerm' β =
    (x.qHu.toFinset.product <| x.Q5).val.map (fun x => - x.1 + x.2)
```

lemma ofPotentialTerm'_W2_finset [source]

```
lemma ofPotentialTerm'_W2_finset {x : ChargeSpectrum 𝒵} :
    x.ofPotentialTerm' W2 = (x.qHd.toFinset.product <|
      x.Q10.product <| x.Q10.product <| x.Q10).val.map
    (fun x => x.1 + x.2.1 + x.2.2.1 + x.2.2.2)
```

lemma ofPotentialTerm'_W3_finset [source]

```
lemma ofPotentialTerm'_W3_finset {x : ChargeSpectrum 𝒵} :
    x.ofPotentialTerm' W3 = (x.qHu.toFinset.product <| x.Q5.product <| x.Q5).val.map
    (fun x => -x.1 - x.1 + x.2.1 + x.2.2)
```

lemma ofPotentialTerm'_W4_finset [source]

```
lemma ofPotentialTerm'_W4_finset {x : ChargeSpectrum 𝒵} :
    x.ofPotentialTerm' W4 = (x.qHd.toFinset.product <|
      x.qHu.toFinset.product <| x.Q5).val.map
    (fun x => x.1 - x.2.1 - x.2.1 + x.2.2)
```

lemma ofPotentialTerm'_K2_finset [source]

```
lemma ofPotentialTerm'_K2_finset {x : ChargeSpectrum 𝒵} :
    x.ofPotentialTerm' K2 = (x.qHd.toFinset.product <|
      x.qHu.toFinset.product <| x.Q10).val.map
    (fun x => x.1 + x.2.1 + x.2.2)
```

lemma ofPotentialTerm'_topYukawa_finset [source]

```
lemma ofPotentialTerm'_topYukawa_finset {x : ChargeSpectrum 𝒵} :
    x.ofPotentialTerm' topYukawa = (x.qHu.toFinset.product <|
      x.Q10.product <| x.Q10).val.map
    (fun x => -x.1 + x.2.1 + x.2.2)
```

lemma ofPotentialTerm'_bottomYukawa_finset [source]

```
lemma ofPotentialTerm'_bottomYukawa_finset {x : ChargeSpectrum 𝒵} :
    x.ofPotentialTerm' bottomYukawa = (x.1.toFinset.product <|
      x.Q5.product <| x.Q10).val.map
    (fun x => x.1 + x.2.1 + x.2.2)
```

lemma ofPotentialTerm'_empty [source]

```
@[simp]
lemma ofPotentialTerm'_empty (T : PotentialTerm) :
    ofPotentialTerm' (∅ : ChargeSpectrum ℤ) T = ∅
```

lemma ofPotentialTerm_subset_ofPotentialTerm' [source]

```
lemma ofPotentialTerm_subset_ofPotentialTerm' {x : ChargeSpectrum ℤ} (T : PotentialTerm) :
    x.ofPotentialTerm T ⊆ x.ofPotentialTerm' T
```

lemma ofPotentialTerm'_subset_ofPotentialTerm [source]

```
lemma ofPotentialTerm'_subset_ofPotentialTerm [DecidableEq ℤ]
    {x : ChargeSpectrum ℤ} (T : PotentialTerm) :
    x.ofPotentialTerm' T ⊆ x.ofPotentialTerm T
```

lemma mem_ofPotentialTerm_iff_mem_ofPotentialTerm [source]

```
lemma mem_ofPotentialTerm_iff_mem_ofPotentialTerm [DecidableEq ℤ]
    {T : PotentialTerm} {n : ℤ} {y : ChargeSpectrum ℤ} :
    n ∈ y.ofPotentialTerm T ↔ n ∈ y.ofPotentialTerm' T
```

lemma ofPotentialTerm'_mono [source]

```
lemma ofPotentialTerm'_mono [DecidableEq ℤ] {x y : ChargeSpectrum ℤ}
    (h : x ⊆ y) (T : PotentialTerm) :
    x.ofPotentialTerm' T ⊆ y.ofPotentialTerm' T
```

2.28 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.PhenoClosed

PhysLean/Particles/SuperSymmetry/SU5/ChargeSpectrum/PhenoClosed.lean on GitHub.

def IsPhenoClosedQ5 [source]

> *The proposition that for multiset set of charges `charges`, adding individual elements of `S5` to the `Q5` charges of elements of `charges` again leads to an element in `charges` or a charge which is phenomenologically constrained, or regenerates dangerous couplings with one singlet insertion.*

```
def IsPhenoClosedQ5 (S5 : Finset ℤ) (charges : Multiset (ChargeSpectrum ℤ)) : Prop
```

lemma isPhenClosedQ5_of_isPhenoConstrainedQ5 [source]

```
lemma isPhenClosedQ5_of_isPhenoConstrainedQ5 {S5 : Finset ℤ} {charges : Multiset (
    ChargeSpectrum ℤ)}
    (h : ∀ q5 ∈ S5, ∀ x ∈ charges,
      let y : ChargeSpectrum ℤ := ⟨x.qHd, x.qHu, insert q5 x.Q5, x.Q10⟩
      IsPhenoConstrainedQ5 x q5 ∨ y ∈ charges ∨ YukawaGeneratesDangerousAtLevel y 1) :
    IsPhenoClosedQ5 S5 charges
```

def IsPhenoClosedQ10 [source]

The proposition that for multiset set of charges `charges`, adding individual elements of `S10` to the `Q10` charges of elements of `charges` again leads to an element in `charges` or a charge which is phenomenologically constrained, or regenerates dangerous couplings with one singlet insertion.

```
def IsPhenoClosedQ10 (S10 : Finset ℤ) (charges : Multiset (ChargeSpectrum ℤ)) : Prop
```

lemma isPhenClosedQ10_of_isPhenoConstrainedQ10 [source]

```
lemma isPhenClosedQ10_of_isPhenoConstrainedQ10 {S10 : Finset ℤ}
    {charges : Multiset (ChargeSpectrum ℤ)}
    (h : ∀ q10 ∈ S10, ∀ x ∈ charges,
      let y : ChargeSpectrum ℤ := ⟨x.qHd, x.qHu, x.Q5, insert q10 x.Q10⟩
      IsPhenoConstrainedQ10 x q10 ∨ y ∈ charges ∨ YukawaGeneratesDangerousAtLevel y 1) :
    IsPhenoClosedQ10 S10 charges
```

def ContainsPhenoCompletionsOfMinimallyAllows [source]

The proposition that for multiset set of charges `charges` contains all viable completions of charges which allow the top Yukawa, given allowed values of `5`d and `10`d charges `S5` and `S10`.

```
def ContainsPhenoCompletionsOfMinimallyAllows (S5 S10 : Finset ℤ)
    (charges : Multiset (ChargeSpectrum ℤ)) : Prop
```

lemma containsPhenoCompletionsOfMinimallyAllows_iff_completionsTopYukawa [source]

```
lemma containsPhenoCompletionsOfMinimallyAllows_iff_completionsTopYukawa {S5 S10 : Finset ℤ}
    {charges : Multiset (ChargeSpectrum ℤ)} :
    ContainsPhenoCompletionsOfMinimallyAllows S5 S10 charges ↔
    ∀ x ∈ (minimallyAllowsTermsOfFinset S5 S10 topYukawa),
    ∀ y ∈ completionsTopYukawa S5 x, ¬ y.IsPhenoConstrained
      ∧ ¬ y.YukawaGeneratesDangerousAtLevel 1 → y ∈ charges
```

instance Decidable (ContainsPhenoCompletionsOfMinimallyAllows S5 S10 charges) [source]

```
instance [DecidableEq ℤ] {S5 S10 : Finset ℤ} {charges : Multiset (ChargeSpectrum ℤ)} :
    Decidable (ContainsPhenoCompletionsOfMinimallyAllows S5 S10 charges)
```

lemma containsPhenoCompletionsOfMinimallyAllows_of_subset [source]

```
lemma containsPhenoCompletionsOfMinimallyAllows_of_subset {S5 S10 : Finset ℤ}
    {charges charges' : Multiset (ChargeSpectrum ℤ)}
    (h' : ContainsPhenoCompletionsOfMinimallyAllows S5 S10 charges)
    (h : ∀ x ∈ charges, x ∈ charges') :
    ContainsPhenoCompletionsOfMinimallyAllows S5 S10 charges'
```

def completeMinSubset [source]

For a given `S5 S10 : Finset ℤ`, the minimal multiset of charges which satifies the condition `ContainsPhenoCompletionsOfMinimallyAllows`. That is to say, every

multiset of charges which satifies `ContainsPhenoCompletionsOfMinimallyAllows` has `completeMinSubset` as a subset.

```
def completeMinSubset (S5 S10 : Finset ℤ) : Multiset (ChargeSpectrum ℤ)
```

lemma completeMinSubset_nodup [source]

```
lemma completeMinSubset_nodup {S5 S10 : Finset ℤ} :
    (completeMinSubset S5 S10).Nodup
```

lemma completeMinSubset_subset_iff_containsPhenoCompletionsOfMinimallyAllows [source]

```
lemma completeMinSubset_subset_iff_containsPhenoCompletionsOfMinimallyAllows
    (S5 S10 : Finset ℤ) (charges : Multiset (ChargeSpectrum ℤ)) :
    completeMinSubset S5 S10 ⊆ charges ↔
    ContainsPhenoCompletionsOfMinimallyAllows S5 S10 charges
```

lemma completeMinSubset_containsPhenoCompletionsOfMinimallyAllows [source]

```
lemma completeMinSubset_containsPhenoCompletionsOfMinimallyAllows (S5 S10 : Finset ℤ) :
    ContainsPhenoCompletionsOfMinimallyAllows S5 S10 (completeMinSubset S5 S10)
```

lemma completeness_of_isPhenoClosedQ5_isPhenoClosedQ10 [source]

```
lemma completeness_of_isPhenoClosedQ5_isPhenoClosedQ10
    {S5 S10 : Finset ℤ} {charges : Multiset (ChargeSpectrum ℤ)}
    (charges_topYukawa : ∀ x ∈ charges, x.AllowsTerm .topYukawa)
    (charges_not_isPhenoConstrained : ∀ x ∈ charges, ¬ x.IsPhenoConstrained)
    (charges_yukawa : ∀ x ∈ charges, ¬ x.YukawaGeneratesDangerousAtLevel 1)
    (charges_complete : ∀ x ∈ charges, x.IsComplete)
    (charges_isPhenoClosedQ5 : IsPhenoClosedQ5 S5 charges)
    (charges_isPhenoClosedQ10 : IsPhenoClosedQ10 S10 charges)
    (charges_exist : ContainsPhenoCompletionsOfMinimallyAllows S5 S10 charges)
    {x : ChargeSpectrum ℤ} (hsub : x ∈ ofFinset S5 S10) :
    x ∈ charges ↔ AllowsTerm x .topYukawa ∧
    ¬ IsPhenoConstrained x ∧ ¬ YukawaGeneratesDangerousAtLevel x 1 ∧ IsComplete x
```

def viableChargesMultiset [source]

All charges, for a given `S5 S10 : Finset ℤ`, which permit a top Yukawa coupling, are not phenomenologically constrained, and do not regenerate dangerous couplings with one insertion of a Yukawa coupling.

This is the unique multiset without duplicates which satifies: `completeness_of_isPhenoClosedQ5_isPhenoClo`

Note this is fast for evaluation, but to slow with `decide`.

```
unsafe def viableChargesMultiset (S5 S10 : Finset ℤ) :
    Multiset (ChargeSpectrum ℤ)
```

2.29 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.PhenoConstrained

PhysLean/Particles/SuperSymmetry/SU5/ChargeSpectrum/PhenoConstrained.lean on GitHub.

def IsPhenoConstrained [source]

A charge is pheno-constrained if it leads to the presence of any term causing proton decay `{W1, Λ, W2, K1}` *or R-parity violation* `{β, Λ, W2, W4, K1, K2}`.

```
def IsPhenoConstrained (x : ChargeSpectrum ℤ) : Prop
```

instance decidableIsPhenoConstrained [source]

```
instance decidableIsPhenoConstrained [DecidableEq ℤ] (x : ChargeSpectrum ℤ) :
    Decidable x.IsPhenoConstrained
```

lemma not_isPhenoConstrained_empty [source]

```
@[simp]
lemma not_isPhenoConstrained_empty :
    ¬ IsPhenoConstrained (∅ : ChargeSpectrum ℤ)
```

lemma isPhenoConstrained_mono [source]

```
lemma isPhenoConstrained_mono {x y : ChargeSpectrum ℤ} (h : x ⊆ y)
    (hx : x.IsPhenoConstrained) : y.IsPhenoConstrained
```

def phenoConstrainingChargesSP [source]

The collection of charges of super-potential terms leading to a pheno-constrained model.

```
def phenoConstrainingChargesSP (x : ChargeSpectrum ℤ) : Multiset ℤ
```

lemma phenoConstrainingChargesSP_empty [source]

```
@[simp]
lemma phenoConstrainingChargesSP_empty :
    phenoConstrainingChargesSP (∅ : ChargeSpectrum ℤ) = ∅
```

lemma phenoConstrainingChargesSP_mono [source]

```
lemma phenoConstrainingChargesSP_mono [DecidableEq ℤ] {x y : ChargeSpectrum ℤ} (h : x ⊆ y) :
    x.phenoConstrainingChargesSP ⊆ y.phenoConstrainingChargesSP
```

def IsPhenoConstrainedQ5 [source]

The proposition which is true if the addition of a charge `q5` *to a set of charegs* `x` *leads* `x` *to being phenomenologically constrained.*

```
def IsPhenoConstrainedQ5 [DecidableEq ℤ] (x : ChargeSpectrum ℤ) (q5 : ℤ) : Prop
```

lemma isPhenoConstrainedQ5_iff [source]

```
lemma isPhenoConstrainedQ5_iff [DecidableEq ℤ] (x : ChargeSpectrum ℤ) (q5 : ℤ) :
    x.IsPhenoConstrainedQ5 q5 ↔
    x.AllowsTermQ5 q5 β ∨ x.AllowsTermQ5 q5 Λ ∨ x.AllowsTermQ5 q5 W4 ∨
```

```
    x.AllowsTermQ5 q5 K1 ∨ x.AllowsTermQ5 q5 W1
```

instance decidableIsPhenoConstrainedQ5 [source]

```
instance decidableIsPhenoConstrainedQ5 [DecidableEq ℤ] (x : ChargeSpectrum ℤ) (q5 : ℤ) :
    Decidable (x.IsPhenoConstrainedQ5 q5)
```

lemma isPhenoConstrained_insertQ5_iff_isPhenoConstrainedQ5 [source]

```
lemma isPhenoConstrained_insertQ5_iff_isPhenoConstrainedQ5 [DecidableEq ℤ] {qHd qHu : Option
    ℤ}
    {Q5 Q10: Finset ℤ} {q5 : ℤ} :
    IsPhenoConstrained ⟨qHd, qHu, insert q5 Q5, Q10⟩↔
    IsPhenoConstrainedQ5 ⟨qHd, qHu, Q5, Q10⟩ q5 ∨
    IsPhenoConstrained ⟨qHd, qHu, Q5, Q10⟩
```

def IsPhenoConstrainedQ10 [source]

The proposition which is true if the addition of a charge `q10` to a set of charegs `x` leads `x` to being phenomenologically constrained.

```
def IsPhenoConstrainedQ10 [DecidableEq ℤ] (x : ChargeSpectrum ℤ) (q10 : ℤ) : Prop
```

lemma isPhenoConstrainedQ10_iff [source]

```
lemma isPhenoConstrainedQ10_iff [DecidableEq ℤ] (x : ChargeSpectrum ℤ) (q10 : ℤ) :
    x.IsPhenoConstrainedQ10 q10 ↔ x.AllowsTermQ10 q10 ∧ x.AllowsTermQ10 q10 W2 ∨
    x.AllowsTermQ10 q10 K1 ∨ x.AllowsTermQ10 q10 K2 ∨ x.AllowsTermQ10 q10 W1
```

instance decidableIsPhenoConstrainedQ10 [source]

```
instance decidableIsPhenoConstrainedQ10 [DecidableEq ℤ] (x : ChargeSpectrum ℤ) (q10 : ℤ) :
    Decidable (x.IsPhenoConstrainedQ10 q10)
```

lemma isPhenoConstrained_insertQ10_iff_isPhenoConstrainedQ10 [source]

```
lemma isPhenoConstrained_insertQ10_iff_isPhenoConstrainedQ10 [DecidableEq ℤ] {qHd qHu : Option
     ℤ}
    {Q5 Q10: Finset ℤ} {q10 : ℤ} :
    IsPhenoConstrained ⟨qHd, qHu, Q5, insert q10 Q10⟩ ↔
    IsPhenoConstrainedQ10 ⟨qHd, qHu, Q5, Q10⟩ q10 ∨
    IsPhenoConstrained ⟨qHd, qHu, Q5, Q10⟩
```

2.30 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.Yukawa

PhysLean/Particles/SuperSymmetry/SU5/ChargeSpectrum/Yukawa.lean on GitHub.

def ofYukawaTerms [source]

The collection of charges associated with Yukawa terms. Correspondingly, the (negative) of the charges of the singlets needed to regenerate all Yukawa terms in the potential.

```
def ofYukawaTerms (x : ChargeSpectrum 𝒵) : Multiset 𝒵
```

lemma ofYukawaTerms_subset_of_subset [source]

```
lemma ofYukawaTerms_subset_of_subset [DecidableEq 𝒵] {x y : ChargeSpectrum 𝒵} (h : x ⊆ y) :
    x.ofYukawaTerms ⊆ y.ofYukawaTerms
```

def ofYukawaTermsNSum [source]

The charges of those terms which can be regenerated with up-to `n` insertions of singlets needed to regenerate the Yukawa terms. Equivalently, the sum of up-to `n` integers each corresponding to a charge of the Yukawa terms.

```
def ofYukawaTermsNSum (x : ChargeSpectrum 𝒵) : ℕ → Multiset 𝒵
  | 0 => {0}
  | n + 1 => x.ofYukawaTermsNSum n + (x.ofYukawaTermsNSum n).bind fun sSum =>
    (x.ofYukawaTerms.map fun s => sSum + s)
```

lemma ofYukawaTermsNSum_subset_of_subset [source]

```
lemma ofYukawaTermsNSum_subset_of_subset [DecidableEq 𝒵] {x y : ChargeSpectrum 𝒵}
    (h : x ⊆ y) (n : ℕ) :
    x.ofYukawaTermsNSum n ⊆ y.ofYukawaTermsNSum n
```

def YukawaGeneratesDangerousAtLevel [source]

For charges `x : Charges`, the proposition which states that the singlets needed to regenerate the Yukawa couplings regnerate a dangerous coupling (in the superpotential) with up-to `n` insertions of the scalars.

Note: If defined as $(x.ofYukawaTermsNSum\ n).toFinset \cap x.phenoConstrainingChargesSP.toFinset \neq \emptyset$ the exicution time is greatley increased.

```
def YukawaGeneratesDangerousAtLevel (x : ChargeSpectrum 𝒵) (n : ℕ) : Prop
```

instance Decidable (YukawaGeneratesDangerousAtLevel x n) [source]

```
instance (x : ChargeSpectrum 𝒵) (n : ℕ) : Decidable (YukawaGeneratesDangerousAtLevel x n)
```

lemma YukawaGeneratesDangerousAtLevel_iff_inter [source]

```
lemma YukawaGeneratesDangerousAtLevel_iff_inter {x : ChargeSpectrum 𝒵} {n : ℕ} :
    YukawaGeneratesDangerousAtLevel x n ↔
    (x.ofYukawaTermsNSum n) ∩ x.phenoConstrainingChargesSP ≠ ∅
```

lemma yukawaGeneratesDangerousAtLevel_iff_toFinset [source]

```
lemma yukawaGeneratesDangerousAtLevel_iff_toFinset (x : ChargeSpectrum 𝒵) (n : ℕ) :
    x.YukawaGeneratesDangerousAtLevel n ↔
    (x.ofYukawaTermsNSum n).toFinset ∩ x.phenoConstrainingChargesSP.toFinset ≠ ∅
```

lemma not_yukawaGeneratesDangerousAtLevel_of_empty [source]

```
@[simp]
lemma not_yukawaGeneratesDangerousAtLevel_of_empty (n : ℕ) :
    ¬ YukawaGeneratesDangerousAtLevel (∅ : ChargeSpectrum 𝒵) n
```

lemma yukawaGeneratesDangerousAtLevel_of_subset [source]

```
lemma yukawaGeneratesDangerousAtLevel_of_subset {x y : ChargeSpectrum 𝒵} {n : ℕ} (h : x ⊆ y)
    (hx : x.YukawaGeneratesDangerousAtLevel n) :
    y.YukawaGeneratesDangerousAtLevel n
```

lemma yukawaGeneratesDangerousAtLevel_succ [source]

```
lemma yukawaGeneratesDangerousAtLevel_succ {x : ChargeSpectrum 𝒵} {n : ℕ}
    (hx : x.YukawaGeneratesDangerousAtLevel n) :
    x.YukawaGeneratesDangerousAtLevel (n + 1)
```

lemma yukawaGeneratesDangerousAtLevel_add_of_left [source]

```
lemma yukawaGeneratesDangerousAtLevel_add_of_left {x : ChargeSpectrum 𝒵} {n k : ℕ}
    (hx : x.YukawaGeneratesDangerousAtLevel n) :
    x.YukawaGeneratesDangerousAtLevel (n + k)
```

lemma yukawaGeneratesDangerousAtLevel_of_le [source]

```
lemma yukawaGeneratesDangerousAtLevel_of_le {x : ChargeSpectrum 𝒵} {n m : ℕ}
    (h : n ≤ m) (hx : x.YukawaGeneratesDangerousAtLevel n) :
    x.YukawaGeneratesDangerousAtLevel m
```

2.31 PhysLean.Particles.SuperSymmetry.SU5.ChargeSpectrum.ZMod

PhysLean/Particles/SuperSymmetry/SU5/ChargeSpectrum/ZMod.lean on GitHub.

def ZModCharges [source]

The finite set of `ZMod n` valued charges which are complete, not pheno-constrained and don't regenerate dangerous couplings with the Yukawa term up-to 4-inserstions of singlets.

```
def ZModCharges (n : ℕ) [NeZero n] : Finset (ChargeSpectrum (ZMod n))
```

lemma ZModCharges_one_eq [source]

This lemma corresponds to the statement that there are no choices of $\mathbb{Z}_1$ representations which give a phenomenologically viable theory.

```
lemma ZModCharges_one_eq : ZModCharges 1 = ∅
```

lemma ZModCharges_two_eq [source]

This lemma corresponds to the statement that there are no choices of $\mathbb{Z}_2$ representations which give a phenomenologically viable theory.

```
lemma ZModCharges_two_eq : ZModCharges 2 = ∅
```

lemma ZModCharges_three_eq [source]

This lemma corresponds to the statement that there are no choices of $\mathbb{Z}_3$ representations which give a phenomenologically viable theory.

```
@[pseudo]
lemma ZModCharges_three_eq : ZModCharges 3 = ∅
```

lemma ZModCharges_four_eq [source]

```
@[pseudo]
lemma ZModCharges_four_eq : ZModCharges 4 = {⟨some 0, some 2, {1}, {3}⟩,
    ⟨some 0, some 2, {3}, {1}⟩, ⟨some 1, some 2, {0}, {3}⟩, ⟨some 3, some 2, {0}, {1}⟩}
```

lemma ZModCharges_five_eq [source]

This lemma corresponds to the statement that there are no choices of $\mathbb{Z}_5$ representations which give a phenomenologically viable theory.

```
@[pseudo]
lemma ZModCharges_five_eq : ZModCharges 5 = ∅
```

lemma ZModCharges_six_eq [source]

```
@[pseudo]
lemma ZModCharges_six_eq : ZModCharges 6 = {⟨some 0, some 2, {5}, {1}⟩,
    ⟨some 0, some 4, {1}, {5}⟩, ⟨some 1, some 0, {2}, {3}⟩, ⟨some 1, some 2, {4}, {1}⟩,
    ⟨some 1, some 4, {0}, {5}⟩, ⟨some 1, some 4, {3}, {2}⟩, ⟨some 2, some 0, {1}, {3}⟩,
    ⟨some 2, some 4, {5}, {5}⟩, ⟨some 3, some 2, {5}, {4}⟩, ⟨some 3, some 4, {1}, {2}⟩,
    ⟨some 4, some 0, {5}, {3}⟩, ⟨some 4, some 2, {1}, {1}⟩, ⟨some 5, some 0, {4}, {3}⟩,
    ⟨some 5, some 2, {0}, {1}⟩, ⟨some 5, some 2, {3}, {4}⟩, ⟨some 5, some 4, {2}, {5}⟩}
```

def ZModZModCharges [source]

The finite set of `ZMod n × ZMod m` valued charges which are complete, not pheno-constrained and don't regenerate dangerous couplings with the Yukawa term up-to 4-inserstions of singlets.

```
def ZModZModCharges (n m : ℕ) [NeZero n] [NeZero m] : Finset (ChargeSpectrum (ZMod n × ZMod m))
```

2.32 PhysLean.QFT.AnomalyCancellation.Basic

PhysLean/QFT/AnomalyCancellation/Basic.lean on GitHub.

lemma linSolsIncl_injective [source]

```
@[sorryful]
lemma linSolsIncl_injective (χ : ACCSystemLinear) :
    Function.Injective χ.linSolsIncl
```

lemma quadSolsInclLinSols_injective [source]

```
@[sorryful]
lemma quadSolsInclLinSols_injective (χ : ACCSystemQuad) :
    Function.Injective χ.quadSolsInclLinSols
```

lemma quadSolsIncl_injective [source]

```
@[sorryful]
lemma quadSolsIncl_injective (χ : ACCSystemQuad) :
    Function.Injective χ.quadSolsIncl
```

lemma solsInclQuadSols_injective [source]

```
@[sorryful]
lemma solsInclQuadSols_injective (χ : ACCSystem) :
    Function.Injective χ.solsInclQuadSols
```

lemma solsInclLinSols_injective [source]

```
@[sorryful]
lemma solsInclLinSols_injective (χ : ACCSystem) :
    Function.Injective χ.solsInclLinSols
```

lemma solsIncl_injective [source]

```
@[sorryful]
lemma solsIncl_injective (χ : ACCSystem) :
    Function.Injective χ.solsIncl
```

2.33 PhysLean.QuantumMechanics.OneDimension.ReflectionlessPotential.Basic

PhysLean/QuantumMechanics/OneDimension/ReflectionlessPotential/Basic.lean on GitHub.

def mulByTemperateGrowth [source]

Pointwise multiplication by a function of temperate growth

```
noncomputable def mulByTemperateGrowth {g : ℝ → ℂ} (hg : g.HasTemperateGrowth) :
    𝓢(ℝ, ℂ) →L[ℂ] 𝓢(ℝ, ℂ)
```

lemma scaled_tanh_hasTemperateGrowth [source]

```
lemma scaled_tanh_hasTemperateGrowth (κ : ℝ) :
    Function.HasTemperateGrowth (fun x => (Real.tanh (κ * x)))
```

lemma complex_embedding_of_temperate_growth [source]

This is a helper lemma to show that the embedding of a real functio with temperate growth in ℂ also has temperate growth

```
private lemma complex_embedding_of_temperate_growth (f : ℝ → ℝ)
    (h : Function.HasTemperateGrowth f) : Function.HasTemperateGrowth (fun x => (f x : ℂ))
```

lemma scaled_tanh_complex_hasTemperateGrowth [source]

```
lemma scaled_tanh_complex_hasTemperateGrowth (κ : ℝ) :
    Function.HasTemperateGrowth (fun x => (Real.tanh (κ * x) : ℂ))
```

def tanhOperatorSchwartz [source]

Define tanh(κ X) multiplication pointwise as a Schwartz map

```
noncomputable def tanhOperatorSchwartz (Q : ReflectionlessPotential) :
    𝓢(ℝ, ℂ) →L[ℂ] 𝓢(ℝ, ℂ)
```

def creationOperatorSchwartz [source]

creation operator defined as a Schwartz map

```
noncomputable def creationOperatorSchwartz (Q : ReflectionlessPotential) : 𝓢(ℝ, ℂ) →L[ℂ] 𝓢(ℝ,
    ℂ)
```

def annihilationOperatorSchwartz [source]

annihilation operator defined as a Schwartz map

```
noncomputable def annihilationOperatorSchwartz (Q : ReflectionlessPotential) :
  𝓢(ℝ, ℂ) →L[ℂ] 𝓢(ℝ, ℂ)
```

2.34 PhysLean.Relativity.LorentzGroup.Boosts.Basic

PhysLean/Relativity/LorentzGroup/Boosts/Basic.lean on GitHub.

lemma γ_det_not_zero [source]

```
@[simp]
lemma γ_det_not_zero (β : ℝ) (hβ : |β| < 1) : (1 - β^2) ≠ 0
```

2.35 PhysLean.Relativity.Tensors.Basic

PhysLean/Relativity/Tensors/Basic.lean on GitHub.

lemma actionT_neg [source]

```
lemma actionT_neg {g : G} {t : S.Tensor c} :
    g • (-t) = -(g • t)
```

2.36 PhysLean.Relativity.Tensors.Product

PhysLean/Relativity/Tensors/Product.lean on GitHub.

lemma prodT_basis [source]

```
lemma prodT_basis {n1 n2} {c : Fin n1 → C} {c1 : Fin n2 → C}
    (b : ComponentIdx c) (b1 : ComponentIdx (S := S) c1) :
    (basis c b).prodT (basis c1 b1) =
    (Pure.basisVector _ (b.prod b1)).toTensor
```

def tensorEquivProd [source]

The linear equivalence between `S.Tensor c ⊗[k] S.Tensor c1` and `S.Tensor (Sum.elim c c1 ∘ ⇑finSumFinEquiv.symm)`.

```
noncomputable def tensorEquivProd {n n2 : ℕ} {c : Fin n → C} {c1 : Fin n2 → C} :
    S.Tensor c ⊗[k] S.Tensor c1 ≃ₗ[k] S.Tensor (Sum.elim c c1 ∘ ⇑finSumFinEquiv.symm) where
```

lemma basis_prod_eq [source]

Rewriting basis for the product in terms of the tensor product basis.

```
lemma basis_prod_eq {n1 n2} {c : Fin n1 → C} {c1 : Fin n2 → C} :
    basis (S := S) (Sum.elim c c1 ∘ ⇑finSumFinEquiv.symm) =
    (((Tensor.basis (S := S) c).tensorProduct (Tensor.basis (S := S) c1)).reindex
    (ComponentIdx.splitEquiv.symm)).map tensorEquivProd
```

2.37 PhysLean.Relativity.Tensors.RealTensor.CoVector.Basic

PhysLean/Relativity/Tensors/RealTensor/CoVector/Basic.lean on GitHub.

def CoVector [source]

Real contravariant Lorentz vector.

```
def CoVector (d : ℕ := 3)
```

instance AddCommMonoid (CoVector d) [source]

```
instance {d} : AddCommMonoid (CoVector d)
```

instance Module ℝ (CoVector d) [source]

```
instance {d} : Module ℝ (CoVector d)
```

instance AddCommGroup (CoVector d) [source]

```
instance {d} : AddCommGroup (CoVector d)
```

instance FiniteDimensional ℝ (CoVector d) [source]

```
instance {d} : FiniteDimensional ℝ (CoVector d)
```

instance isNormedAddCommGroup [source]

```
instance isNormedAddCommGroup (d : ℕ) : NormedAddCommGroup (CoVector d)
```

instance isNormedSpace [source]

```
instance isNormedSpace (d : ℕ) :
    NormedSpace ℝ (CoVector d)
```

instance ChartedSpace (CoVector d) (CoVector d) [source]

The instance of a `ChartedSpace` on `Vector d`.

```
instance : ChartedSpace (CoVector d) (CoVector d)
```

instance CoeFun (CoVector d) (fun _ => Fin 1 ⊕ Fin d → ℝ) [source]

```
instance {d} : CoeFun (CoVector d) (fun _ => Fin 1 ⊕ Fin d → ℝ) where
```

lemma apply_smul [source]

```
@[simp]
lemma apply_smul {d : ℕ} (c : ℝ) (v : CoVector d) (i : Fin 1 ⊕ Fin d) :
    (c • v) i = c * v i
```

lemma apply_add [source]

```
@[simp]
lemma apply_add {d : ℕ} (v w : CoVector d) (i : Fin 1 ⊕ Fin d) :
    (v + w) i = v i + w i
```

lemma apply_sub [source]

```
@[simp]
lemma apply_sub {d : ℕ} (v w : CoVector d) (i : Fin 1 ⊕ Fin d) :
    (v - w) i = v i - w i
```

lemma neg_apply [source]

```
@[simp]
lemma neg_apply {d : ℕ} (v : CoVector d) (i : Fin 1 ⊕ Fin d) :
    (-v) i = - v i
```

lemma zero_apply [source]

```
@[simp]
lemma zero_apply {d : ℕ} (i : Fin 1 ⊕ Fin d) :
    (0 : CoVector d) i = 0
```

def indexEquiv [source]

The equivalence between the type of indices of a Lorentz vector and `Fin 1 ⊕ Fin d`.

```
def indexEquiv {d : ℕ} :
    ComponentIdx (S := (realLorentzTensor d)) ![Color.down] ≃ Fin 1 ⊕ Fin d
```

instance tensorial [source]

```
instance tensorial {d : ℕ} : Tensorial (realLorentzTensor d) ![.down] (CoVector d) where
```

lemma toTensor_symm_apply [source]

```
lemma toTensor_symm_apply {d : ℕ} (p : ℝT[d, .down]) :
    (toTensor (self := tensorial)).symm p =
    (Equiv.piCongrLeft' _ indexEquiv <|
    Finsupp.equivFunOnFinite <|
    (Tensor.basis (S := (realLorentzTensor d)) _).repr p)
```

lemma toTensor_symm_pure [source]

```
lemma toTensor_symm_pure {d : ℕ} (p : Pure (realLorentzTensor d) ![.down]) (i : Fin 1 ⊕ Fin d)
    :
    (toTensor (self := tensorial)).symm p.toTensor i =
    ((Lorentz.coBasisFin d).repr (p 0)) (indexEquiv.symm i 0)
```

def basis [source]

The basis on `Vector d` indexed by `Fin 1 ⊕ Fin d`.

```
def basis {d : ℕ} : Basis (Fin 1 ⊕ Fin d) ℝ (CoVector d)
```

lemma basis_apply [source]

```
@[simp]
lemma basis_apply {d : ℕ} (μ ν : Fin 1 ⊕ Fin d) :
    basis μ ν = if μ = ν then 1 else 0
```

lemma toTensor_symm_basis [source]

```
lemma toTensor_symm_basis {d : ℕ} (μ : Fin 1 ⊕ Fin d) :
    (toTensor (self := tensorial)).symm (Tensor.basis ![Color.down] (indexEquiv.symm μ)) =
    basis μ
```

lemma toTensor_basis_eq_tensor_basis [source]

```
lemma toTensor_basis_eq_tensor_basis {d : ℕ} (μ : Fin 1 ⊕ Fin d) :
    toTensor (basis μ) = Tensor.basis ![Color.down] (indexEquiv.symm μ)
```

lemma basis_eq_map_tensor_basis [source]

```
lemma basis_eq_map_tensor_basis : basis = ((Tensor.basis (S := realLorentzTensor) ![Color.down
    ]).map
    toTensor.symm).reindex indexEquiv
```

lemma tensor_basis_map_eq_basis_reindex [source]

```
lemma tensor_basis_map_eq_basis_reindex :
    (Tensor.basis (S := realLorentzTensor) ![Color.down]).map toTensor.symm =
    basis.reindex indexEquiv.symm
```

lemma tensor_basis_repr_toTensor_apply [source]

```
lemma tensor_basis_repr_toTensor_apply {d : ℕ} (p : CoVector d) (μ : ComponentIdx ![Color.down
    ]) :
    (Tensor.basis ![Color.down]).repr (toTensor p) μ =
    p (indexEquiv μ)
```

lemma basis_repr_apply [source]

```
lemma basis_repr_apply {d : ℕ} (p : CoVector d) (μ : Fin 1 ⊕ Fin d) :
    basis.repr p μ = p μ
```

lemma map_apply_eq_basis_mulVec [source]

```
lemma map_apply_eq_basis_mulVec {d : ℕ} (f : CoVector d →ₗ[ℝ] CoVector d) (p : CoVector d) :
    (f p) = (LinearMap.toMatrix basis basis) f *ᵥ p
```

lemma smul_eq_sum [source]

```
lemma smul_eq_sum {d : ℕ} (i : Fin 1 ⊕ Fin d) (Λ : LorentzGroup d) (p : CoVector d) :
    (Λ • p) i = ∑ j, Λ⁻¹.1 j i * p j
```

lemma smul_eq_mulVec [source]

```
lemma smul_eq_mulVec {d} (Λ : LorentzGroup d) (p : CoVector d) :
    Λ • p = (LorentzGroup.transpose Λ⁻¹).1 *ᵥ p
```

lemma smul_add [source]

```
lemma smul_add {d : ℕ} (Λ : LorentzGroup d) (p q : CoVector d) :
    Λ • (p + q) = Λ • p + Λ • q
```

lemma smul_sub [source]

```
@[simp]
lemma smul_sub {d : ℕ} (Λ : LorentzGroup d) (p q : CoVector d) :
    Λ • (p - q) = Λ • p - Λ • q
```

lemma smul_zero [source]

```
lemma smul_zero {d : ℕ} (Λ : LorentzGroup d) :
    Λ • (0 : CoVector d) = 0
```

lemma smul_neg [source]

```
lemma smul_neg {d : ℕ} (Λ : LorentzGroup d) (p : CoVector d) :
    Λ • (-p) = - (Λ • p)
```

def actionCLM [source]

The Lorentz action on vectors as a continuous linear map.

```
def actionCLM {d : ℕ} (Λ : LorentzGroup d) :
    CoVector d →L[ℝ] CoVector d
```

lemma actionCLM_apply [source]

```
lemma actionCLM_apply {d : ℕ} (Λ : LorentzGroup d) (p : CoVector d) :
    actionCLM Λ p = Λ • p
```

lemma smul_basis [source]

```
lemma smul_basis {d : ℕ} (Λ : LorentzGroup d) (μ : Fin 1 ⊕ Fin d) :
    Λ • basis μ = ∑ ν, Λ⁻¹.1 μ ν • basis ν
```

2.38 PhysLean.Relativity.Tensors.RealTensor.Vector.Basic

PhysLean/Relativity/Tensors/RealTensor/Vector/Basic.lean on GitHub.

lemma basis_eq_map_tensor_basis [source]

```
lemma basis_eq_map_tensor_basis : basis =
    ((Tensor.basis (S := realLorentzTensor) ![Color.up]).map toTensor.symm).reindex indexEquiv
```

lemma tensor_basis_map_eq_basis_reindex [source]

```
lemma tensor_basis_map_eq_basis_reindex :
    (Tensor.basis (S := realLorentzTensor) ![Color.up]).map toTensor.symm =
    basis.reindex indexEquiv.symm
```

def actionCLM [source]

The Lorentz action on vectors as a continuous linear map.

```
def actionCLM {d : ℕ} (Λ : LorentzGroup d) :
    Vector d →L[ℝ] Vector d
```

lemma actionCLM_apply [source]

```
lemma actionCLM_apply {d : ℕ} (Λ : LorentzGroup d) (p : Vector d) :
    actionCLM Λ p = Λ • p
```

lemma smul_basis [source]

```
lemma smul_basis {d : ℕ} (Λ : LorentzGroup d) (μ : Fin 1 ⊕ Fin d) :
    Λ • basis μ = ∑ ν, Λ.1 ν μ • basis ν
```

2.39 PhysLean.Relativity.Tensors.RealTensor.Vector.Pre.Basic

PhysLean/Relativity/Tensors/RealTensor/Vector/Pre/Basic.lean on GitHub.

lemma coBasisFin_repr_apply [source]

```
lemma coBasisFin_repr_apply {d : ℕ} (p : Co d) (i : Fin (1 + d)) :
    (coBasisFin d).repr p i = p.val (finSumFinEquiv.symm i)
```

2.40 PhysLean.Relativity.Tensors.RealTensor.Vector.Pre.Modules

PhysLean/Relativity/Tensors/RealTensor/Vector/Pre/Modules.lean on GitHub.

lemma mulVec_val [source]

```
@[simp]
lemma mulVec_val (M : Matrix (Fin 1 ⊕ Fin d) (Fin 1 ⊕ Fin d) ℝ) (v : CoMod d) :
    (M *ᵥ v).val = M *ᵥ v.val
```

2.41 PhysLean.Relativity.Tensors.Tensorial

PhysLean/Relativity/Tensors/Tensorial.lean on GitHub.

lemma smul_add [source]

```
@[simp]
lemma smul_add {g : G} [Tensorial S c M] (m m' : M) :
    g • (m + m') = g • m + g • m'
```

lemma smul_neg [source]

```
@[simp]
lemma smul_neg {n : ℕ} {c : Fin n → C} {M : Type} [AddCommGroup M] [Module k M]
    [Tensorial S c M] (g : G) (m : M) :
    g • (-m) = - (g • m)
```

lemma smul_zero [source]

```
@[simp]
lemma smul_zero [Tensorial S c M] {g : G} :
    g • (0 : M) = 0
```

def smulLinearMap [source]

The action of the group on a `Tensorial` instance as a linear map.

```
noncomputable def smulLinearMap (g : G) [Tensorial S c M] : M →ₗ[k] M where
```

lemma smulLinearMap_apply [source]

```
lemma smulLinearMap_apply {g : G} [Tensorial S c M] (m : M) :
    smulLinearMap g m = g • m
```

lemma basis_toTensor_apply [source]

```
lemma basis_toTensor_apply [Tensorial S c M] (m : M) :
    (Tensor.basis c).repr (toTensor m) = ((Tensor.basis c).map toTensor.symm).repr m
```

instance prod [source]

```
noncomputable instance prod [Tensorial S c M] {n2 : ℕ} {c2 : Fin n2 → C}
    {M₂ : Type} [AddCommMonoid M₂] [Module k M₂] [Tensorial S c2 M₂] :
    Tensorial S (Sum.elim c c2 ∘ ⇑finSumFinEquiv.symm) (M ⊗[k] M₂) where
```

lemma toTensor_tprod [source]

```
lemma toTensor_tprod {n2 : ℕ} {c2 : Fin n2 → C} {M₂ : Type}
    [Tensorial S c M] [AddCommMonoid M₂] [Module k M₂]
    [Tensorial S c2 M₂] (m : M) (m2 : M₂) :
    toTensor (m ⊗ₜ[k] m2) = Tensor.prodT (toTensor m) (toTensor m2)
```

lemma smul_prod [source]

```
lemma smul_prod {n2 : ℕ} {c2 : Fin n2 → C} {M₂ : Type}
    [Tensorial S c M] [AddCommMonoid M₂] [Module k M₂]
    [Tensorial S c2 M₂] (g : G) (m : M) (m2 : M₂) :
    g • (m ⊗ₜ[k] m2) = (g • m) ⊗ₜ[k] (g • m2)
```

lemma basis_map_prod [source]

```
lemma basis_map_prod {n2 : ℕ} {c2 : Fin n2 → C} {M₂ : Type}
    [Tensorial S c M] [AddCommMonoid M₂] [Module k M₂]
    [Tensorial S c2 M₂] :
    (Tensor.basis (S := S) (Sum.elim c c2 ∘ ⇑finSumFinEquiv.symm)).map
      (toTensor (M := (M ⊗[k] M₂))).symm =
    (((Tensor.basis (S := S) c).map (toTensor (M := M)).symm).tensorProduct
    ((Tensor.basis (S := S) c2).map (toTensor (M := M₂)).symm)).reindex
    (Tensor.ComponentIdx.splitEquiv.symm)
```

2.42 PhysLean.SpaceAndTime.Space.Basic

PhysLean/SpaceAndTime/Space/Basic.lean on GitHub.

lemma sum_apply [source]

```
@[simp]
lemma sum_apply {ι : Type} [Fintype ι] (f : ι → Space d) (i : Fin d) :
    (∑ x, f x) i = ∑ x, f x i
```

2.43 PhysLean.SpaceAndTime.Space.Distributions

PhysLean/SpaceAndTime/Space/Distributions.lean on GitHub.

lemma gradD_eq_of_inner [source]

```
lemma gradD_eq_of_inner {d} (f : (Space d) →d[ℝ] ℝ) (g : (Space d) →d[ℝ] EuclideanSpace ℝ (Fin
    d))
    (h : ∀ η y, fderivD ℝ f η y = ⟪g η, y⟫_ℝ) :
    gradD f = g
```

lemma divD_ofFunction [source]

The divergence of a distribution from a bounded function.

```
lemma divD_ofFunction {dm1 : ℕ} {f : Space dm1.succ → EuclideanSpace ℝ (Fin dm1.succ)}
    {hf : IsDistBounded f}
    {hae: AEStronglyMeasurable (fun x => f x) volume} (η : 𝓢(EuclideanSpace ℝ (Fin dm1.succ),
    ℝ)) :
    divD (Distribution.ofFunction f hf hae) η =
    - ∫ x : Space dm1.succ, ⟪f x, Space.grad η x⟫_ℝ
```

lemma integrable_isDistBounded_inner_grad_schwartzMap [source]

```
lemma integrable_isDistBounded_inner_grad_schwartzMap {dm1 : ℕ}
    {f : Space dm1.succ → EuclideanSpace ℝ (Fin dm1.succ)}
    (hf : IsDistBounded f)
    (hae: AEStronglyMeasurable (fun x => f x) volume) (η : 𝓢(EuclideanSpace ℝ (Fin dm1.succ),
    ℝ)) :
    Integrable (fun x => ⟪f x, Space.grad η x⟫_ℝ) volume
```

lemma integrable_isDistBounded_inner_grad_schwartzMap_spherical [source]

```
lemma integrable_isDistBounded_inner_grad_schwartzMap_spherical{dm1 : ℕ}
    {f : Space dm1.succ → EuclideanSpace ℝ (Fin dm1.succ)}
    (hf : IsDistBounded f)
    (hae: AEStronglyMeasurable (fun x => f x) volume) (η : 𝓢(EuclideanSpace ℝ (Fin dm1.succ),
    ℝ)) :
    Integrable ((fun x => ⟪f x.1, Space.grad η x.1⟫_ℝ)
      ∘ (homeomorphUnitSphereProd (Space dm1.succ)).symm)
      ((volume (α := Space dm1.succ)).toSphere.prod
      (Measure.volumeIoiPow (Module.finrank ℝ (EuclideanSpace ℝ (Fin dm1.succ)) - 1)))
```

2.44 PhysLean.SpaceAndTime.Space.Translations

PhysLean/SpaceAndTime/Space/Translations.lean on GitHub.

instance VAdd (EuclideanSpace ℝ (Fin d)) (Space d) [source]

```
noncomputable instance {d} : VAdd (EuclideanSpace ℝ (Fin d)) (Space d) where
```

instance AddAction (EuclideanSpace ℝ (Fin d)) (Space d) [source]

```
noncomputable instance {d} : AddAction (EuclideanSpace ℝ (Fin d)) (Space d) where
```

def translateSchwartz [source]

The continuous linear map translating Schwartz maps.

```
noncomputable def translateSchwartz {d : ℕ} (a : EuclideanSpace ℝ (Fin d)) :
    𝓢(Space d, X) →L[ℝ] 𝓢(Space d, X)
```

lemma translateSchwartz_apply [source]

```
@[simp]
lemma translateSchwartz_apply {d : ℕ} (a : EuclideanSpace ℝ (Fin d))
    (η : 𝓢(Space d, X)) (x : Space d) :
    translateSchwartz a η x = η (x - a)
```

lemma translateSchwartz_coe_eq [source]

```
lemma translateSchwartz_coe_eq {d : ℕ} (a : EuclideanSpace ℝ (Fin d))
    (η : 𝓢(Space d, X)) :
    (translateSchwartz a η : Space d → X) = fun x => η (x - a)
```

def translateD [source]

The continuous linear map translating distributions.

```
noncomputable def translateD {d : ℕ} (a : EuclideanSpace ℝ (Fin d)) :
    ((Space d) →d[ℝ] X) →ₗ[ℝ] ((Space d) →d[ℝ] X) where
```

lemma translateD_apply [source]

```
lemma translateD_apply {d : ℕ} (a : EuclideanSpace ℝ (Fin d))
    (T : (Space d) →d[ℝ] X) (η : 𝓢(Space d, ℝ)) :
    translateD a T η = T (translateSchwartz (-a) η)
```

lemma translateD_gradD [source]

```
@[simp]
lemma translateD_gradD {d : ℕ} (a : EuclideanSpace ℝ (Fin d))
    (T : (Space d) →d[ℝ] ℝ) :
    gradD (translateD a T) = translateD a (gradD T)
```

lemma translateD_ofFunction [source]

```
lemma translateD_ofFunction {d : ℕ} (a : EuclideanSpace ℝ (Fin d.succ))
    (f : Space d.succ → X) (hf : IsDistBounded f)
    (hae: AEStronglyMeasurable f volume) :
    translateD a (ofFunction f hf hae) =
    ofFunction (fun x => f (x - a)) (IsDistBounded.comp_add_right hf fun i => -a i)
    (by
      change AEStronglyMeasurable (f ∘ fun x => x - a) volume
      rw [MeasureTheory.MeasurePreserving.aestronglyMeasurable_comp_iff (μb := volume)]
      · fun_prop
      · exact measurePreserving_sub_right volume a
      · exact measurableEmbedding_subRight a)
```

lemma divD_translateD [source]

```
@[simp]
lemma divD_translateD {d : ℕ} (a : EuclideanSpace ℝ (Fin d))
    (T : (Space d) →d[ℝ] EuclideanSpace ℝ (Fin d)) :
    divD (translateD a T) = translateD a (divD T)
```

2.45 PhysLean.SpaceAndTime.Space.VectorIdentities

PhysLean/SpaceAndTime/Space/VectorIdentities.lean on GitHub.

lemma deriv_component [source]

```
lemma deriv_component (μ ν : Fin d) (x : Space d) :
    (deriv ν (fun x => x μ) x) = if ν = μ then 1 else 0
```

lemma deriv_component_sq [source]

```
lemma deriv_component_sq {d : ℕ} {ν μ : Fin d} (x : Space d) :
    (deriv ν (fun x => (x μ) ^ 2) x) = if ν = μ then 2 * x μ else 0
```

lemma norm_sq_differentiable [source]

```
@[fun_prop]
lemma norm_sq_differentiable : Differentiable ℝ (fun x : Space d => ‖x‖ ^ 2)
```

lemma deriv_norm_sq [source]

```
lemma deriv_norm_sq (x : Space d) (i : Fin d) :
    deriv i (fun x => ‖x‖ ^ 2) x = 2 * x i
```

lemma deriv_eq_inner_self [source]

```
lemma deriv_eq_inner_self (x : Space d) (i : Fin d) :
    deriv i (fun x => ⟪x, x⟫_ℝ) x = 2 * x i
```

lemma grad_apply [source]

```
lemma grad_apply {d} (f : Space d → ℝ) (x : Space d) (i : Fin d) :
    (∇ f x) i = deriv i f x
```

lemma grad_norm_sq [source]

```
lemma grad_norm_sq (x : Space d) :
    ∇ (fun x => ‖x‖ ^ 2) x = (2 : ℝ) • x
```

2.46 PhysLean.SpaceAndTime.SpaceTime.Basic

PhysLean/SpaceAndTime/SpaceTime/Basic.lean on GitHub.

lemma toTimeAndSpace_fderiv [source]

```
@[simp]
lemma toTimeAndSpace_fderiv {d : ℕ} (x : SpaceTime d) :
    fderiv ℝ toTimeAndSpace x = toTimeAndSpace.toContinuousLinearMap
```

lemma toTimeAndSpace_symm_fderiv [source]

```
@[simp]
lemma toTimeAndSpace_symm_fderiv {d : ℕ} (x : Time × Space d) :
    fderiv ℝ toTimeAndSpace.symm x = toTimeAndSpace.symm.toContinuousLinearMap
```

lemma deriv_apply_eq [source]

```
lemma deriv_apply_eq {d : ℕ} (μ ν : Fin 1 ⊕ Fin d) (f : SpaceTime d → Lorentz.Vector d)
    (hf : Differentiable ℝ f)
    (y : SpaceTime d) :
    ∂_ μ f y ν = fderiv ℝ (fun x => f x ν) y (Lorentz.Vector.basis μ)
```

lemma deriv_comp_lorentz_action [source]

```
lemma deriv_comp_lorentz_action {M : Type} [NormedAddCommGroup M] [NormedSpace ℝ M] {d : ℕ}
    (μ : Fin 1 ⊕ Fin d)
    (f : SpaceTime d → M) (hf : Differentiable ℝ f) (Λ : LorentzGroup d)
    (x : SpaceTime d) :
    ∂_ μ (fun x => f (Λ • x)) x = ∑ ν, Λ.1 ν μ • ∂_ ν f (Λ • x)
```

lemma deriv_sum_inr [source]

```
lemma deriv_sum_inr {d : ℕ} {M : Type} [NormedAddCommGroup M] [NormedSpace ℝ M]
    (f : SpaceTime d → M)
    (hf : Differentiable ℝ f) (x : SpaceTime d) (i : Fin d) :
    ∂_ (Sum.inr i) f x
    = Space.deriv i (fun y => f (toTimeAndSpace.symm ((toTimeAndSpace x).1, y)))
      (toTimeAndSpace x).2
```

lemma deriv_sum_inl [source]

```
lemma deriv_sum_inl {d : ℕ} {M : Type} [NormedAddCommGroup M]
    [NormedSpace ℝ M] (f : SpaceTime d → M)
    (hf : Differentiable ℝ f) (x : SpaceTime d) :
    ∂_ (Sum.inl 0) f x
    = Time.deriv (fun t => f (toTimeAndSpace.symm (t, (toTimeAndSpace x).2)))
      (toTimeAndSpace x).1
```

instance MeasurableSpace SpaceTime [source]

```
instance : MeasurableSpace SpaceTime
```

instance BorelSpace SpaceTime [source]

```
instance : BorelSpace SpaceTime where
```

def innerProductSpace [source]

The Euclidean inner product structure on `SpaceTime`.

```
def innerProductSpace (d : ℕ) : InnerProductSpace ℝ (SpaceTime d)
```

instance MeasureSpace SpaceTime [source]

```
instance : MeasureSpace SpaceTime where
```

instance (volume (α [source]

```
instance : (volume (α := SpaceTime)).IsOpenPosMeasure
```

instance IsFiniteMeasureOnCompacts (volume (α [source]

```
instance : IsFiniteMeasureOnCompacts (volume (α := SpaceTime))
```

instance Measure.IsAddHaarMeasure (volume (α [source]

```
instance : Measure.IsAddHaarMeasure (volume (α := SpaceTime))
```

2.47 PhysLean.SpaceAndTime.SpaceTime.Boosts

PhysLean/SpaceAndTime/SpaceTime/Boosts.lean on GitHub.

lemma boost_x_smul [source]

```
lemma boost_x_smul (β : ℝ) (hβ : |β| < 1) (x : SpaceTime) :
    LorentzGroup.boost (d := 3) 0 β hβ • x =
      fun | Sum.inl 0 => γ β * (x (Sum.inl 0) - β * x (Sum.inr 0))
          | Sum.inr 0 => γ β * (x (Sum.inr 0) - β * x (Sum.inl 0))
          | Sum.inr 1=> x (Sum.inr 1)
          | Sum.inr 2=> x (Sum.inr 2)
```

2.48 PhysLean.SpaceAndTime.Time.Basic

PhysLean/SpaceAndTime/Time/Basic.lean on GitHub.

lemma deriv_differentiable_of_contDiff [source]

```
@[fun_prop]
lemma deriv_differentiable_of_contDiff {M : Type}
    [NormedAddCommGroup M] [NormedSpace ℝ M] (f : Time → M) (hf : ContDiff ℝ ∞ f) :
    Differentiable ℝ (∂ₜ f)
```

lemma deriv_contDiff_of_contDiff [source]

```
@[fun_prop]
lemma deriv_contDiff_of_contDiff {M : Type}
    [NormedAddCommGroup M] [NormedSpace ℝ M] (f : Time → M) (hf : ContDiff ℝ ∞ f) :
    ContDiff ℝ ∞ (∂ₜ f)
```

2.49 PhysLean.StringTheory.FTheory.SU5.Charges.AnomalyFree

PhysLean/StringTheory/FTheory/SU5/Charges/AnomalyFree.lean on GitHub.

def IsAnomalyFree [source]

The condition on a collection of charges `c` that it extends to an anomaly free `Quanta`. That anomaly free `Quanta` is not tracked by this proposition.

```
def IsAnomalyFree [DecidableEq ℤ] [CommRing ℤ] (c : ChargeSpectrum ℤ) : Prop
```

instance Decidable (IsAnomalyFree c) [source]

```
instance [DecidableEq ℤ] [CommRing ℤ] {c : ChargeSpectrum ℤ} : Decidable (IsAnomalyFree c)
```

lemma isAnomalyFree_map [source]

```
lemma isAnomalyFree_map (f : ℤ →+* ℤ1) {c : ChargeSpectrum ℤ}
    (h : IsAnomalyFree c) : IsAnomalyFree (c.map (f.toAddMonoidHom))
```

lemma viable_anomalyFree [source]

The viable charges which are anomaly free.

```
lemma viable_anomalyFree (I : CodimensionOneConfig) :
    (viableCharges I).filter IsAnomalyFree =
    (match I with
    | .same => {(some 2, some (-2), {-1, 1}, {-1}), (some (-2), some 2, {-1, 1}, {1})}
    | .nearestNeighbor => {(some 6, some (-14), {-9, 1}, {-7})}
    | .nextToNearestNeighbor => ∅)
```

2.50 PhysLean.StringTheory.FTheory.SU5.Charges.Viable

PhysLean/StringTheory/FTheory/SU5/Charges/Viable.lean on GitHub.

def viableCompletions [source]

The multiset of charges which are completions of charges which minimally allow the top Yukawa, which are not phenomenologically constrained, and do not regenerate dangerous couplings.

This can be constructed via

#eval completeMinSubset same.allowedBarFiveCharges same.allowedTenCharges

```
private def viableCompletions (I : CodimensionOneConfig) : Multiset (ChargeSpectrum ℤ)
```

lemma viableCompletions_card [source]

```
lemma viableCompletions_card (I : CodimensionOneConfig) :
    (viableCompletions I).card = if I = same then 70 else
      if I = nearestNeighbor then 48 else 37
```

lemma viableCompletions_nodup [source]

```
lemma viableCompletions_nodup (I : CodimensionOneConfig) :
    (viableCompletions I).Nodup
```

lemma viableCompletions_isPhenoConstrained [source]

```
lemma viableCompletions_isPhenoConstrained (I : CodimensionOneConfig) :
    ∀ x ∈ viableCompletions I, ¬ IsPhenoConstrained x
```

lemma viableCompletions_yukawaGeneratesDangerousAtLevel_one [source]

```
lemma viableCompletions_yukawaGeneratesDangerousAtLevel_one (I : CodimensionOneConfig) :
    ∀ x ∈ viableCompletions I, ¬ YukawaGeneratesDangerousAtLevel x 1
```

lemma containsPhenoCompletionsOfMinimallyAllows_viableCompletions [source]

```
lemma containsPhenoCompletionsOfMinimallyAllows_viableCompletions :
    (I : CodimensionOneConfig) →
    ContainsPhenoCompletionsOfMinimallyAllows I.allowedBarFiveCharges
      I.allowedTenCharges (viableCompletions I)
```

def viableChargesAdditional [source]

The charges in addition to `viableCompletions` which which permit a top Yukawa coupling, are not phenomenologically constrained, and do not regenerate dangerous couplings with one insertion of a Yukawa coupling.

These can be found with e.g. #eval (viableChargesMultiset same.allowedBarFiveCharges same.allowedTenCharges).toFinset \{} (completeMinSubset same.allowedBarFiveCharges same.allowedTenCharges).toFinset

```
def viableChargesAdditional : CodimensionOneConfig → Multiset (ChargeSpectrum ℤ)
  | .same =>
    {⟨some 1, some (-3), {-1, 2}, {-3, 0}⟩, ⟨some 2, some (-3), {-2, 1}, {-3, 0}⟩,
      ⟨some (-3), some (-2), {0}, {3, -1}⟩, ⟨some (-3), some (-2), {0, 2}, {-1}⟩,
      ⟨some (-3), some (-2), {2}, {-3, -1}⟩, ⟨some (-1), some (-2), {0, 2}, {-1}⟩,
      ⟨some 0, some (-2), {-3}, {3, -1}⟩, ⟨some 0, some (-2), {-3, 1}, {-1}⟩,
      ⟨some 0, some (-2), {1}, {3, -1}⟩, ⟨some 1, some (-2), {0}, {3, -1}⟩,
      ⟨some 2, some (-2), {-3}, {-3, -1}⟩, ⟨some 2, some (-2), {-3, -1}, {-1}⟩,
      ⟨some 2, some (-2), {-3, 0}, {-1}⟩, ⟨some 2, some (-2), {-1, 1}, {-1}⟩,
      ⟨some 0, some (-2), {-1, 3}, {-3, 1}⟩, ⟨some (-2), some 2, {-1, 1}, {1}⟩,
      ⟨some (-2), some 2, {0, 3}, {1}⟩, ⟨some (-2), some 2, {1, 3}, {1}⟩,
      ⟨some (-2), some 2, {3}, {3, 1}⟩, ⟨some (-1), some 2, {0}, {-3, 1}⟩,
      ⟨some 0, some 2, {-1}, {-3, 1}⟩, ⟨some 0, some 2, {-1, 3}, {1}⟩,
      ⟨some 0, some 2, {3}, {-3, 1}⟩, ⟨some 1, some 2, {-2, 0}, {1}⟩,
      ⟨some 3, some 2, {-2}, {3, 1}⟩, ⟨some 3, some 2, {-2, 0}, {1}⟩,
      ⟨some 3, some 2, {0}, {-3, 1}⟩, ⟨some 0, some 2, {-3, 1}, {-1, 3}⟩,
      ⟨some (-2), some 3, {-1, 2}, {0, 3}⟩, ⟨some (-1), some 3, {-2, 1}, {0, 3}⟩,
      ⟨some 0, some (-2), {-3, 1}, {3, -1}⟩, ⟨some 0, some 2, {-1, 3}, {-3, 1}⟩}
  | .nearestNeighbor =>
      {⟨some (-9), some (-14), {-4}, {13, -7}⟩, ⟨some (-9), some (-14), {-4, 6}, {-7}⟩,
      ⟨some (-4), some (-14), {-9}, {13, -7}⟩, ⟨some (-4), some (-14), {-9, 6}, {-7}⟩,
      ⟨some (-4), some (-14), {-9, 11}, {-7}⟩, ⟨some (-4), some (-14), {6, 11}, {-7}⟩,
      ⟨some (-4), some (-14), {11}, {13, -7}⟩, ⟨some 1, some (-14), {-4, 6}, {-7}⟩,
      ⟨some 6, some (-14), {-9, -4}, {-7}⟩, ⟨some 6, some (-14), {-4}, {-12, -7}⟩,
      ⟨some 6, some (-14), {-9, 1}, {-7}⟩, ⟨some 6, some (-14), {1, 11}, {-7}⟩,
      ⟨some 11, some (-14), {-9, -4}, {-7}⟩, ⟨some 11, some (-14), {-4}, {13, -7}⟩,
      ⟨some 11, some (-14), {-4, 1}, {-7}⟩, ⟨some 11, some (-14), {-9, 6}, {-7}⟩,
      ⟨some (-9), some (-4), {-14, 11}, {-2}⟩, ⟨some (-9), some (-4), {11}, {-12, -2}⟩,
      ⟨some 1, some (-4), {-14, 11}, {-2}⟩, ⟨some (-14), some 6, {1, 11}, {3}⟩,
```

```
    ⟨some 11, some 6, {-14, -4}, {3}⟩, ⟨some (-4), some 6, {-9, 11}, {-7, 13}⟩,
    ⟨some (-4), some (-14), {-9, 11}, {13, -7}⟩}
| .nextToNearestNeighbor =>
    {⟨some (-13), some 2, {-8, 12}, {1}⟩, ⟨some (-8), some 2, {-13, 7}, {1}⟩,
    ⟨some 7, some 2, {-8, 12}, {1}⟩, ⟨some 12, some 2, {-13, 7}, {1}⟩,
    ⟨some (-3), some 2, {-8, 12}, {-9, 11}⟩, ⟨some (-13), some 12, {-8}, {-9, 6}⟩,
    ⟨some (-13), some 12, {-8, 7}, {6}⟩, ⟨some (-13), some 12, {7}, {-4, 6}⟩,
    ⟨some (-8), some 12, {-13}, {-9, 6}⟩, ⟨some (-8), some 12, {-13, 2}, {6}⟩,
    ⟨some (-8), some 12, {2, 7}, {6}⟩, ⟨some 2, some 12, {-13, -8}, {6}⟩,
    ⟨some 2, some 12, {-8, 7}, {6}⟩, ⟨some 7, some 12, {-13, 2}, {6}⟩}
```

lemma viableChargesAdditional_nodup [source]

```
lemma viableChargesAdditional_nodup (I : CodimensionOneConfig) :
    (viableChargesAdditional I).Nodup
```

lemma not_isPhenoConstrained_of_mem_viableChargesAdditional [source]

```
lemma not_isPhenoConstrained_of_mem_viableChargesAdditional (I : CodimensionOneConfig) :
    ∀ x ∈ (viableChargesAdditional I), ¬ IsPhenoConstrained x
```

lemma yukawaGeneratesDangerousAtLevel_one_of_mem_viableChargesAdditional [source]

```
lemma yukawaGeneratesDangerousAtLevel_one_of_mem_viableChargesAdditional
    (I : CodimensionOneConfig) :
    ∀ x ∈ (viableChargesAdditional I), ¬ YukawaGeneratesDangerousAtLevel x 1
```

lemma viableCompletions_disjiont_viableChargesAdditional [source]

```
lemma viableCompletions_disjiont_viableChargesAdditional (I : CodimensionOneConfig) :
    Disjoint (viableCompletions I) (viableChargesAdditional I)
```

def viableCharges [source]

All charges, for a given `CodimensionOneConfig`, `I`, which permit a top Yukawa coupling, are not phenomenologically constrained, and do not regenerate dangerous couplings with one insertion of a Yukawa coupling.

These trees can be found with e.g. `#eval viableChargesExt nextToNearestNeighbor`.

```
def viableCharges (I : CodimensionOneConfig) : Multiset (ChargeSpectrum ℤ)
```

lemma viableCharges_nodup [source]

```
lemma viableCharges_nodup (I : CodimensionOneConfig) :
    (viableCharges I).Nodup
```

lemma viableCharges_card [source]

```
lemma viableCharges_card (I : CodimensionOneConfig) :
    (viableCharges I).card =
    if I = .same then 102 else
    if I = .nearestNeighbor then 71 else 51
```

lemma isComplete_of_mem_viableCharges [source]

```
lemma isComplete_of_mem_viableCharges (I : CodimensionOneConfig) :
    ∀ x ∈ (viableCharges I), IsComplete x
```

lemma allowsTerm_topYukawa_of_mem_viableCharges [source]

```
lemma allowsTerm_topYukawa_of_mem_viableCharges (I : CodimensionOneConfig) :
    ∀ x ∈ (viableCharges I), x.AllowsTerm topYukawa
```

lemma not_isPhenoConstrained_of_mem_viableCharges [source]

```
lemma not_isPhenoConstrained_of_mem_viableCharges (I : CodimensionOneConfig) :
    ∀ x ∈ (viableCharges I), ¬ IsPhenoConstrained x
```

lemma not_yukawaGeneratesDangerousAtLevel_one_of_mem_viableCharges [source]

```
lemma not_yukawaGeneratesDangerousAtLevel_one_of_mem_viableCharges
    (I : CodimensionOneConfig) :
    ∀ x ∈ (viableCharges I), ¬ YukawaGeneratesDangerousAtLevel x 1
```

lemma card_five_bar_le_of_mem_viableCharges [source]

```
lemma card_five_bar_le_of_mem_viableCharges (I : CodimensionOneConfig) :
    ∀ x ∈ (viableCharges I), x.Q5.card ≤ 2
```

lemma card_ten_le_of_mem_viableCharges [source]

```
lemma card_ten_le_of_mem_viableCharges (I : CodimensionOneConfig) :
    ∀ x ∈ (viableCharges I), x.Q10.card ≤ 2
```

lemma isPhenoClosedQ5_viableCharges [source]

Inserting a `q5` charge into an element of `viableCharges I` either 1. produces another element of `viableCharges I`, or 2. produce a charge which is phenomenolically constrained or regenerates dangourous couplings with the Yukawas.

```
lemma isPhenoClosedQ5_viableCharges : (I : CodimensionOneConfig) →
    IsPhenoClosedQ5 I.allowedBarFiveCharges (viableCharges I)
```

lemma isPhenoClosedQ10_viableCharges [source]

Inserting a `q10` charge into an element of `viableCharges I` either 1. produces another element of `viableCharges I`, or 2. produce a charge which is phenomenolically constrained or regenerates dangourous couplings with the Yukawas.

```
lemma isPhenoClosedQ10_viableCharges : (I : CodimensionOneConfig) →
    IsPhenoClosedQ10 I.allowedTenCharges (viableCharges I)
```

lemma viableCompletions_subset_viableCharges [source]

```
lemma viableCompletions_subset_viableCharges (I : CodimensionOneConfig) :
    ∀ x ∈ (viableCompletions I), x ∈ viableCharges I
```

lemma mem_viableCharges_iff [source]

```
lemma mem_viableCharges_iff {I} {x : ChargeSpectrum}
    (hsub : x ∈ ofFinset I.allowedBarFiveCharges I.allowedTenCharges) :
    x ∈ viableCharges I ↔ AllowsTerm x topYukawa ∧
    ¬ IsPhenoConstrained x ∧ ¬ YukawaGeneratesDangerousAtLevel x 1 ∧ IsComplete x
```

2.51 PhysLean.StringTheory.FTheory.SU5.Fluxes.Basic

PhysLean/StringTheory/FTheory/SU5/Fluxes/Basic.lean on GitHub.

instance DecidableEq FluxesTen [source]

```
instance : DecidableEq FluxesTen
```

2.52 PhysLean.StringTheory.FTheory.SU5.Quanta.Basic

PhysLean/StringTheory/FTheory/SU5/Quanta/Basic.lean on GitHub.

instance Decidable (AnomalyCancellation (ℤ [source]

```
instance {qHd qHu F T} [CommRing ℤ] [DecidableEq ℤ] :
    Decidable (AnomalyCancellation (ℤ := ℤ) qHd qHu F T)
```

2.53 PhysLean.Units.Basic

PhysLean/Units/Basic.lean on GitHub.

lemma dimScale_apply [source]

```
lemma dimScale_apply (u1 u2 : UnitChoices) (d : Dimension) :
    dimScale u1 u2 d =
      (u1.length / u2.length) ^ (d.length : ℝ) *
      (u1.time / u2.time) ^ (d.time : ℝ) *
      (u1.mass / u2.mass) ^ (d.mass : ℝ) *
      (u1.charge / u2.charge) ^ (d.charge : ℝ) *
      (u1.temperature / u2.temperature) ^ (d.temperature : ℝ)
```

lemma dimScale_one [source]

```
@[simp]
lemma dimScale_one (u1 u2 : UnitChoices) :
    dimScale u1 u2 1 = 1
```

lemma smul_dimScale_injective [source]

```
@[simp]
lemma smul_dimScale_injective {M : Type} [MulAction ℝ≥0 M] (u1 u2 : UnitChoices) (d : Dimension
    )
```

```
    (m1 m2 : M) :
    (u1.dimScale u2 d) • m1 = (u1.dimScale u2 d) • m2 ↔ m1 = m2
```

lemma dimScale_pos [source]

```
@[simp]
lemma dimScale_pos (u1 u2 : UnitChoices) (d : Dimension) :
    0 < (dimScale u1 u2 d)
```

def SIPrimed [source]

A `UnitChoices` which is related to `SI` by a prime scaling of each of the underlying units. This is useful in proving that a result is not dimensionally correct.

```
noncomputable def SIPrimed : UnitChoices where
```

lemma dimScale_SI_SIPrimed [source]

```
@[simp]
lemma dimScale_SI_SIPrimed (d : Dimension) :
    dimScale SI SIPrimed d =
      (2⁻¹ : ℝ≥0) ^ (d.length : ℝ) *
      (3⁻¹ : ℝ≥0) ^ (d.time : ℝ) *
      (5⁻¹ : ℝ≥0) ^ (d.mass : ℝ) *
      (7⁻¹ : ℝ≥0) ^ (d.charge : ℝ) *
      (11⁻¹ : ℝ≥0) ^ (d.temperature : ℝ)
```

lemma dimScale_SIPrimed_SI [source]

```
@[simp]
lemma dimScale_SIPrimed_SI (d : Dimension) :
    dimScale SIPrimed SI d =
      (2 : ℝ≥0) ^ (d.length : ℝ) *
      (3 : ℝ≥0) ^ (d.time : ℝ) *
      (5 : ℝ≥0) ^ (d.mass : ℝ) *
      (7 : ℝ≥0) ^ (d.charge : ℝ) *
      (11 : ℝ≥0) ^ (d.temperature : ℝ)
```

lemma d_eq_CarriesDimension_d [source]

```
@[simp]
lemma ModuleCarriesDimension.d_eq_CarriesDimension_d {M : Type} [AddCommMonoid M] [Module ℝ M]
    [ModuleCarriesDimension M] :
    ModuleCarriesDimension.d M = CarriesDimension.d M
```

instance CoeFun (Dimensionful M) (fun _ => UnitChoices → M) [source]

```
instance {M : Type} [CarriesDimension M] : CoeFun (Dimensionful M) (fun _ => UnitChoices → M)
    where
```

2.54 PhysLean.Units.Dimension

`PhysLean/Units/Dimension.lean` on GitHub.

instance One Dimension [source]

```
instance : One Dimension where
```

lemma one_length [source]

```
@[simp]
lemma one_length : (1 : Dimension).length = 0
```

lemma one_time [source]

```
@[simp]
lemma one_time : (1 : Dimension).time = 0
```

lemma one_mass [source]

```
@[simp]
lemma one_mass : (1 : Dimension).mass = 0
```

lemma one_charge [source]

```
@[simp]
lemma one_charge : (1 : Dimension).charge = 0
```

lemma one_temperature [source]

```
@[simp]
lemma one_temperature : (1 : Dimension).temperature = 0
```

lemma div_length [source]

```
@[simp]
lemma div_length (d1 d2 : Dimension) : (d1 / d2).length = d1.length - d2.length
```

lemma div_time [source]

```
@[simp]
lemma div_time (d1 d2 : Dimension) : (d1 / d2).time = d1.time - d2.time
```

lemma div_mass [source]

```
@[simp]
lemma div_mass (d1 d2 : Dimension) : (d1 / d2).mass = d1.mass - d2.mass
```

lemma div_charge [source]

```
@[simp]
lemma div_charge (d1 d2 : Dimension) : (d1 / d2).charge = d1.charge - d2.charge
```

lemma div_temperature [source]

```
@[simp]
lemma div_temperature (d1 d2 : Dimension) :
    (d1 / d2).temperature = d1.temperature - d2.temperature
```

lemma npow_length [source]

```
@[simp]
lemma npow_length (d : Dimension) (n : ℕ) : (d ^ n).length = n • d.length
```

lemma npow_time [source]

```
@[simp]
lemma npow_time (d : Dimension) (n : ℕ) : (d ^ n).time = n • d.time
```

lemma npow_mass [source]

```
@[simp]
lemma npow_mass (d : Dimension) (n : ℕ) : (d ^ n).mass = n • d.mass
```

lemma npow_charge [source]

```
@[simp]
lemma npow_charge (d : Dimension) (n : ℕ) : (d ^ n).charge = n • d.charge
```

lemma npow_temperature [source]

```
@[simp]
lemma npow_temperature (d : Dimension) (n : ℕ) : (d ^ n).temperature = n • d.temperature
```

2.55 PhysLean.Units.Examples

PhysLean/Units/Examples.lean on GitHub.

def EnergyMassWithDim' [source]

An example of dimensions corresponding to `E = m c^2` using `WithDim`.

```
def EnergyMassWithDim' (m : WithDim M⌈ ℝ) (E : WithDim (M⌈ * L⌈ * L⌈ * T⌈⁻¹ * T⌈⁻¹) ℝ)
    (c : WithDim (L⌈ * T⌈⁻¹) ℝ) : Prop
```

lemma energyMassWithDim'_isDimensionallyCorrect [source]

```
lemma energyMassWithDim'_isDimensionallyCorrect :
    IsDimensionallyCorrect EnergyMassWithDim'
```

def NewtonsSecondWithDim' [source]

An example of dimensions corresponding to `F = m a` using `WithDim`.

```
def NewtonsSecondWithDim' (m : WithDim M⌈ ℝ) (F : WithDim (M⌈ * L⌈ * T⌈⁻¹ * T⌈⁻¹) ℝ)
    (a : WithDim (L⌈ * T⌈⁻¹ * T⌈⁻¹) ℝ) : Prop
```

lemma newtonsSecondWithDim'_isDimensionallyCorrect [source]

```
lemma newtonsSecondWithDim'_isDimensionallyCorrect :
    IsDimensionallyCorrect NewtonsSecondWithDim'
```

def SpeedEq [source]

An example of dimensions corresponding to `s = d/t` using `WithDim`.

```
def SpeedEq (s : WithDim (L⌈ * T⌈⁻¹) ℝ) (d : WithDim L⌈ ℝ) (t : WithDim T⌈ ℝ) : Prop
```

lemma speedEq_isDimensionallyCorrect [source]

```
lemma speedEq_isDimensionallyCorrect : IsDimensionallyCorrect SpeedEq
```

def OddDimensions [source]

An example with complicated dimensions.

```
def OddDimensions (m1 m2 : WithDim (M⌈) ℝ)
    (θ : WithDim θ⌈ ℝ) (I1 I2 : WithDim (C⌈/T⌈) ℝ) (d : WithDim L⌈ ℝ) (t : WithDim T⌈ ℝ)
    (X : WithDim (L⌈ * T⌈⁻¹ ^ 3 * θ⌈⁻¹ * C⌈ ^2) ℝ) : Prop
```

lemma oddDimensions_isDimensionallyCorrect [source]

```
lemma oddDimensions_isDimensionallyCorrect : IsDimensionallyCorrect OddDimensions
```

def EnergyMassWithDim [source]

An example of dimensions corresponding to `E = m c^2` using `WithDim` with `.val`.

```
def EnergyMassWithDim (m : WithDim M⌈ ℝ) (E : WithDim (M⌈ * L⌈ * L⌈ * T⌈⁻¹ * T⌈⁻¹) ℝ)
    (c : WithDim (L⌈ * T⌈⁻¹) ℝ) : Prop
```

lemma energyMassWithDim_isDimensionallyCorrect [source]

```
lemma energyMassWithDim_isDimensionallyCorrect : IsDimensionallyCorrect EnergyMassWithDim
```

def NewtonsSecondWithDim [source]

An example of dimensions corresponding to `F = m a` using `WithDim` with `.val`.

```
def NewtonsSecondWithDim (m : WithDim M⌈ ℝ) (F : WithDim (M⌈ * L⌈ * T⌈⁻¹ * T⌈⁻¹) ℝ)
    (a : WithDim (L⌈ * T⌈⁻¹ * T⌈⁻¹) ℝ) : Prop
```

lemma newtonsSecondWithDim_isDimensionallyCorrect [source]

```
lemma newtonsSecondWithDim_isDimensionallyCorrect :
    IsDimensionallyCorrect NewtonsSecondWithDim
```

def EnergyMassWithDimNot [source]

An example of dimensions corresponding to `E = m c` using `WithDim` with `.val`, which is not dimensionally correct.

```
def EnergyMassWithDimNot (m : WithDim M⌈ ℝ) (E : WithDim (M⌈ * L⌈ * L⌈ * T⌈⁻¹ * T⌈⁻¹) ℝ)
    (c : WithDim (L⌈ * T⌈⁻¹) ℝ) : Prop
```

lemma energyMassWithDimNot_not_isDimensionallyCorrect [source]

```
lemma energyMassWithDimNot_not_isDimensionallyCorrect :
    ¬ IsDimensionallyCorrect EnergyMassWithDimNot
```

lemma energyMass_isDimensionallyCorrect [source]

The lemma that the proposition `EnergyMass` is dimensionally correct

```
lemma energyMass_isDimensionallyCorrect :
    IsDimensionallyCorrect EnergyMass
```

def CosDim [source]

An example of a dimensionafully correct result using functions.

```
def CosDim (t : WithDim T⌈ ℝ) (ω : WithDim T⌈⁻¹ ℝ) (a : ℝ) : Prop
```

lemma cosDim_isDimensionallyCorrect [source]

```
lemma cosDim_isDimensionallyCorrect : IsDimensionallyCorrect CosDim
```

2.56 PhysLean.Units.FDeriv

PhysLean/Units/FDeriv.lean on GitHub.

lemma fderiv_apply_scaleUnit [source]

```
lemma fderiv_apply_scaleUnit (u1 u2 : UnitChoices) (x dm : M1)
    (f : M1 → M2) (hf : IsDimensionallyCorrect f) (f_diff : Differentiable ℝ f) :
    fderiv ℝ f (scaleUnit u2 u1 x) dm =
    u2.dimScale u1 (d M2) • u1.dimScale u2 (d M1) • fderiv ℝ f x dm
```

lemma fderiv_isDimensionallyCorrect [source]

If a function is dimensionally valid then so is it's derivative.

```
lemma fderiv_isDimensionallyCorrect (f : M1 → M2) (hf : IsDimensionallyCorrect f)
    (f_diff : Differentiable ℝ f) :
    IsDimensionallyCorrect (fderiv ℝ f)
```

lemma fderiv_dimension_const_direction [source]

The expression `fderiv ℝ f x dm = v.1` for a fixed `dm` and for `v` with dimension `d M2 * (d M1)⁻¹` is dimensionally correct. This is the ordinary manifestation of dimensions of a derivative, usually `dm` is taken as e.g. `1`.

This result also shows that dimensional correctness does depend on what quantities are condsidered dimensionful.

```
lemma fderiv_dimension_const_direction (dm : M1) (f : M1 → M2) (hf : IsDimensionallyCorrect f)
    (f_diff : Differentiable ℝ f) :
    IsDimensionallyCorrect (fun x (v : WithDim (d M2 * (d M1)⁻¹) M2) =>
      fderiv ℝ f x dm = v.1)
```

2.57 PhysLean.Units.Integral

PhysLean/Units/Integral.lean on GitHub.

lemma integral_isDimensionallyCorrect [source]

The statement that for a measure `μ` of dimension `d`, and a function `f : M → G` of dimension `(CarriesDimension.d G * d⁻¹)` (where `CarriesDimension.d G` is the dimension associated with terms of type `G`), then `∫ x, f x ∂μ` has the correct dimension, namely `CarriesDimension.d G`.

In other words, the function: `` ` fun (μ : DimSet (MeasureTheory.Measure M) d) (f : DimSet (M → G) (CarriesDimension.d G * d⁻¹)) ↦ ∫ x, f.1 x ∂μ.1 ` `` is dimensionally correct.

```
lemma integral_isDimensionallyCorrect (d : Dimension) :
    IsDimensionallyCorrect (fun (μ : DimSet (MeasureTheory.Measure M) d)
      (f : DimSet (M → G) (CarriesDimension.d G * d⁻¹)) ↦ ∫ x, f.1 x ∂μ.1)
```

2.58 PhysLean.Units.UnitDependent

PhysLean/Units/UnitDependent.lean on GitHub.

lemma scaleUnit_apply [source]

```
lemma ModuleCarriesDimension.scaleUnit_apply {M : Type} [AddCommMonoid M] [Module ℝ M]
    [ModuleCarriesDimension M] (u1 u2 : UnitChoices) (m : M) :
    scaleUnit u1 u2 m = ((u1.dimScale u2 (d M) : ℝ) • m : M)
```

lemma scaleUnit_apply_fun_right [source]

```
@[simp]
lemma UnitDependent.scaleUnit_apply_fun_right {M1 M2 : Type} [UnitDependent M2]
    (u1 u2 : UnitChoices) (f : M1 → M2) (m1 : M1) :
    scaleUnit u1 u2 f m1 = scaleUnit u1 u2 (f m1)
```

lemma scaleUnit_apply_fun_left [source]

```
@[simp]
lemma UnitDependent.scaleUnit_apply_fun_left {M1 M2 : Type} [UnitDependent M1]
    (u1 u2 : UnitChoices) (f : M1 → M2) (m1 : M1) :
    scaleUnit u1 u2 f m1 = f (scaleUnit u2 u1 m1)
```

lemma UnitDependent.scaleUnit_apply_fun [source]

```
@[simp]
lemma UnitDependent.scaleUnit_apply_fun {M1 M2 : Type} [UnitDependent M1]
    [UnitDependent M2] (u1 u2 : UnitChoices) (f : M1 → M2) (m1 : M1) :
    scaleUnit u1 u2 f m1 = scaleUnit u1 u2 (f (scaleUnit u2 u1 m1))
```

lemma ContinuousLinearUnitDependent.scaleUnit_apply_fun [source]

```
@[simp]
lemma ContinuousLinearUnitDependent.scaleUnit_apply_fun {M1 M2 : Type}
    [AddCommGroup M1] [Module ℝ M1]
    [TopologicalSpace M1] [ContinuousLinearUnitDependent M1]
    [AddCommGroup M2] [Module ℝ M2] [TopologicalSpace M2] [ContinuousConstSMul ℝ M2]
    [IsTopologicalAddGroup M2]
    [ContinuousLinearUnitDependent M2]
    (u1 u2 : UnitChoices) (f : M1 →L[ℝ] M2) (m1 : M1) :
    scaleUnit u1 u2 f m1 =
      scaleUnit u1 u2 (f (scaleUnit u2 u1 m1))
```

def IsDimensionallyCorrect [source]

A term of type `M` carrying an instance of `UnitDependent M` is said to be dimensionally correct if under a change of units it remains the same.

More colloquially, something is dimensionally correct if it (e.g. it's value or its truth for a proposition) does not depend on the units it is written in.

For the case of `m : M` with `CarriesDimension M` then `IsDimensionallyCorrect m` corresponds to the statement that `m` does not depend on units, e.g. is zero or the dimension of `M` is zero.

```
def IsDimensionallyCorrect {M : Type} [UnitDependent M] (m : M) : Prop
```

lemma isDimensionallyCorrect_iff [source]

```
lemma isDimensionallyCorrect_iff {M : Type} [UnitDependent M] (m : M) :
    IsDimensionallyCorrect m ↔ ∀ u1 u2 : UnitChoices,
      scaleUnit u1 u2 m = m
```

lemma isDimensionallyCorrect_fun_iff [source]

```
@[simp]
lemma isDimensionallyCorrect_fun_iff {M1 M2 : Type} [UnitDependent M1] [UnitDependent M2]
    {f : M1 → M2} :
    IsDimensionallyCorrect f ↔
    ∀ u1 u2 : UnitChoices, ∀ m, scaleUnit u1 u2 (f (scaleUnit u2 u1 m)) = f m
```

lemma isDimensionallyCorrect_fun_left [source]

```
@[simp]
lemma isDimensionallyCorrect_fun_left {M1 M2 : Type} [UnitDependent M1]
    {f : M1 → M2} :
    IsDimensionallyCorrect f ↔
    ∀ u1 u2 : UnitChoices, ∀ m, (f (scaleUnit u2 u1 m)) = f m
```

lemma isDimensionallyCorrect_fun_right [source]

```
@[simp]
lemma isDimensionallyCorrect_fun_right {M1 M2 : Type} [UnitDependent M2]
    {f : M1 → M2} :
    IsDimensionallyCorrect f ↔
    ∀ u1 u2 : UnitChoices, ∀ m, scaleUnit u1 u2 (f m) = f m
```

lemma mem_iff [source]

```
lemma DimSet.mem_iff {M : Type} [MulAction ℝ≥0 M] [MulUnitDependent M] (d : Dimension) (m : M)
    :
    m ∈ DimSet M d ↔ ∀ u1 u2, scaleUnit u1 u2 m = (UnitChoices.dimScale u1 u2 d) • m
```

2.59 PhysLean.Units.WithDim.Basic

PhysLean/Units/WithDim/Basic.lean on GitHub.

lemma val_mul_eq_mul [source]

```
@[simp]
lemma val_mul_eq_mul {d1 d2 : Dimension} (m1 : WithDim d1 ℝ) (m2 : WithDim d2 ℝ) :
    m1.val * m2.val = (m1 * m2).val
```

lemma val_pow_two_eq_mul [source]

```
@[simp]
lemma val_pow_two_eq_mul {d1 : Dimension} (m1 : WithDim d1 ℝ) :
    m1.val ^ 2 = (m1 * m1).val
```

lemma scaleUnit_val_eq_scaleUnit_val [source]

```
@[simp]
lemma scaleUnit_val_eq_scaleUnit_val {d : Dimension} (M : Type) [MulAction ℝ≥0 M]
    (u1 u2 : UnitChoices) (m1 m2 : WithDim d M) :
    (scaleUnit u1 u2 m1).val = (scaleUnit u1 u2 m2).val ↔ m1.val = m2.val
```

lemma scaleUnit_val_eq_scaleUnit_val_of_dim_eq [source]

```
lemma scaleUnit_val_eq_scaleUnit_val_of_dim_eq {d1 d2 : Dimension} {M : Type} [MulAction ℝ≥0 M]
    {u1 u2 : UnitChoices} {m1 : WithDim d1 M} {m2 : WithDim d2 M}
    (h : d1 = d2 := by ext <;> {simp; try ring}) :
    (scaleUnit u1 u2 m1).val = (scaleUnit u1 u2 m2).val ↔ m1.val = m2.val
```

lemma scaleUnit_val [source]

```
lemma scaleUnit_val {d : Dimension} (M : Type) [MulAction ℝ≥0 M]
    (u1 u2 : UnitChoices) (m1 : WithDim d M) :
    (scaleUnit u1 u2 m1).val = u1.dimScale u2 d • m1.val
```

instance HDiv (WithDim d1 ℝ) (WithDim d2 ℝ) (WithDim (d1 * d2⁻¹) ℝ) [source]

```
noncomputable instance (d1 d2 : Dimension) :
    HDiv (WithDim d1 ℝ) (WithDim d2 ℝ) (WithDim (d1 * d2⁻¹) ℝ) where
```

lemma val_div_val [source]

```
@[simp]
lemma val_div_val {d1 d2 : Dimension} (m1 : WithDim d1 ℝ) (m2 : WithDim d2 ℝ) :
    (m1.val / m2.val) = (m1 / m2).val
```

lemma div_scaleUnit [source]

```
@[simp]
lemma div_scaleUnit {d1 d2 : Dimension} (m1 : WithDim d1 ℝ) (m2 : WithDim d2 ℝ)
    (u1 u2 : UnitChoices) :
    (scaleUnit u1 u2 m1) / (scaleUnit u1 u2 m2) = scaleUnit u1 u2 (m1 / m2)
```

lemma scaleUnit_dim_eq_zero [source]

```
@[simp]
lemma scaleUnit_dim_eq_zero {d : Dimension} (m : WithDim d ℝ) (u1 u2 : UnitChoices)
    (h : d = 1 := by ext <;> {simp; try ring}) : scaleUnit u1 u2 m = m
```

def cast [source]

The casting from `WithDim d M` to `WithDim d2 M` when `d = d2`.

```
@[nolint unusedArguments]
def cast {d d2 : Dimension} {M : Type} [MulAction ℝ≥0 M] (m : WithDim d M)
    (h : d = d2 := by ext <;> {simp; try ring}) : WithDim d2 M
```

lemma cast_refl [source]

```
@[simp]
lemma cast_refl {d : Dimension} {M : Type} [MulAction ℝ≥0 M] (m : WithDim d M) :
    cast m rfl = m
```

lemma cast_scaleUnit [source]

```
@[simp]
lemma cast_scaleUnit {d d2 : Dimension} {M : Type} [MulAction ℝ≥0 M] (m : WithDim d M)
    (h : d = d2) (u1 u2 : UnitChoices) :
    cast (scaleUnit u1 u2 m) h = scaleUnit u1 u2 (cast m h)
```

2.60 PhysLean.Units.WithDim.Speed

PhysLean/Units/WithDim/Speed.lean on GitHub.

def speedOfLight [source]

The dimensionful speed of light correspnoding to 299792458 meters per second.

```
noncomputable def speedOfLight : Dimensionful (WithDim (L⌈ * T⌈⁻¹) ℝ)
```

lemma speedOfLight_in_SI [source]

```
@[simp]
lemma speedOfLight_in_SI : speedOfLight SI = ⟨299792458⟩
```

2.61 scripts.MetaPrograms.module_doc_lint

scripts/MetaPrograms/module_doc_lint.lean on GitHub.

structure DocLintError [source]

```
structure DocLintError where
  msg : String
  file : FilePath
```

def getHeaddings [source]

```
def getHeaddings (f : FilePath) : IO (Array String)
```

def getTableOfContents [source]

```
def getTableOfContents (f : FilePath) : IO (Array String)
```

inductive Steps [source]

```
inductive Steps where
  | titleHead
  | overviewHead
  | keyResultsHead
  | tableOfContentsHead
  | referencesHead
  | otherHeadings
  | tableOfContentsCorrect
```

def toString [source]

```
def Steps.toString : Steps → String
  | .titleHead => "Add a title heading starting with '# ' for the whole module."
  | .overviewHead => "Add an overview section '## i. Overview' after the title heading."
  | .keyResultsHead => "Add a key results section '## ii. Key results' after the overview
    section."
  | .tableOfContentsHead => "Add a table of contents section '## iii. Table of contents' after
    the key results section. This can be filled in later."
  | .referencesHead => "Add a references section '## iv. References' after the table of
    contents section."
  | .otherHeadings => "Add other headings for sections and subsections using e.g. '## A.', '###
     A.1.', '#### A.1.2' etc."
  | .tableOfContentsCorrect => "Fix the table of contents to match the headings in the file."
```

def anyTrue [source]

```
def Steps.anyTrue (e  : Steps → Bool × String) : Bool
```

def checkHeadings [source]

```
def checkHeadings (f : FilePath) : IO (List DocLintError)
```

def noLintArray [source]

The array of modules not to be linted.

```
def noLintArray : IO (Array FilePath)
```