Physlib Monthly Report

1–31 August 2025

Contributors: CasualPhysicsEnjoier, Daniel Morrison, David Feng, Garmelon, Joseph Tooby-Smith, LibertasSpZ, Matteo Cipollina

Reviewers: Eric Wieser, Joseph Tooby-Smith

Generated 14 August 2026 from commit `a439539`.

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 1–31 August 2025 on branch `master` of `leanprover-community/physlib`, between commits `1d07315` and `a439539`. Over that period 7 contributors landed 122 commits, changing 16,731 lines across 258 files and adding 531 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.Mass.MassUnit` . . . 3
- 2.2 `PhysLean.Electromagnetism.Charge.ChargeUnit` . . . 3
- 2.3 `PhysLean.Electromagnetism.Electrostatics.ThreeDimension.PointParticle` 4
- 2.4 `PhysLean.Mathematics.DataStructures.Matrix.LieTrace` . . . 4
- 2.5 `PhysLean.Mathematics.Trigonometry.Tanh` . . . 8
- 2.6 `PhysLean.Particles.SuperSymmetry.SU5.Charges.AllowsTerm` . . . 8
- 2.7 `PhysLean.Particles.SuperSymmetry.SU5.Charges.Completions` . . . 10
- 2.8 `PhysLean.Particles.SuperSymmetry.SU5.Charges.Map` . . . 10
- 2.9 `PhysLean.Particles.SuperSymmetry.SU5.Charges.MinimallyAllowsTerm.OfFinset` 11
- 2.10 `PhysLean.Particles.SuperSymmetry.SU5.Charges.MinimalSuperSet` . . . 11
- 2.11 `PhysLean.Particles.SuperSymmetry.SU5.Charges.OfPotentialTerm` . . . 12
- 2.12 `PhysLean.Particles.SuperSymmetry.SU5.Charges.PhenoClosed` . . . 13
- 2.13 `PhysLean.Particles.SuperSymmetry.SU5.Charges.PhenoConstrained` . . . 16
- 2.14 `PhysLean.Particles.SuperSymmetry.SU5.Charges.Yukawa` . . . 17
- 2.15 `PhysLean.QuantumMechanics.FiniteTarget.Basic` . . . 17
- 2.16 `PhysLean.QuantumMechanics.OneDimension.ReflectionlessPotential.Basic` 17
- 2.17 `PhysLean.Relativity.LorentzAlgebra.ExponentialMap` . . . 18
- 2.18 `PhysLean.SpaceAndTime.Space.Distributions` . . . 19
- 2.19 `PhysLean.SpaceAndTime.Space.LengthUnit` . . . 19

2.20 PhysLean.SpaceAndTime.Space.SpaceStruct . . . 20
2.21 PhysLean.SpaceAndTime.Space.VectorIdentities . . . 21
2.22 PhysLean.SpaceAndTime.Time.Basic . . . 22
2.23 PhysLean.SpaceAndTime.Time.TimeUnit . . . 23
2.24 PhysLean.StatisticalMechanics.CanonicalEnsemble.Basic . . . 24
2.25 PhysLean.StatisticalMechanics.CanonicalEnsemble.Finite . . . 30
2.26 PhysLean.StatisticalMechanics.CanonicalEnsemble.Lemmas . . . 32
2.27 PhysLean.StringTheory.FTheory.SU5.Charges.AnomalyFree . . . 37
2.28 PhysLean.StringTheory.FTheory.SU5.Charges.OfRationalSection . . . 37
2.29 PhysLean.StringTheory.FTheory.SU5.Charges.Viable . . . 40
2.30 PhysLean.StringTheory.FTheory.SU5.Fluxes.Basic . . . 44
2.31 PhysLean.StringTheory.FTheory.SU5.Fluxes.NoExotics.Elems . . . 44
2.32 PhysLean.StringTheory.FTheory.SU5.Quanta.Basic . . . 45
2.33 PhysLean.StringTheory.FTheory.SU5.Quanta.FiveQuanta . . . 46
2.34 PhysLean.StringTheory.FTheory.SU5.Quanta.IsViable . . . 50
2.35 PhysLean.StringTheory.FTheory.SU5.Quanta.TenQuanta . . . 54
2.36 PhysLean.Thermodynamics.Temperature.Basic . . . 57
2.37 PhysLean.Thermodynamics.Temperature.TemperatureUnits . . . 58
2.38 PhysLean.Units.Basic . . . 58
2.39 PhysLean.Units.Examples . . . 69
2.40 PhysLean.Units.FDeriv . . . 70
2.41 PhysLean.Units.Integral . . . 70
2.42 PhysLean.Units.WithDim.Area . . . 71
2.43 PhysLean.Units.WithDim.Basic . . . 73
2.44 PhysLean.Units.WithDim.Energy . . . 74
2.45 PhysLean.Units.WithDim.Pressure . . . 74
2.46 PhysLean.Units.WithDim.Speed . . . 75

1 Introduction

During Aug 2025, 7 contributors submitted 122 commits that changed 16,731 lines (+10,440 / -6,291) across 258 files spanning 21 top-level areas of the repository. Of those, 22 files were newly added and 21 were removed outright. This report catalogues 531 newly added Lean declarations: 360 lemmas, 87 defs, 56 instances, 10 theorems, 7 abbrevs, 7 classs, 3 structures, and 1 inductive. We're delighted to recognize the following first-time contributors this month: Daniel Morrison, Garmelon, and LibertasSpZ.

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	206	11
Daniel Morrison	268	2
David Feng	280	9
Garmelon	149	1
Joseph Tooby-Smith	16,568	24
LibertasSpZ	302	2
Matteo Cipollina	10,307	73

1.2 Reviewers

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

Reviewer	PRs Reviewed
Eric Wieser	1
Joseph Tooby-Smith	13

1.3 Modified subfolders

Subfolder	Files	New	Deleted	Additions	Deletions
`PhysLean/StringTheory`	33	+6	-20	+2,849	-4,132
`PhysLean/Particles`	41	+1	-1	+1,518	-615
`PhysLean/StatisticalMechanics`	4	+1	–	+1,556	-140
`PhysLean/Units`	12	+8	–	+1,502	-57
`PhysLean/Relativity`	56	–	–	+511	-478
`PhysLean/Mathematics`	25	+2	–	+707	-157
`PhysLean/SpaceAndTime`	9	+1	–	+480	-183
`PhysLean/Electromagnetism`	6	+1	–	+403	-73
`PhysLean/QFT`	43	–	–	+186	-192
`PhysLean/Thermodynamics`	2	–	–	+211	-64
`scripts`	2	+2	–	+269	-0
`PhysLean/QuantumMechanics`	10	–	–	+117	-96
`PhysLean/ClassicalMechanics`	7	–	–	+69	-43

PhysLean.lean	1	–	–	+25	-26
lake-manifest.json	1	–	–	+21	-21
PhysLean/CondensedMatter	1	–	–	+3	-4
README.md	1	–	–	+3	-4
lakefile.toml	1	–	–	+5	-2
PhysLean/Meta	1	–	–	+3	-2
.github/ISSUE_TEMPLATE	1	–	–	+1	-1
lean-toolchain	1	–	–	+1	-1

1.4 New declarations by kind

Kind	Count
lemma	360
def	87
instance	56
theorem	10
abbrev	7
class	7
structure	3
inductive	1

2 Details by file

2.1 PhysLean.ClassicalMechanics.Mass.MassUnit

PhysLean/ClassicalMechanics/Mass/MassUnit.lean on GitHub.

instance HDiv MassUnit MassUnit ℝ≥0 [source]

```
noncomputable instance : HDiv MassUnit MassUnit ℝ≥0 where
```

lemma div_neq_zero [source]

```
@[simp]
lemma div_neq_zero (x y : MassUnit) : ¬ x / y = (0 : ℝ≥0)
```

lemma div_mul_div_coe [source]

```
@[simp]
lemma div_mul_div_coe (x y z : MassUnit) :
    (x / y : ℝ) * (y /z : ℝ) = x /z
```

lemma self_div_scale [source]

```
@[simp]
lemma self_div_scale (x : MassUnit) (r : ℝ) (hr : 0 < r) :
    x / scale r x hr = ((1/r, _root_.div_nonneg (by simp) (le_of_lt hr)) : ℝ≥0)
```

2.2 PhysLean.Electromagnetism.Charge.ChargeUnit

PhysLean/Electromagnetism/Charge/ChargeUnit.lean on GitHub.

instance HDiv ChargeUnit ChargeUnit ℝ≥0 [source]

```
noncomputable instance : HDiv ChargeUnit ChargeUnit ℝ≥0 where
```

lemma div_neq_zero [source]

```
@[simp]
lemma div_neq_zero (x y : ChargeUnit) : ¬ x / y = (0 : ℝ≥0)
```

lemma div_mul_div_coe [source]

```
@[simp]
lemma div_mul_div_coe (x y z : ChargeUnit) :
    (x / y : ℝ) * (y /z : ℝ) = x /z
```

lemma self_div_scale [source]

```
@[simp]
lemma self_div_scale (x : ChargeUnit) (r : ℝ) (hr : 0 < r) :
    x / scale r x hr = ((1/r, _root_.div_nonneg (by simp) (le_of_lt hr)) : ℝ≥0)
```

2.3 PhysLean.Electromagnetism.Electrostatics.ThreeDimension.PointParticle

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

def chargeDistribution [source]

The charge distribution of a point particle of charge `q` in 1d space sitting at the origin. Mathematically, this corresponds to a dirac delta distribution centered at the origin.

```
def chargeDistribution (q : ℝ) : ChargeDistribution 3
```

def electricField [source]

The electric field of a point particle of charge `q` in 3d space sitting at the origin. Mathematically, this corresponds to the distribution associated to the distribution `(q/(4 * π * ε)) • ‖r‖⁻¹ ^ 3 • r`.

```
def electricField (q ε : ℝ) : StaticElectricField 3
```

lemma gaussLaw [source]

Guass' law for a point particle in 3-dimensions, 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 (q ε : ℝ) : (electricField q ε).GaussLaw ε (chargeDistribution q)
```

2.4 PhysLean.Mathematics.DataStructures.Matrix.LieTrace

PhysLean/Mathematics/DataStructures/Matrix/LieTrace.lean on GitHub.

instance UniformSpace (Matrix m n 𝕂) [source]

```
instance [UniformSpace 𝕂] : UniformSpace (Matrix m n 𝕂)
```

lemma tsum_eq_zero [source]

If every term of a series is zero, then its sum is zero.

```
lemma tsum_eq_zero
    {β : Type*} [TopologicalSpace β] [AddCommMonoid β]
    {f : ℕ → β} (h : ∀ n, f n = 0) :
    (∑' n, f n) = 0
```

lemma matrix_tsum_apply [source]

Apply a matrix `tsum` to a given entry.

```
lemma matrix_tsum_apply
    {f : ℕ → Matrix m m 𝕂} (hf : Summable f) (i j : m) :
    (∑' n, f n) i j = ∑' n, (f n) i j
```

lemma diag_mul_of_blockTriangular_id [source]

For upper-triangular matrices, the diagonal of a product is the product of the diagonals. This is a specific case of a more general property for block-triangular matrices.

```
lemma diag_mul_of_blockTriangular_id {A B : Matrix m m 𝕂}
    (hA : BlockTriangular A id) (hB : BlockTriangular B id) : (A * B).diag = A.diag * B.diag
```

lemma pow [source]

Powers of block triangular matrices are block triangular.

```
lemma blockTriangular.pow {A : Matrix m m 𝕂} (hA : BlockTriangular A id) (k : ℕ) :
    BlockTriangular (A ^ k) id
```

lemma diag_pow_of_blockTriangular_id [source]

For an upper-triangular matrix, the diagonal of a power is the power of the diagonal.

```
lemma diag_pow_of_blockTriangular_id {A : Matrix m m 𝕂}
    (hA : BlockTriangular A id) (k : ℕ) : (A ^ k).diag = A.diag ^ k
```

lemma blockTriangular_exp_of_blockTriangular_id [source]

The exponential of an upper-triangular matrix is upper-triangular.

```
lemma blockTriangular_exp_of_blockTriangular_id
    {A : Matrix m m 𝕂} (hA : BlockTriangular A id) :
    (NormedSpace.exp 𝕂 A).BlockTriangular id
```

lemma diag_pow_entry_eq_pow_diag_entry [source]

For an upper–triangular matrix `A`, the `(i,i)` entry of the power `A ^ n` is simply the `n`-th power of the original diagonal entry.

```
lemma diag_pow_entry_eq_pow_diag_entry {A : Matrix m m 𝕂}
    (hA : BlockTriangular A id) (n : ℕ) (i : m) :
    (A ^ n) i i = (A i i) ^ n
```

lemma exp_series_diag_term_eq [source]

Each term in the matrix exponential series equals the corresponding scalar term on the diagonal

```
lemma exp_series_diag_term_eq {A : Matrix m m 𝕂} (hA : BlockTriangular A id)
    (n : ℕ) (i : m) :
    ((n.factorial : 𝕂)⁻¹ • (A ^ n)) i i = (n.factorial : 𝕂)⁻¹ • (A i i) ^ n
```

lemma matrix_exp_series_diag_eq_scalar_series [source]

The diagonal of the matrix exponential series equals the scalar exponential series

```
lemma matrix_exp_series_diag_eq_scalar_series {A : Matrix m m 𝕂} (hA : BlockTriangular A id)
    (i : m) :
    (∑' n, ((n.factorial : 𝕂)⁻¹ • (A ^ n)) i i) = ∑' n, (n.factorial : 𝕂)⁻¹ • (A i i) ^ n
```

theorem diag_exp_of_blockTriangular_id [source]

The diagonal of the exponential of an upper-triangular matrix `A` consists of the exponentials of the diagonal entries of `A`.

```
theorem diag_exp_of_blockTriangular_id
    {A : Matrix m m 𝕂} (hA : BlockTriangular A id) :
    (NormedSpace.exp 𝕂 A).diag = fun i => NormedSpace.exp 𝕂 (A i i)
```

lemma det_exp_of_blockTriangular_id [source]

Lie's trace formula for upper triangular matrices.

```
lemma det_exp_of_blockTriangular_id {A : Matrix m m 𝕂} (hA : BlockTriangular A id) :
    (NormedSpace.exp 𝕂 A).det = NormedSpace.exp 𝕂 A.trace
```

lemma trace_unitary_conj [source]

The trace is invariant under unitary conjugation.

```
lemma trace_unitary_conj (A : Matrix m m 𝕂) (U : unitaryGroup m 𝕂) :
    trace ((U : Matrix m m 𝕂) * A * star (U : Matrix m m 𝕂)) = trace A
```

lemma det_unitary_conj [source]

The determinant is invariant under unitary conjugation.

```
lemma det_unitary_conj (A : Matrix m m 𝕂) (U : unitaryGroup m 𝕂) :
    det ((U : Matrix m m 𝕂) * A * star (U : Matrix m m 𝕂)) = det A
```

lemma exp_unitary_conj [source]

The exponential of a matrix commutes with unitary conjugation.

```
lemma exp_unitary_conj (A : Matrix m m 𝕂) (U : unitaryGroup m 𝕂) :
    NormedSpace.exp 𝕂 ((U : Matrix m m 𝕂) * A * star (U : Matrix m m 𝕂)) =
      (U : Matrix m m 𝕂) * NormedSpace.exp 𝕂 A * star (U : Matrix m m 𝕂)
```

lemma det_exp_unitary_conj [source]

```
lemma det_exp_unitary_conj (A : Matrix m m 𝕂) (U : unitaryGroup m 𝕂) :
    (NormedSpace.exp 𝕂 ((U : Matrix m m 𝕂) * A * star (U : Matrix m m 𝕂))).det =
    (NormedSpace.exp 𝕂 A).det
```

theorem det_exp [source]

*The determinant of the exponential of a matrix is the exponential of its trace. This is also known as **Lie's trace formula**.*

```
theorem det_exp {𝕂 m : Type*} [RCLike 𝕂] [IsAlgClosed 𝕂] [Fintype m] [LinearOrder m]
    (A : Matrix m m 𝕂) :
    (NormedSpace.exp 𝕂 A).det = NormedSpace.exp 𝕂 A.trace
```

lemma map_tsum [source]

```
lemma map_tsum {α β m n : Type*}
    [AddCommMonoid α] [AddCommMonoid β] [TopologicalSpace α] [TopologicalSpace β]
    [T2Space β]
    (f : α →+ β) (hf : Continuous f) {s : ℕ → Matrix m n α} (hs : Summable s) :
    (∑' k, s k).map f = ∑' k, (s k).map f
```

lemma map_pow [source]

```
lemma map_pow {α β m : Type*}
    [Fintype m] [DecidableEq m] [Semiring α] [Semiring β]
    (f : α →+* β) (A : Matrix m m α) (k : ℕ) :
    (A ^ k).map f = (A.map f) ^ k
```

lemma exp_map_algebraMap [source]

```
lemma exp_map_algebraMap {n : Type*} [Fintype n] [DecidableEq n]
    (A : Matrix n n ℝ) :
    (exp ℝ A).map (algebraMap ℝ ℂ) = exp ℂ (A.map (algebraMap ℝ ℂ))
```

theorem det_exp_real [source]

Lie's trace formula over $\mathbb{R}$: $det(exp(A)) = exp(tr(A))$ for any real matrix A. This is proved by transferring the result from $\mathbb{C}$ using the naturality of polynomial identities.

```
theorem det_exp_real {n : Type*} [Fintype n] [LinearOrder n]
    (A : Matrix n n ℝ) : (NormedSpace.exp ℝ A).det = Real.exp A.trace
```

2.5 PhysLean.Mathematics.Trigonometry.Tanh

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

lemma tanh_lt_one [source]

tanh(x) is less than 1 for all x

```
lemma tanh_lt_one (x : ℝ) : tanh x < 1
```

lemma minus_one_lt_tanh [source]

tanh(x) is greater than -1 for all x

```
lemma minus_one_lt_tanh (x : ℝ) : -1 < tanh x
```

lemma abs_tanh_lt_one [source]

The absolute value of tanh is bounded by 1

```
lemma abs_tanh_lt_one (x : ℝ) : |tanh x| < 1
```

lemma deriv_tanh [source]

The derivative of tanh(x) is 1 - tanh(x)^2

```
@[sorryful]
lemma deriv_tanh (x: ℝ) : deriv Real.tanh = fun x => 1 - Real.tanh x ^ 2
```

lemma contDiff_tanh [source]

Tanh(x) is n times continuously differentiable for all n

```
lemma contDiff_tanh {n : ℕ} : ContDiff ℝ n tanh
```

lemma iteratedDeriv_tanh_is_polynomial_of_tanh [source]

The nth derivative of Tanh(x) is a polynomial of Tanh(x)

```
@[sorryful]
lemma iteratedDeriv_tanh_is_polynomial_of_tanh (n : ℕ) (x : ℝ) : ∃ P : Polynomial ℝ, ∀ x,
    iteratedDeriv n Real.tanh x = P.eval (Real.tanh x)
```

2.6 PhysLean.Particles.SuperSymmetry.SU5.Charges.AllowsTerm

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

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 : Charges ℤ) (q5 : ℤ) (T : PotentialTerm) : Prop
```

instance Decidable (AllowsTermQ5 x q5 T) [source]

```
instance [DecidableEq ℤ] (x : Charges 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 : Charges ℤ) (q10 : ℤ) (T : PotentialTerm) : Prop
```

instance Decidable (AllowsTermQ10 x q10 T) [source]

```
instance [DecidableEq ℤ] (x : Charges 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
```

2.7 PhysLean.Particles.SuperSymmetry.SU5.Charges.Completions

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

lemma completions_nodup [source]

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

def completionsTopYukawa [source]

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

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

lemma completionsTopYukawa_nodup [source]

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

lemma completions_eq_completionsTopYukawa_of_mem_minimallyAllowsTermsOfFinset [source]

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

2.8 PhysLean.Particles.SuperSymmetry.SU5.Charges.Map

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

lemma mem_map_ofPotentialTerm'_iff [source]

```
lemma mem_map_ofPotentialTerm'_iff[DecidableEq ℤ]
    (f : ℤ →+ ℤ1) (x : Charges 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 : Charges ℤ) (T : PotentialTerm) :
    (ofPotentialTerm' (map f x) T).toFinset = (ofPotentialTerm' x T).toFinset.image f
```

2.9 PhysLean.Particles.SuperSymmetry.SU5.Charges.MinimallyAllowsTerm.OfFinset

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

lemma not_isPhenoConstrained_of_minimallyAllowsTermsOfFinset_topYukawa [source]

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

2.10 PhysLean.Particles.SuperSymmetry.SU5.Charges.MinimalSuperSet

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

lemma insert_filter_card_zero [source]

```
lemma insert_filter_card_zero
    (T : Multiset (Charges ℤ)) (S5 S10 : Finset ℤ)
    (p : Charges ℤ → Prop) [DecidablePred p]
    (hComplet : ∀ x ∈ T, IsComplete x)
    (h10 : ∀ q10 : S10, ((T.map fun x => (x.1, x.2.1, x.2.2.1, insert q10.1 x.2.2.2)).filter
      fun y => (y ∉ T ∧ p y)) = ∅)
    (h5 : ∀ q5 : S5, ((T.map fun x => (x.1, x.2.1, insert q5.1 x.2.2.1, x.2.2.2)).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 (Charges ℤ))
    (S5 S10 : Finset ℤ)
    (p : Charges ℤ → Prop) [DecidablePred p]
    (hnotSubset : ∀ (x y : Charges ℤ), x ⊆ y → ¬ p x → ¬ p y)
    (hComplet : ∀ x ∈ T, IsComplete x)
    (x : Charges ℤ)
    (hx : x ∈ T) (y : Charges ℤ) (hsubset : x ⊆ y)
    (hy : y ∈ ofFinset S5 S10)
    (h10 : ∀ q10 : S10, ((T.map fun x => (x.1, x.2.1, x.2.2.1, insert q10.1 x.2.2.2)).filter
      fun y => (y ∉ T ∧ p y)) = ∅)
    (h5 : ∀ q5 : S5, ((T.map fun x => (x.1, x.2.1, insert q5.1 x.2.2.1, x.2.2.2)).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 (Charges 𝒵))
    (S5 S10 : Finset 𝒵)
    (p : Charges 𝒵 → Prop) [DecidablePred p]
    (hnotSubset : ∀ (x y : Charges 𝒵), x ⊆ y → ¬ p x → ¬ p y)
    (hComplet : ∀ x ∈ T, IsComplete x)
    (x : Charges 𝒵)
    (hx : x ∈ T) (y : Charges 𝒵) (hsubset : x ⊆ y)
    (hy : y ∈ ofFinset S5 S10)
    (h10 : ∀ q10 : S10, ((T.map fun x => (x.1, x.2.1, x.2.2.1, insert q10.1 x.2.2.2)).filter
      fun y => (y ∉ T ∧ p y)) = ∅)
    (h5 : ∀ q5 : S5, ((T.map fun x => (x.1, x.2.1, insert q5.1 x.2.2.1, x.2.2.2)).filter
      fun y => (y ∉ T ∧ p y)) = ∅) :
      y ∉ T → ¬ p y
```

2.11 PhysLean.Particles.SuperSymmetry.SU5.Charges.OfPotentialTerm

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

lemma ofPotentialTerm'_μ_finset [source]

```
lemma ofPotentialTerm'_μ_finset {x : Charges 𝒵} :
    x.ofPotentialTerm' μ =
    (x.1.toFinset.product <| x.2.1.toFinset).val.map (fun x => x.1 - x.2)
```

lemma ofPotentialTerm'_β_finset [source]

```
lemma ofPotentialTerm'_β_finset {x : Charges 𝒵} :
    x.ofPotentialTerm' β =
    (x.2.1.toFinset.product <| x.2.2.1).val.map (fun x => - x.1 + x.2)
```

lemma ofPotentialTerm'_W2_finset [source]

```
lemma ofPotentialTerm'_W2_finset {x : Charges 𝒵} :
    x.ofPotentialTerm' W2 = (x.1.toFinset.product <|
      x.2.2.2.product <| x.2.2.2.product <| x.2.2.2).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 : Charges 𝒵} :
    x.ofPotentialTerm' W3 = (x.2.1.toFinset.product <| x.2.2.1.product <| x.2.2.1).val.map
    (fun x => -x.1 - x.1 + x.2.1 + x.2.2)
```

lemma ofPotentialTerm'_W4_finset [source]

```
lemma ofPotentialTerm'_W4_finset {x : Charges 𝒵} :
    x.ofPotentialTerm' W4 = (x.1.toFinset.product <|
      x.2.1.toFinset.product <| x.2.2.1).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 : Charges 𝓩} :
    x.ofPotentialTerm' K2 = (x.1.toFinset.product <|
      x.2.1.toFinset.product <| x.2.2.2).val.map
    (fun x => x.1 + x.2.1 + x.2.2)
```

lemma ofPotentialTerm'_topYukawa_finset [source]

```
lemma ofPotentialTerm'_topYukawa_finset {x : Charges 𝓩} :
    x.ofPotentialTerm' topYukawa = (x.2.1.toFinset.product <|
      x.2.2.2.product <| x.2.2.2).val.map
    (fun x => -x.1 + x.2.1 + x.2.2)
```

lemma ofPotentialTerm'_bottomYukawa_finset [source]

```
lemma ofPotentialTerm'_bottomYukawa_finset {x : Charges 𝓩} :
    x.ofPotentialTerm' bottomYukawa = (x.1.toFinset.product <|
      x.2.2.1.product <| x.2.2.2).val.map
    (fun x => x.1 + x.2.1 + x.2.2)
```

lemma ofPotentialTerm'_mono [source]

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

lemma ofPotentialTerm'_empty [source]

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

2.12 PhysLean.Particles.SuperSymmetry.SU5.Charges.PhenoClosed

PhysLean/Particles/SuperSymmetry/SU5/Charges/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 (Charges 𝓩)) : Prop
```

lemma isPhenClosedQ5_of_isPhenoConstrainedQ5 [source]

```
lemma isPhenClosedQ5_of_isPhenoConstrainedQ5 {S5 : Finset 𝓩} {charges : Multiset (Charges 𝓩)}
    (h : ∀ q5 ∈ S5, ∀ x ∈ charges,
```

```
    let y : Charges 𝒵 := (x.1, x.2.1, insert q5 x.2.2.1, x.2.2.2)
    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 (Charges 𝒵)) : Prop
```

lemma isPhenClosedQ10_of_isPhenoConstrainedQ10 [source]

```
lemma isPhenClosedQ10_of_isPhenoConstrainedQ10 {S10 : Finset 𝒵} {charges : Multiset (Charges
    𝒵)}
    (h : ∀ q10 ∈ S10, ∀ x ∈ charges,
      let y : Charges 𝒵 := (x.1, x.2.1, x.2.2.1, insert q10 x.2.2.2)
      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 (Charges
     𝒵)) :
    Prop
```

lemma containsPhenoCompletionsOfMinimallyAllows_iff_completionsTopYukawa [source]

```
lemma containsPhenoCompletionsOfMinimallyAllows_iff_completionsTopYukawa {S5 S10 : Finset 𝒵}
    {charges : Multiset (Charges 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 (Charges 𝒵)} :
    Decidable (ContainsPhenoCompletionsOfMinimallyAllows S5 S10 charges)
```

lemma containsPhenoCompletionsOfMinimallyAllows_of_subset [source]

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

lemma completeness_of_isPhenoClosedQ5_isPhenoClosedQ10 [source]

```
lemma completeness_of_isPhenoClosedQ5_isPhenoClosedQ10
    {S5 S10 : Finset ℤ} {charges : Multiset (Charges 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 : Charges ℤ} (hsub : x ∈ ofFinset S5 S10) :
    x ∈ charges ↔ AllowsTerm x .topYukawa ∧
    ¬ IsPhenoConstrained x ∧ ¬ YukawaGeneratesDangerousAtLevel x 1 ∧ IsComplete x
```

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 (Charges 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 (Charges 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)
```

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 (Charges ℤ)
```

2.13 PhysLean.Particles.SuperSymmetry.SU5.Charges.PhenoConstrained

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

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 : Charges ℤ) (q5 : ℤ) : Prop
```

lemma isPhenoConstrainedQ5_iff [source]

```
lemma isPhenoConstrainedQ5_iff [DecidableEq ℤ] (x : Charges 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 : Charges 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 : Charges ℤ) (q10 : ℤ) : Prop
```

lemma isPhenoConstrainedQ10_iff [source]

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

instance decidableIsPhenoConstrainedQ10 [source]

```
instance decidableIsPhenoConstrainedQ10 [DecidableEq ℤ] (x : Charges 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)
```

2.14 PhysLean.Particles.SuperSymmetry.SU5.Charges.Yukawa

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

lemma YukawaGeneratesDangerousAtLevel_iff_inter [source]

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

lemma yukawaGeneratesDangerousAtLevel_iff_toFinset [source]

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

2.15 PhysLean.QuantumMechanics.FiniteTarget.Basic

PhysLean/QuantumMechanics/FiniteTarget/Basic.lean on GitHub.

def timeEvolutionMatrixStandard [source]

An instance of timeEvolutionmatrix over the standard basis.

```
noncomputable def timeEvolutionMatrixStandard (t : ℝ) :
  Matrix (Fin n) (Fin n) ℂ
```

2.16 PhysLean.QuantumMechanics.OneDimension.ReflectionlessPotential.Basic

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

def tanhOperator [source]

Define tanh(κ X) operator

```
noncomputable def tanhOperator (ψ : ℝ → ℂ) : ℝ → ℂ
```

def creationOperator [source]

Creation operator: a† as defined in https://arxiv.org/pdf/2411.14941 a† = 1/√(2m) (P + iħκ tanh(κX))

```
noncomputable def creationOperator (ψ : ℝ → ℂ) : ℝ → ℂ
```

def annihilationOperator [source]

Annihilation operator: a as defined in https://arxiv.org/pdf/2411.14941 a = 1/√(2m) (P - iħκ tanh(κX))

```
noncomputable def annihilationOperator (ψ : ℝ → ℂ) : ℝ → ℂ
```

2.17 PhysLean.Relativity.LorentzAlgebra.ExponentialMap

PhysLean/Relativity/LorentzAlgebra/ExponentialMap.lean on GitHub.

instance UniformSpace (Matrix m n 𝕂) [source]

```
instance [UniformSpace 𝕂] : UniformSpace (Matrix m n 𝕂)
```

lemma trace_eq_sum_diagonal [source]

The trace of a matrix equals the sum of its diagonal elements.

```
lemma trace_eq_sum_diagonal (A : Matrix (Fin 1 ⊕ Fin 3) (Fin 1 ⊕ Fin 3) ℝ) :
    trace A = ∑ i, A i i
```

lemma trace_of_mem_is_zero [source]

The trace of any element of the Lorentz algebra is zero.

```
lemma trace_of_mem_is_zero (A : lorentzAlgebra) : trace A.1 = 0
```

lemma trace_reindex [source]

```
@[simp]
lemma trace_reindex [Semiring R] [Fintype ι] (e : n ≃ ι) (A : Matrix n n R) :
    trace (A.submatrix e.symm e.symm) = trace A
```

lemma exp_reindex [source]

```
@[simp]
lemma exp_reindex {k : Type*}
    [RCLike k] [Fintype ι] [DecidableEq ι] (e : n ≃ ι) (A : Matrix n n k) :
    NormedSpace.exp k (A.submatrix e.symm e.symm) = reindex e e (NormedSpace.exp k A)
```

theorem exp_isProper [source]

The exponential of an element of the Lorentz algebra is proper (has determinant 1).

```
theorem exp_isProper (A : lorentzAlgebra) :
    LorentzGroup.IsProper ⟨(NormedSpace.exp ℝ) A.1, exp_mem_lorentzGroup A⟩
```

theorem exp_isOrthochronous [source]

The exponential of an element of the Lorentz algebra is orthochronous.

```
theorem exp_isOrthochronous (A : lorentzAlgebra) :
    LorentzGroup.IsOrthochronous ⟨(NormedSpace.exp ℝ) A.1, exp_mem_lorentzGroup A⟩
```

theorem exp_mem_restricted_lorentzGroup [source]

The exponential of an element of the Lorentz algebra is a member of the restricted Lorentz group.

```
theorem exp_mem_restricted_lorentzGroup (A : lorentzAlgebra) :
    (((NormedSpace.exp ℝ) A.1, exp_mem_lorentzGroup A) : LorentzGroup 3) ∈
    LorentzGroup.restricted 3
```

2.18 PhysLean.SpaceAndTime.Space.Distributions

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

lemma divD_ofBounded [source]

The divergence of a distribution from a bounded function.

```
lemma divD_ofBounded {dm1 : ℕ} {f : Space dm1.succ → EuclideanSpace ℝ (Fin dm1.succ)}
    {hf : ∃ c1 c2 n, 0 ≤ c1 ∧ 0 ≤ c2 ∧ ∀ x, ‖f x‖ ≤ c1 * ‖x‖ ^ (-dm1 : ℝ) + c2 * ‖x‖ ^ n}
    {hae: AEStronglyMeasurable (fun x => f x) volume} (η : 𝓢(EuclideanSpace ℝ (Fin dm1.succ),
    ℝ)) :
    divD (Distribution.ofBounded f hf hae) η =
    - ∫ x : Space dm1.succ, ⟪f x, Space.grad η x⟫_ℝ
```

lemma integrable_ofBounded_inner_grad_schwartzMap [source]

```
lemma integrable_ofBounded_inner_grad_schwartzMap {dm1 : ℕ}
    {f : Space dm1.succ → EuclideanSpace ℝ (Fin dm1.succ)}
    (hf : ∃ c1 c2 n, 0 ≤ c1 ∧ 0 ≤ c2 ∧ ∀ x, ‖f x‖ ≤ c1 * ‖x‖ ^ (-dm1 : ℝ) + c2 * ‖x‖ ^ n)
    (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_ofBounded_inner_grad_schwartzMap_spherical [source]

```
lemma integrable_ofBounded_inner_grad_schwartzMap_spherical{dm1 : ℕ}
    {f : Space dm1.succ → EuclideanSpace ℝ (Fin dm1.succ)}
    (hf : ∃ c1 c2 n, 0 ≤ c1 ∧ 0 ≤ c2 ∧ ∀ x, ‖f x‖ ≤ c1 * ‖x‖ ^ (-dm1 : ℝ) + c2 * ‖x‖ ^ n)
    (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)
      ∘ (homeomorphUnitSpherePerod (Space dm1.succ)).symm)
      ((volume (α := Space dm1.succ)).toSphere.prod
      (Measure.volumeIoiPow (Module.finrank ℝ (EuclideanSpace ℝ (Fin dm1.succ)) - 1)))
```

2.19 PhysLean.SpaceAndTime.Space.LengthUnit

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

instance HDiv LengthUnit LengthUnit ℝ≥0 [source]

```
noncomputable instance : HDiv LengthUnit LengthUnit ℝ≥0 where
```

lemma div_neq_zero [source]

```
@[simp]
lemma div_neq_zero (x y : LengthUnit) : ¬ x / y = (0 : ℝ≥0)
```

lemma div_mul_div_coe [source]

```
@[simp]
lemma div_mul_div_coe (x y z : LengthUnit) :
    (x / y : ℝ) * (y /z : ℝ) = x /z
```

lemma self_div_scale [source]

```
@[simp]
lemma self_div_scale (x : LengthUnit) (r : ℝ) (hr : 0 < r) :
    x / scale r x hr = (⟨1/r, _root_.div_nonneg (by simp) (le_of_lt hr)⟩ : ℝ≥0)
```

lemma scale_scale [source]

```
@[simp]
lemma scale_scale (x : LengthUnit) (r1 r2 : ℝ) (hr1 : 0 < r1) (hr2 : 0 < r2) :
    scale r1 (scale r2 x hr2) hr1 = scale (r1 * r2) x (mul_pos hr1 hr2)
```

2.20 PhysLean.SpaceAndTime.Space.SpaceStruct

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

structure SpaceStruct [source]

The type `SpaceStruct d` representes `d` dimensional Euclidean space. The default value of `d` is `3`. Thus `SpaceStuct = Spacestruct 3`.

```
structure SpaceStruct (d : ℕ := 3) where
    val : EuclideanSpace ℝ (Fin d)
```

instance Add (SpaceStruct d) [source]

```
noncomputable instance {d : Nat} : Add (SpaceStruct d) where
```

lemma add_val [source]

```
@[simp]
lemma add_val {d: ℕ} (x y : SpaceStruct d) :
    x + y = ⟨x.val + y.val⟩
```

instance Neg (SpaceStruct d) [source]

```
instance {d : Nat} : Neg (SpaceStruct d) where
```

lemma neg_val [source]

```
@[simp]
lemma neg_val {d: ℕ} (x : SpaceStruct d) :
    (-x).val = -x.val
```

instance Sub (SpaceStruct d) [source]

```
noncomputable instance {d: ℕ} : Sub (SpaceStruct d)
  where sub x y
```

instance SMul ℝ (SpaceStruct d) [source]

```
instance {d : Nat} : SMul ℝ (SpaceStruct d) where
```

instance Zero (SpaceStruct d) [source]

```
instance {d : Nat} : Zero (SpaceStruct d)
```

instance Inner ℝ (SpaceStruct d) [source]

```
noncomputable instance (d: ℕ) : Inner ℝ (SpaceStruct d) where
```

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

```
noncomputable instance : VAdd (EuclideanSpace ℝ (Fin d)) (SpaceStruct d) where
```

instance AddGroup (SpaceStruct d) [source]

```
noncomputable instance {d : Nat} : AddGroup (SpaceStruct d) where
```

instance AddCommMonoid (SpaceStruct d) [source]

```
noncomputable instance {d: ℕ} : AddCommMonoid (SpaceStruct d) where
```

instance AddCommGroup (SpaceStruct d) [source]

```
noncomputable instance {d : Nat} : AddCommGroup (SpaceStruct d) where
```

lemma inner_eq_sum [source]

```
lemma inner_eq_sum {d} (p q : SpaceStruct d) :
    inner ℝ p q = ∑ i, p.val i * q.val i
```

2.21 PhysLean.SpaceAndTime.Space.VectorIdentities

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

lemma grad_inner_space_unit_vector [source]

The gradient in the direction of the space position.

```
lemma grad_inner_space_unit_vector {d} (x : Space d) (f : Space d → ℝ) (hd : Differentiable ℝ f
    ) :
    ⟪∇ f x, ‖x‖⁻¹ • x⟫_ℝ = _root_.deriv (fun r => f (r • ‖x‖⁻¹ • x)) ‖x‖
```

lemma grad_inner_space [source]

```
lemma grad_inner_space {d} (x : Space d) (f : Space d → ℝ) (hd : Differentiable ℝ f) :
    ⟪∇ f x, x⟫_ℝ = ‖x‖ * _root_.deriv (fun r => f (r • ‖x‖⁻¹ • x)) ‖x‖
```

2.22 PhysLean.SpaceAndTime.Time.Basic

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

instance NatCast Time [source]

```
instance : NatCast Time where
```

instance OfNat Time n [source]

The casting of a natural number to an element of `Time`. This corresponds to a choice of (1) zero point in time, and (2) a choice of metric on time (defining `1`).

```
instance {n : ℕ} : OfNat Time n where
```

instance Coe ℝ Time [source]

```
instance : Coe ℝ Time where
```

lemma realCast_val [source]

```
lemma realCast_val {r : ℝ} : (r : Time).val = r
```

instance Inhabited Time [source]

```
instance : Inhabited Time where
```

lemma default_eq_zero [source]

```
@[simp]
lemma default_eq_zero : default = 0
```

lemma natCast_val [source]

```
@[simp]
lemma natCast_val {n : ℕ} : val n = n
```

lemma natCast_zero [source]

```
@[simp]
lemma natCast_zero : ((0 : ℕ) : Time) = 0
```

lemma natCast_one [source]

```
@[simp]
lemma natCast_one : ((1 : ℕ) : Time) = 1
```

lemma ofNat_val [source]

```
@[simp]
lemma ofNat_val {n : ℕ} : val (OfNat.ofNat n : Time) = n
```

lemma one_ne_zero [source]

```
lemma one_ne_zero : (1 : Time) ≠ (0 : Time)
```

lemma realCast_of_natCast [source]

```
@[simp]
lemma realCast_of_natCast {n : ℕ} : ((n : ℝ) : Time) = n
```

lemma eq_zero_iff [source]

```
@[simp]
lemma eq_zero_iff (t : Time) : t = 0 ↔ t.val = 0
```

lemma eq_one_iff [source]

```
@[simp]
lemma eq_one_iff (t : Time) : t = 1 ↔ t.val = 1
```

2.23 PhysLean.SpaceAndTime.Time.TimeUnit

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

instance HDiv TimeUnit TimeUnit ℝ≥0 [source]

```
noncomputable instance : HDiv TimeUnit TimeUnit ℝ≥0 where
```

lemma div_neq_zero [source]

```
@[simp]
lemma div_neq_zero (x y : TimeUnit) : ¬ x / y = (0 : ℝ≥0)
```

lemma div_mul_div_coe [source]

```
@[simp]
lemma div_mul_div_coe (x y z : TimeUnit) :
    (x / y : ℝ) * (y /z : ℝ) = x /z
```

lemma self_div_scale [source]

```
@[simp]
lemma self_div_scale (x : TimeUnit) (r : ℝ) (hr : 0 < r) :
    x / scale r x hr = ((1/r, _root_.div_nonneg (by simp) (le_of_lt hr)) : ℝ≥0)
```

lemma scale_scale [source]

```
@[simp]
lemma scale_scale (x : TimeUnit) (r1 r2 : ℝ) (hr1 : 0 < r1) (hr2 : 0 < r2) :
    scale r1 (scale r2 x hr2) hr1 = scale (r1 * r2) x (mul_pos hr1 hr2)
```

2.24 PhysLean.StatisticalMechanics.CanonicalEnsemble.Basic

PhysLean/StatisticalMechanics/CanonicalEnsemble/Basic.lean on GitHub.

lemma dof_add [source]

```
@[simp]
lemma dof_add (𝒞1 : CanonicalEnsemble ι) (𝒞2 : CanonicalEnsemble ι1) :
    (𝒞1 + 𝒞2).dof = 𝒞1.dof + 𝒞2.dof
```

lemma phase_space_unit_add [source]

```
@[simp]
lemma phase_space_unit_add (𝒞1 : CanonicalEnsemble ι) (𝒞2 : CanonicalEnsemble ι1) :
    (𝒞1 + 𝒞2).phaseSpaceunit = 𝒞1.phaseSpaceunit
```

lemma dof_nsmul [source]

```
@[simp]
lemma dof_nsmul (n : ℕ) : (nsmul n 𝒞).dof = n * 𝒞.dof
```

lemma phase_space_unit_nsmul [source]

```
@[simp]
lemma phase_space_unit_nsmul (n : ℕ) :
    (nsmul n 𝒞).phaseSpaceunit = 𝒞.phaseSpaceunit
```

lemma dof_congr [source]

```
@[simp]
lemma dof_congr (e : ι1 ≃ᵐ ι) :
    (𝒞.congr e).dof = 𝒞.dof
```

lemma phase_space_unit_congr [source]

```
@[simp]
lemma phase_space_unit_congr (e : ι1 ≃ᵐ ι) :
    (𝒞.congr e).phaseSpaceunit = 𝒞.phaseSpaceunit
```

def mathematicalPartitionFunction [source]

The mathematical partition function, defined as the integral of the Boltzmann factor. This quantity may have physical dimensions. See `CanonicalEnsemble.partitionFunction` for the dimensionless physical version.

```
noncomputable def mathematicalPartitionFunction (T : Temperature) : ℝ
```

lemma mathematicalPartitionFunction_eq_integral [source]

```
lemma mathematicalPartitionFunction_eq_integral (T : Temperature) :
    mathematicalPartitionFunction 𝒞 T = ∫ i, exp (- T.β * 𝒞.energy i) ∂𝒞.μ
```

lemma mathematicalPartitionFunction_add [source]

```
lemma mathematicalPartitionFunction_add {T : Temperature} :
    (𝒞 + 𝒞1).mathematicalPartitionFunction T =
    𝒞.mathematicalPartitionFunction T * 𝒞1.mathematicalPartitionFunction T
```

lemma mathematicalPartitionFunction_congr [source]

```
@[simp]
lemma mathematicalPartitionFunction_congr (e : ι1 ≃ᵐ ι) (T : Temperature) :
    (𝒞.congr e).mathematicalPartitionFunction T = 𝒞.mathematicalPartitionFunction T
```

lemma mathematicalPartitionFunction_nsmul [source]

The `mathematicalPartitionFunction_nsmul` function of `n` copies of a canonical ensemble.

```
lemma mathematicalPartitionFunction_nsmul (n : ℕ) (T : Temperature) :
    (nsmul n 𝒞).mathematicalPartitionFunction T = (𝒞.mathematicalPartitionFunction T) ^ n
```

lemma mathematicalPartitionFunction_nonneg [source]

```
lemma mathematicalPartitionFunction_nonneg (T : Temperature) :
    0 ≤ 𝒞.mathematicalPartitionFunction T
```

lemma mathematicalPartitionFunction_eq_zero_iff [source]

```
lemma mathematicalPartitionFunction_eq_zero_iff (T : Temperature) [IsFiniteMeasure (𝒞.μBolt T)]
     :
    mathematicalPartitionFunction 𝒞 T = 0 ↔ 𝒞.μ = 0
```

lemma mathematicalPartitionFunction_comp_ofβ_apply [source]

```
lemma mathematicalPartitionFunction_comp_ofβ_apply (β : ℝ≥0) :
    𝒞.mathematicalPartitionFunction (ofβ β) =
    (𝒞.μ.withDensity (fun i => ENNReal.ofReal (exp (- β * 𝒞.energy i)))).real Set.univ
```

lemma mathematicalPartitionFunction_pos [source]

The partition function is strictly positive provided the underlying measure is non-zero and the Boltzmann measure is finite.

```
lemma mathematicalPartitionFunction_pos (T : Temperature)
    [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] :
    0 < 𝒞.mathematicalPartitionFunction T
```

def differentialEntropy [source]

The (differential) entropy of the canonical ensemble. In the continuous case, this quantity is not absolute but depends on the choice of units for the measure. It can be negative. See `thermodynamicEntropy` for the absolute physical quantity.

```
noncomputable def differentialEntropy (T : Temperature) : ℝ
```

lemma probability_nonneg [source]

Probabilities are non-negative, assuming a positive partition function.

```
lemma probability_nonneg
    (T : Temperature) [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] (i : ι) :
    0 ≤ 𝒞.probability T i
```

lemma probability_pos [source]

Probabilities are strictly positive.

```
lemma probability_pos
    (T : Temperature) [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] (i : ι) :
    0 < 𝒞.probability T i
```

lemma differentialEntropy_nonneg_of_prob_le_one [source]

General entropy non-negativity under a pointwise upper bound `probability ≤ 1`. This assumption holds automatically in the finite/counting case (since sums bound each term), but can fail in general (continuous) settings; hence we separate it as a hypothesis. Finite case: see `CanonicalEnsemble.entropy_nonneg` in `Finite`.

```
lemma differentialEntropy_nonneg_of_prob_le_one
    (T : Temperature) [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ]
    (hInt : Integrable (fun i => Real.log (𝒞.probability T i)) (𝒞.μProd T))
    (hP_le_one : ∀ i, 𝒞.probability T i ≤ 1) :
    0 ≤ 𝒞.differentialEntropy T
```

lemma partitionFunction_def [source]

```
@[simp]
lemma partitionFunction_def (𝒞 : CanonicalEnsemble ι) (T : Temperature) :
    𝒞.partitionFunction T =
      𝒞.mathematicalPartitionFunction T / (𝒞.phaseSpaceunit ^ 𝒞.dof)
```

lemma partitionFunction_pos [source]

```
lemma partitionFunction_pos
    (𝒞 : CanonicalEnsemble ι) (T : Temperature)
    [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] :
    0 < 𝒞.partitionFunction T
```

lemma partitionFunction_dof_zero [source]

```
lemma partitionFunction_dof_zero
    (𝒞 : CanonicalEnsemble ι) (T : Temperature) (h : 𝒞.dof = 0) :
    𝒞.partitionFunction T = 𝒞.mathematicalPartitionFunction T
```

lemma partitionFunction_phase_space_unit_one [source]

```
lemma partitionFunction_phase_space_unit_one
    (𝒞 : CanonicalEnsemble ι) (T : Temperature) (h : 𝒞.phaseSpaceunit = 1) :
    𝒞.partitionFunction T = 𝒞.mathematicalPartitionFunction T
```

lemma log_partitionFunction [source]

```
lemma log_partitionFunction
    (𝒞 : CanonicalEnsemble ι) (T : Temperature)
    [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] :
    Real.log (𝒞.partitionFunction T)
      = Real.log (𝒞.mathematicalPartitionFunction T)
        - (𝒞.dof : ℝ) * Real.log 𝒞.phaseSpaceunit
```

lemma log_partitionFunction_ofβ [source]

A rewriting form convenient under a coercion to a temperature obtained from an inverse temperature.

```
lemma log_partitionFunction_ofβ
    (𝒞 : CanonicalEnsemble ι) (β : ℝ≥0)
    [IsFiniteMeasure (𝒞.μBolt (ofβ β))] [NeZero 𝒞.μ] :
    Real.log (𝒞.partitionFunction (ofβ β))
      = Real.log (𝒞.mathematicalPartitionFunction (ofβ β))
        - (𝒞.dof : ℝ) * Real.log 𝒞.phaseSpaceunit
```

lemma log_mathematicalPartitionFunction_eq [source]

The logarithm of the mathematical partition function as an integral.

```
lemma log_mathematicalPartitionFunction_eq
    (𝒞 : CanonicalEnsemble ι) (T : Temperature) :
    Real.log (𝒞.mathematicalPartitionFunction T)
      = Real.log (∫ i, Real.exp (- T.β * 𝒞.energy i) ∂ 𝒞.μ)
```

def helmholtzFreeEnergy [source]

The Helmholtz free energy, `F = -k_B T log(Z)`. This is the central quantity from which other thermodynamic properties are derived.

```
noncomputable def helmholtzFreeEnergy (T : Temperature) : ℝ
```

lemma helmholtzFreeEnergy_def [source]

```
@[simp]
lemma helmholtzFreeEnergy_def
    (𝒞 : CanonicalEnsemble ι) (T : Temperature) :
    𝒞.helmholtzFreeEnergy T = - kB * T.val * Real.log (𝒞.partitionFunction T)
```

lemma helmholtzFreeEnergy_congr [source]

```
lemma helmholtzFreeEnergy_congr
    (𝒞 : CanonicalEnsemble ι) (e : ι1 ≃ᵐ ι) (T : Temperature) :
    (𝒞.congr e).helmholtzFreeEnergy T = 𝒞.helmholtzFreeEnergy T
```

lemma helmholtzFreeEnergy_dof_zero [source]

```
lemma helmholtzFreeEnergy_dof_zero
    (𝒞 : CanonicalEnsemble ι) (T : Temperature) (h : 𝒞.dof = 0) :
    𝒞.helmholtzFreeEnergy T
      = -kB * T.val * Real.log (𝒞.mathematicalPartitionFunction T)
```

lemma helmholtzFreeEnergy_phase_space_unit_one [source]

```
lemma helmholtzFreeEnergy_phase_space_unit_one
    (𝒞 : CanonicalEnsemble ι) (T : Temperature) (h : 𝒞.phaseSpaceunit = 1) :
    𝒞.helmholtzFreeEnergy T
      = -kB * T.val * Real.log (𝒞.mathematicalPartitionFunction T)
```

lemma helmholtzFreeEnergy_add [source]

```
lemma helmholtzFreeEnergy_add
    (𝒞 : CanonicalEnsemble ι) (𝒞1 : CanonicalEnsemble ι1) (T : Temperature)
    [IsFiniteMeasure (𝒞.μBolt T)] [IsFiniteMeasure (𝒞1.μBolt T)]
    [NeZero 𝒞.μ] [NeZero 𝒞1.μ]
    (h : 𝒞.phaseSpaceunit = 𝒞1.phaseSpaceunit) :
    (𝒞 + 𝒞1).helmholtzFreeEnergy T
      = 𝒞.helmholtzFreeEnergy T + 𝒞1.helmholtzFreeEnergy T
```

lemma helmholtzFreeEnergy_nsmul [source]

```
lemma helmholtzFreeEnergy_nsmul
    (𝒞 : CanonicalEnsemble ι) (n : ℕ) (T : Temperature) :
    (nsmul n 𝒞).helmholtzFreeEnergy T
      = n * 𝒞.helmholtzFreeEnergy T
```

def physicalProbability [source]

The dimensionless physical probability density. This is is the probability density w.r.t. the measure, obtained by dividing the phase space measure by the fundamental unit `h^dof`, making the probability density `ρ_phys = ρ_math * h^dof` dimensionless.

```
noncomputable def physicalProbability (T : Temperature) (i : ι) : ℝ
```

lemma physicalProbability_def [source]

```
@[simp]
lemma physicalProbability_def (T : Temperature) (i : ι) :
    𝒞.physicalProbability T i
      = 𝒞.probability T i * (𝒞.phaseSpaceunit ^ 𝒞.dof)
```

lemma physicalProbability_measurable [source]

```
lemma physicalProbability_measurable (T : Temperature) :
    Measurable (𝒞.physicalProbability T)
```

lemma physicalProbability_nonneg [source]

```
lemma physicalProbability_nonneg
    (T : Temperature) [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] (i : ι) :
    0 ≤ 𝒞.physicalProbability T i
```

lemma physicalProbability_pos [source]

```
lemma physicalProbability_pos
    (T : Temperature) [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] (i : ι) :
    0 < 𝒞.physicalProbability T i
```

lemma log_physicalProbability [source]

```
lemma log_physicalProbability
    (T : Temperature) [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] (i : ι) :
    Real.log (𝒞.physicalProbability T i)
      = Real.log (𝒞.probability T i) + (𝒞.dof : ℝ) * Real.log 𝒞.phaseSpaceunit
```

lemma integral_probability [source]

```
lemma integral_probability
    (𝒞 : CanonicalEnsemble ι) (T : Temperature)
    [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] :
    (∫ i, 𝒞.probability T i ∂ 𝒞.μ) = 1
```

lemma integral_physicalProbability_base [source]

Normalization of the dimensionless physical probability density over the base measure.

```
lemma integral_physicalProbability_base
    (𝒞 : CanonicalEnsemble ι) (T : Temperature)
    [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] :
    (∫ i, 𝒞.physicalProbability T i ∂ 𝒞.μ)
      = 𝒞.phaseSpaceunit ^ 𝒞.dof
```

lemma physicalProbability_dof_zero [source]

```
lemma physicalProbability_dof_zero
    (T : Temperature) (h : 𝒞.dof = 0) (i : ι) :
    𝒞.physicalProbability T i = 𝒞.probability T i
```

lemma physicalProbability_phase_space_unit_one [source]

```
lemma physicalProbability_phase_space_unit_one
    (T : Temperature) (h : 𝒞.phaseSpaceunit = 1) (i : ι) :
    𝒞.physicalProbability T i = 𝒞.probability T i
```

lemma physicalProbability_congr [source]

```
lemma physicalProbability_congr (e : ι1 ≃ᵐ ι) (T : Temperature) (i : ι1) :
    (𝒞.congr e).physicalProbability T i
      = 𝒞.physicalProbability T (e i)
```

lemma physicalProbability_add [source]

```
lemma physicalProbability_add
    {ι1} [MeasurableSpace ι1]
    (𝒞1 : CanonicalEnsemble ι1) (T : Temperature) (i : ι × ι1)
    (h : 𝒞.phaseSpaceunit = 𝒞1.phaseSpaceunit) :
    (𝒞 + 𝒞1).physicalProbability T i
      = 𝒞.physicalProbability T i.1 * 𝒞1.physicalProbability T i.2
```

def thermodynamicEntropy [source]

The absolute thermodynamic entropy, defined from its statistical mechanical foundation as the Gibbs-Shannon entropy of the dimensionless physical probability distribution. This corresponds to Landau & Lifshitz, Statistical Physics, §7, Eq. 7.12.

```
noncomputable def thermodynamicEntropy (T : Temperature) : ℝ
```

lemma thermodynamicEntropy_def [source]

```
@[simp]
lemma thermodynamicEntropy_def (T : Temperature) :
    𝒞.thermodynamicEntropy T = -kB * ∫ i, Real.log (𝒞.physicalProbability T i) ∂ 𝒞.μProd T
```

2.25 PhysLean.StatisticalMechanics.CanonicalEnsemble.Finite

PhysLean/StatisticalMechanics/CanonicalEnsemble/Finite.lean on GitHub.

instance NeZero 𝒞.μ [source]

In the finite (counting) case a nonempty index type gives a nonzero measure.

```
instance [IsFinite 𝒞] [Nonempty ι] : NeZero 𝒞.μ
```

def shannonEntropy [source]

The Shannon entropy of a finite canonical ensemble. This is defined by the formula `S = -k_B ∑ pᵢ log pᵢ`. It is proven to be equivalent to the `differentialEntropy` and the `thermodynamicEntropy` for systems satisfying the `IsFinite` property.

```
noncomputable def shannonEntropy (T : Temperature) : ℝ
```

lemma mathematicalPartitionFunction_of_fintype [source]

```
lemma mathematicalPartitionFunction_of_fintype [IsFinite 𝒞] (T : Temperature) :
    𝒞.mathematicalPartitionFunction T = ∑ i, exp (- β T * 𝒞.energy i)
```

lemma probability_le_one [source]

```
lemma probability_le_one [IsFinite 𝒞] [Nonempty ι] (T : Temperature) (i : ι) :
    𝒞.probability T i ≤ 1
```

lemma mathematicalPartitionFunction_pos_finite [source]

Finite specialization: strict positivity of the mathematical partition function.

```
lemma mathematicalPartitionFunction_pos_finite
    [IsFinite 𝒞] [Nonempty ι] (T : Temperature) :
    0 < 𝒞.mathematicalPartitionFunction T
```

lemma partitionFunction_pos_finite [source]

Finite specialization: strict positivity of the (physical) partition function.

```
lemma partitionFunction_pos_finite [IsFinite 𝒞] [Nonempty ι] (T : Temperature) :
    0 < 𝒞.partitionFunction T
```

lemma probability_nonneg_finite [source]

Finite specialization: non-negativity (indeed positivity) of probabilities.

```
lemma probability_nonneg_finite [IsFinite 𝒞] [Nonempty ι] (T : Temperature) (i : ι) :
    0 ≤ 𝒞.probability T i
```

lemma sum_probability_eq_one [source]

The sum of probabilities over all microstates is 1.

```
lemma sum_probability_eq_one [IsFinite 𝒞] [Nonempty ι] (T : Temperature) :
    ∑ i, 𝒞.probability T i = 1
```

lemma entropy_nonneg [source]

The entropy of a finite canonical ensemble (Shannon entropy) is non-negative.

```
lemma entropy_nonneg [IsFinite 𝒞] [Nonempty ι] (T : Temperature) :
    0 ≤ 𝒞.shannonEntropy T
```

lemma shannonEntropy_eq_differentialEntropy [source]

```
lemma shannonEntropy_eq_differentialEntropy [IsFinite 𝒞] (T : Temperature) :
    𝒞.shannonEntropy T = 𝒞.differentialEntropy T
```

theorem thermodynamicEntropy_eq_shannonEntropy [source]

In the finite, nonempty case the thermodynamic and Shannon entropies coincide. All semi-classical correction factors vanish (`dof = 0`, `phaseSpaceUnit = 1`), so the absolute thermodynamic entropy reduces to the discrete Shannon form.

```
theorem thermodynamicEntropy_eq_shannonEntropy [IsFinite 𝒞]
    (T : Temperature) :
    𝒞.thermodynamicEntropy T = 𝒞.shannonEntropy T
```

2.26 PhysLean.StatisticalMechanics.CanonicalEnsemble.Lemmas

PhysLean/StatisticalMechanics/CanonicalEnsemble/Lemmas.lean on GitHub.

def mathematicalHelmholtzFreeEnergy [source]

An intermediate potential defined from the mathematical partition function. See `helmholtzFreeEnergy` for the physical thermodynamic quantity.

```
noncomputable def mathematicalHelmholtzFreeEnergy (T : Temperature) : ℝ
```

lemma helmholtzFreeEnergy_eq_helmholtzMathematicalFreeEnergy_add_correction [source]

The relationship between the physical Helmholtz Free Energy and the Helmholtz Potential.

```
lemma helmholtzFreeEnergy_eq_helmholtzMathematicalFreeEnergy_add_correction (T : Temperature)
    [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] :
    𝒞.helmholtzFreeEnergy T = 𝒞.mathematicalHelmholtzFreeEnergy T +
      kB * T.val * 𝒞.dof * Real.log (𝒞.phaseSpaceunit)
```

lemma differentialEntropy_eq_kB_beta_meanEnergy_add_kB_log_mathZ [source]

General identity: $S_diff = kB\ \beta\ \langle E\rangle + kB\ log\ Z_math$. This connects the differential entropy to the mean energy and the mathematical partition function. Integrability of `log (probability …)` follows from the pointwise formula.

```
lemma differentialEntropy_eq_kB_beta_meanEnergy_add_kB_log_mathZ
    (T : Temperature) [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ]
    (hE : Integrable 𝒞.energy (𝒞.μProd T)) :
    𝒞.differentialEntropy T = kB * (T.β : ℝ) * 𝒞.meanEnergy T +
      kB * Real.log (𝒞.mathematicalPartitionFunction T)
```

lemma log_probability [source]

Pointwise logarithm of the Boltzmann probability.

```
lemma log_probability
    (𝒞 : CanonicalEnsemble ι) (T : Temperature)
    [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] (i : ι) :
```

```
    Real.log (𝒞.probability T i)
      = - (β T) * 𝒞.energy i - Real.log (𝒞.mathematicalPartitionFunction T)
```

lemma kB_mul_beta [source]

Auxiliary identity: `kB · β = 1 / T`. `β` is defined as `1 / (kB · T)` (see `Temperature.β`).

```
@[simp]
private lemma kB_mul_beta (T : Temperature) (hT : 0 < T.val) :
    (kB : ℝ) * (T.β : ℝ) = 1 / T.val
```

lemma thermodynamicEntropy_eq_differentialEntropy_sub_correction [source]

Fundamental relation between thermodynamic and differential entropy: `S_thermo = S_diff - kB * dof * log h`.

```
lemma thermodynamicEntropy_eq_differentialEntropy_sub_correction
    (T : Temperature)
    (hE : Integrable 𝒞.energy (𝒞.μProd T))
    [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] :
    𝒞.thermodynamicEntropy T
      = 𝒞.differentialEntropy T
        - kB * 𝒞.dof * Real.log 𝒞.phaseSpaceunit
```

lemma thermodynamicEntropy_eq_differentialEntropy_of_dof_zero [source]

No semiclassical correction when `dof = 0`.

```
lemma thermodynamicEntropy_eq_differentialEntropy_of_dof_zero
    (T : Temperature) (hE : Integrable 𝒞.energy (𝒞.μProd T))
    (h0 : 𝒞.dof = 0)
    [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] :
    𝒞.thermodynamicEntropy T = 𝒞.differentialEntropy T
```

lemma thermodynamicEntropy_eq_differentialEntropy_of_phase_space_unit_one [source]

No semiclassical correction when `phase_space_unit = 1`.

```
lemma thermodynamicEntropy_eq_differentialEntropy_of_phase_space_unit_one
    (T : Temperature) (hE : Integrable 𝒞.energy (𝒞.μProd T))
    (h1 : 𝒞.phaseSpaceunit = 1)
    [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ] :
    𝒞.thermodynamicEntropy T = 𝒞.differentialEntropy T
```

theorem helmholtzFreeEnergy_eq_meanEnergy_sub_temp_mul_thermodynamicEntropy [source]

The Helmholtz free energy `F` is related to the mean energy `U` and the absolute thermodynamic entropy `S` by the identity `F = U - TS`. This theorem shows that the statistically-defined quantities in this framework correctly satisfy this principle of thermodynamics.

```
theorem helmholtzFreeEnergy_eq_meanEnergy_sub_temp_mul_thermodynamicEntropy
    (T : Temperature) (hT : 0 < T.val)
    [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ]
    (hE : Integrable 𝒞.energy (𝒞.μProd T)) :
```

```
    𝒞.helmholtzFreeEnergy T
      = 𝒞.meanEnergy T - T.val * 𝒞.thermodynamicEntropy T
```

theorem differentialEntropy_eq_meanEnergy_sub_helmholtz_div_temp_add_correction [source]

***Theorem: Helmholtz identity with semi–classical correction term**. Physical identity (always true for `T > 0`): (U - F)/T = S_thermo and: S_thermo = S_diff - kB * dof * log h. Hence: S_diff = (U - F)/T + kB * dof * log h. This theorem gives the correct relation for the (mathematical / differential) entropy. (Removing the correction is only valid in normalized discrete cases with `dof = 0` (or `phaseSpaceUnit = 1`).)*

```
theorem differentialEntropy_eq_meanEnergy_sub_helmholtz_div_temp_add_correction
    (𝒞 : CanonicalEnsemble ι) (T : Temperature)
    [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ]
    (hT : 0 < T.val)
    (hE : Integrable 𝒞.energy (𝒞.μProd T)) :
    𝒞.differentialEntropy T
      = (𝒞.meanEnergy T - 𝒞.helmholtzFreeEnergy T) / T.val
        + kB * 𝒞.dof * Real.log 𝒞.phaseSpaceunit
```

lemma differentialEntropy_eq_meanEnergy_sub_helmholtz_div_temp [source]

Discrete / normalized specialization of the previous theorem. If either `dof = 0` (no semiclassical correction) or `phaseSpaceUnit = 1` (so `log h = 0`), the correction term vanishes and we recover the bare Helmholtz identity for the (differential) entropy.

```
lemma differentialEntropy_eq_meanEnergy_sub_helmholtz_div_temp
    (𝒞 : CanonicalEnsemble ι) (T : Temperature)
    [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ]
    (hT : 0 < T.val)
    (hE : Integrable 𝒞.energy (𝒞.μProd T))
    (hNorm : 𝒞.dof = 0 ∨ 𝒞.phaseSpaceunit = 1) :
    𝒞.differentialEntropy T
      = (𝒞.meanEnergy T - 𝒞.helmholtzFreeEnergy T) / T.val
```

lemma hasDerivWithinAt_log_comp [source]

Chain rule convenience lemma for `log ∘ f` on a set.

```
lemma hasDerivWithinAt_log_comp
    {f : ℝ → ℝ} {f' : ℝ} {s : Set ℝ} {x : ℝ}
    (hf : HasDerivWithinAt f f' s x) (hx : f x ≠ 0) :
    HasDerivWithinAt (fun t => Real.log (f t)) ((f x)⁻¹ * f') s x
```

lemma hasDerivWithinAt_log_comp' [source]

A version rewriting the derivative value with `1 / f x`.

```
lemma hasDerivWithinAt_log_comp'
    {f : ℝ → ℝ} {f' : ℝ} {s : Set ℝ} {x : ℝ}
    (hf : HasDerivWithinAt f f' s x) (hx : f x ≠ 0) :
    HasDerivWithinAt (fun t => Real.log (f t))
      ((1 / f x) * f') s x
```

lemma integral_bolt_eq_integral_mul_exp [source]

```
lemma integral_bolt_eq_integral_mul_exp
    {ι} [MeasurableSpace ι] (𝒞 : CanonicalEnsemble ι) (T : Temperature)
    (φ : ι → ℝ) :
    ∫ x, φ x ∂ 𝒞.μBolt T
      = ∫ x, φ x * Real.exp (-T.β * 𝒞.energy x) ∂ 𝒞.μ
```

lemma integral_energy_bolt [source]

A specialization of `integral_bolt_eq_integral_mul_exp` to the energy observable.

```
lemma integral_energy_bolt
    {ι} [MeasurableSpace ι] (𝒞 : CanonicalEnsemble ι) (T : Temperature) :
    ∫ x, 𝒞.energy x ∂ 𝒞.μBolt T
      = ∫ x, 𝒞.energy x * Real.exp (-T.β * 𝒞.energy x) ∂ 𝒞.μ
```

lemma meanEnergy_eq_ratio_of_integrals [source]

The mean energy can be expressed as a ratio of integrals.

```
lemma meanEnergy_eq_ratio_of_integrals
    (𝒞 : CanonicalEnsemble ι) (T : Temperature) :
    𝒞.meanEnergy T =
      (∫ i, 𝒞.energy i * Real.exp (- T.β * 𝒞.energy i) ∂ 𝒞.μ) /
        (∫ i, Real.exp (- T.β * 𝒞.energy i) ∂ 𝒞.μ)
```

lemma meanEnergy_eq_neg_deriv_log_mathZ_of_beta [source]

The mean energy is the negative derivative of the logarithm of the (mathematical) partition function with respect to $\beta = 1/(kB\ T)$. see: Tong (§1.3.2, §1.3.3), L&L (§31, implicitly, and §36) Here the derivative is a `derivWithin` over `Set.Ioi 0` since `β > 0`.

```
lemma meanEnergy_eq_neg_deriv_log_mathZ_of_beta
    (𝒞 : CanonicalEnsemble ι) (T : Temperature)
    (hT_pos : 0 < T.val) [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ]
    (h_deriv :
        HasDerivWithinAt
          (fun β : ℝ => ∫ i, Real.exp (-β * 𝒞.energy i) ∂ 𝒞.μ)
          (- ∫ i, 𝒞.energy i * Real.exp (-(T.β : ℝ) * 𝒞.energy i) ∂𝒞.μ)
          (Set.Ioi 0) (T.β : ℝ)) :
    𝒞.meanEnergy T =
      - (derivWithin
          (fun β : ℝ => Real.log (∫ i, Real.exp (-β * 𝒞.energy i) ∂𝒞.μ))
          (Set.Ioi 0) (T.β : ℝ))
```

lemma log_phys_eq_log_math_sub_const_on_Ioi [source]

*Helper: equality (on `Set.Ioi 0`) between the β–parametrized logarithm of the physical partition function and the β–parametrized logarithm of the *mathematical* partition function up to the (β–independent) semiclassical correction. This is used only to identify derivatives (the correction drops). We add the hypothesis `h_fin` giving finiteness of the Boltzmann measure for every $\beta > 0$ (as needed to ensure the mathematical partition function is strictly positive).*

```
private lemma log_phys_eq_log_math_sub_const_on_Ioi
    (𝒞 : CanonicalEnsemble ι) [NeZero 𝒞.μ]
    (h_fin :
      ∀ β > 0,
        IsFiniteMeasure (𝒞.μBolt (Temperature.ofβ (Real.toNNReal β))))
    (h_const :
        (𝒞.dof : ℝ) * Real.log 𝒞.phaseSpaceunit =
        (𝒞.dof : ℝ) * Real.log 𝒞.phaseSpaceunit := rfl) :
    Set.EqOn
      (fun β : ℝ =>
        Real.log (𝒞.partitionFunction (Temperature.ofβ (Real.toNNReal β))))
      (fun β : ℝ =>
        Real.log (∫ i, Real.exp (-β * 𝒞.energy i) ∂ 𝒞.μ)
          - (𝒞.dof : ℝ) * Real.log 𝒞.phaseSpaceunit)
      (Set.Ioi (0 : ℝ))
```

lemma derivWithin_log_phys_eq_derivWithin_log_math [source]

Derivative equality needed in `meanEnergy_eq_neg_deriv_log_Z_of_beta`. Adds `h_fin` (finiteness of the Boltzmann measure for every $\beta > 0$).

```
private lemma derivWithin_log_phys_eq_derivWithin_log_math
    (𝒞 : CanonicalEnsemble ι) (T : Temperature)
    (hT_pos : 0 < T.val)
    [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ]
    (h_integrable :
        ∀ β > 0, Integrable (fun i => Real.exp (-β * 𝒞.energy i)) 𝒞.μ)
    (h_fin :
        ∀ β > 0,
          IsFiniteMeasure (𝒞.μBolt (Temperature.ofβ (Real.toNNReal β)))) :
    derivWithin
      (fun β : ℝ => Real.log (𝒞.partitionFunction (ofβ (Real.toNNReal β))))
      (Set.Ioi 0) (T.β : ℝ)
    =
    derivWithin
      (fun β : ℝ => Real.log (∫ i, Real.exp (-β * 𝒞.energy i) ∂ 𝒞.μ))
      (Set.Ioi 0) (T.β : ℝ)
```

theorem meanEnergy_eq_neg_deriv_log_Z_of_beta [source]

*The mean energy can also be expressed as the negative derivative of the logarithm of the *physical* partition function with respect to β. This follows from the fact that the physical and mathematical partition functions differ only by a constant factor, which vanishes upon differentiation.*

```
theorem meanEnergy_eq_neg_deriv_log_Z_of_beta
    (𝒞 : CanonicalEnsemble ι) (T : Temperature)
    (hT_pos : 0 < T.val) [IsFiniteMeasure (𝒞.μBolt T)] [NeZero 𝒞.μ]
    (h_integrable :
        ∀ β > 0, Integrable (fun i => Real.exp (-β * 𝒞.energy i)) 𝒞.μ)
    (h_fin :
        ∀ β > 0,
          IsFiniteMeasure (𝒞.μBolt (Temperature.ofβ (Real.toNNReal β))))
    (h_deriv :
        HasDerivWithinAt
          (fun β : ℝ => ∫ i, Real.exp (-β * 𝒞.energy i) ∂ 𝒞.μ)
```

```
        (- ∫ i, 𝒞.energy i * Real.exp (-(T.β : ℝ) * 𝒞.energy i) ∂𝒞.μ)
        (Set.Ioi 0) (T.β : ℝ)) :
    𝒞.meanEnergy T =
      - (derivWithin
        (fun β : ℝ => Real.log (𝒞.partitionFunction (ofβ (Real.toNNReal β))))
        (Set.Ioi 0) (T.β : ℝ))
```

2.27 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 : Charges 𝒵) : Prop
```

instance Decidable (IsAnomalyFree c) [source]

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

lemma isAnomalyFree_map [source]

```
lemma isAnomalyFree_map (f : 𝒵 →+* 𝒵1) {c : Charges 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 => ∅)
```

2.28 PhysLean.StringTheory.FTheory.SU5.Charges.OfRationalSection

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

inductive CodimensionOneConfig [source]

The distinct codimension one configurations of the zero-section (σ_0) relativity to the additional rational section ($\sigma_1 s$).

```
inductive CodimensionOneConfig
  | same : CodimensionOneConfig
  | nearestNeighbor : CodimensionOneConfig
  | nextToNearestNeighbor : CodimensionOneConfig
```

instance Fintype CodimensionOneConfig [source]

```
instance : Fintype CodimensionOneConfig where
```

def allowedBarFiveCharges [source]

The allowed `U(1)`-charges of matter in the 5-bar representation of `SU(5)` given a `CodimensionOneConfig`.

```
def allowedBarFiveCharges : CodimensionOneConfig → Finset ℤ
  | same => {-3, -2, -1, 0, 1, 2, 3}
  | nearestNeighbor => {-14, -9, -4, 1, 6, 11}
  | nextToNearestNeighbor => {-13, -8, -3, 2, 7, 12}
```

instance (I : CodimensionOneConfig) → Fintype I.allowedBarFiveCharges | same =>… [source]

```
instance : (I : CodimensionOneConfig) → Fintype I.allowedBarFiveCharges
  | same => inferInstance
  | nearestNeighbor => inferInstance
  | nextToNearestNeighbor => inferInstance
```

def allowedBarFiveChargesList [source]

The allowed `U(1)`-charges of matter in the 5-bar representation of `SU(5)` given a `CodimensionOneConfig`, as ordered lists.

```
def allowedBarFiveChargesList : CodimensionOneConfig → List ℤ
  | same => [-3, -2, -1, 0, 1, 2, 3]
  | nearestNeighbor => [-14, -9, -4, 1, 6, 11]
  | nextToNearestNeighbor => [-13, -8, -3, 2, 7, 12]
```

lemma mem_allowedBarFiveChargesList_iff [source]

```
@[simp]
lemma mem_allowedBarFiveChargesList_iff {I : CodimensionOneConfig} (x : ℤ) :
    x ∈ I.allowedBarFiveChargesList ↔ x ∈ I.allowedBarFiveCharges
```

lemma allowedBarFiveChargesList_nodup [source]

```
lemma allowedBarFiveChargesList_nodup (I : CodimensionOneConfig) :
    I.allowedBarFiveChargesList.Nodup
```

def allowedTenCharges [source]

The allowed `U(1)`-charges of matter in the 10d representation of `SU(5)` given a `CodimensionOneConfig`.

```
def allowedTenCharges : CodimensionOneConfig → Finset ℤ
  | same => {-3, -2, -1, 0, 1, 2, 3}
  | nearestNeighbor => {-12, -7, -2, 3, 8, 13}
  | nextToNearestNeighbor => {-9, -4, 1, 6, 11}
```

instance (I : CodimensionOneConfig) → Fintype I.allowedTenCharges | same => infe… [source]

```
instance : (I : CodimensionOneConfig) → Fintype I.allowedTenCharges
  | same => inferInstance
  | nearestNeighbor => inferInstance
  | nextToNearestNeighbor => inferInstance
```

def allowedTenChargesList [source]

The allowed `U(1)`-charges of matter in the 10d representation of `SU(5)` given a `CodimensionOneConfig`, as ordered lists.

```
def allowedTenChargesList : CodimensionOneConfig → List ℤ
  | same => [-3, -2, -1, 0, 1, 2, 3]
  | nearestNeighbor => [-12, -7, -2, 3, 8, 13]
  | nextToNearestNeighbor => [-9, -4, 1, 6, 11]
```

lemma mem_allowedTenChargesList_iff [source]

```
@[simp]
lemma mem_allowedTenChargesList_iff {I : CodimensionOneConfig} (x : ℤ) :
    x ∈ I.allowedTenChargesList ↔ x ∈ I.allowedTenCharges
```

lemma allowedTenChargesList_nodup [source]

```
lemma allowedTenChargesList_nodup (I : CodimensionOneConfig) :
    I.allowedTenChargesList.Nodup
```

def fiveChargeMultisetToList [source]

Given a multiset `S : Multiset ℤ` for which all elements are members of `I.allowedBarFiveCharges`, `fiveChargeMultisetToList I S` is a computable list whose underlying multiset is `S`.

```
def fiveChargeMultisetToList (I : CodimensionOneConfig) (S : Multiset ℤ) : List ℤ
```

def tenChargeMultisetToList [source]

Given a multiset `S : Multiset ℤ` for which all elements are members of `I.allowedTenCharges`, `tenChargeMultisetToList I S` is a computable list whose underlying multiset is `S`.

```
def tenChargeMultisetToList (I : CodimensionOneConfig) (S : Multiset ℤ) : List ℤ
```

lemma fiveChargeMultisetToList_mem_iff [source]

```
lemma fiveChargeMultisetToList_mem_iff {I : CodimensionOneConfig} {S : Multiset ℤ} {a : ℤ}:
    a ∈ fiveChargeMultisetToList I S ↔ (a ∈ S ∧ a ∈ I.allowedBarFiveCharges)
```

lemma tenChargeMultisetToList_mem_iff [source]

```
lemma tenChargeMultisetToList_mem_iff {I : CodimensionOneConfig} {S : Multiset ℤ} {a : ℤ}:
    a ∈ tenChargeMultisetToList I S ↔ (a ∈ S ∧ a ∈ I.allowedTenCharges)
```

lemma fiveChargeMultisetToList_count [source]

```
lemma fiveChargeMultisetToList_count {I : CodimensionOneConfig} {S : Multiset ℤ} {a : ℤ}
    (hmem : a ∈ I.allowedBarFiveCharges) :
    (fiveChargeMultisetToList I S).count a = S.count a
```

lemma tenChargeMultisetToList_count [source]

```
lemma tenChargeMultisetToList_count {I : CodimensionOneConfig} {S : Multiset ℤ} {a : ℤ}
    (hmem : a ∈ I.allowedTenCharges) :
    (tenChargeMultisetToList I S).count a = S.count a
```

lemma coe_fiveChargeMultisetToList_of_all_mem [source]

```
lemma coe_fiveChargeMultisetToList_of_all_mem (I : CodimensionOneConfig) (S : Multiset ℤ)
    (hs : ∀ s ∈ S, s ∈ I.allowedBarFiveCharges) :
    ↑(fiveChargeMultisetToList I S) = S
```

lemma coe_tenChargeMultisetToList_of_all_mem [source]

```
lemma coe_tenChargeMultisetToList_of_all_mem (I : CodimensionOneConfig) (S : Multiset ℤ)
    (hs : ∀ s ∈ S, s ∈ I.allowedTenCharges) :
    ↑(tenChargeMultisetToList I S) = S
```

lemma fiveChargeMultisetToList_length [source]

```
lemma fiveChargeMultisetToList_length (I : CodimensionOneConfig) (S : Multiset ℤ)
    (hs : ∀ s ∈ S, s ∈ I.allowedBarFiveCharges) :
    (fiveChargeMultisetToList I S).length = S.card
```

lemma tenChargeMultisetToList_length [source]

```
lemma tenChargeMultisetToList_length (I : CodimensionOneConfig) (S : Multiset ℤ)
    (hs : ∀ s ∈ S, s ∈ I.allowedTenCharges) :
    (I.tenChargeMultisetToList S).length = S.card
```

2.29 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 (Charges 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 (Charges 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 (Charges 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.2.2.1.card ≤ 2
```

lemma card_ten_le_of_mem_viableCharges [source]

```
lemma card_ten_le_of_mem_viableCharges (I : CodimensionOneConfig) :
    ∀ x ∈ (viableCharges I), x.2.2.2.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 : Charges}
    (hsub : x ∈ ofFinset I.allowedBarFiveCharges I.allowedTenCharges) :
    x ∈ viableCharges I ↔ AllowsTerm x topYukawa ∧
    ¬ IsPhenoConstrained x ∧ ¬ YukawaGeneratesDangerousAtLevel x 1 ∧ IsComplete x
```

2.30 PhysLean.StringTheory.FTheory.SU5.Fluxes.Basic

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

lemma numChiralD_eq_sum_sub_numAntiChiralD [source]

```
lemma numChiralD_eq_sum_sub_numAntiChiralD (F : FluxesFive) :
    F.numChiralD = (chiralIndicesOfD F).sum - F.numAntiChiralD
```

lemma numChiralL_eq_sum_sub_numAntiChiralL [source]

```
lemma numChiralL_eq_sum_sub_numAntiChiralL (F : FluxesFive) :
    F.numChiralL = (chiralIndicesOfL F).sum - F.numAntiChiralL
```

lemma numChiralQ_eq_sum_sub_numAntiChiralQ [source]

```
lemma numChiralQ_eq_sum_sub_numAntiChiralQ (F : FluxesTen) :
    F.numChiralQ = (chiralIndicesOfQ F).sum - F.numAntiChiralQ
```

lemma numChiralU_eq_sum_sub_numAntiChiralU [source]

```
lemma numChiralU_eq_sum_sub_numAntiChiralU (F : FluxesTen) :
    F.numChiralU = (chiralIndicesOfU F).sum - F.numAntiChiralU
```

lemma numChiralE_eq_sum_sub_numAntiChiralE [source]

```
lemma numChiralE_eq_sum_sub_numAntiChiralE (F : FluxesTen) :
    F.numChiralE = (chiralIndicesOfE F).sum - F.numAntiChiralE
```

2.31 PhysLean.StringTheory.FTheory.SU5.Fluxes.NoExotics.Elems

PhysLean/StringTheory/FTheory/SU5/Fluxes/NoExotics/Elems.lean on GitHub.

lemma map_sum_add_of_mem_powerset_elemsNoExotics [source]

```
lemma map_sum_add_of_mem_powerset_elemsNoExotics (F S : FluxesFive)
    (hf : F ∈ FluxesFive.elemsNoExotics)
    (hS: S ∈ Multiset.powerset F) :
    (S.map (fun x => (|x.1|, -|x.1|))).sum +
    (S.map (fun x => (0, |x.1 + x.2|))).sum = S.sum
```

2.32 PhysLean.StringTheory.FTheory.SU5.Quanta.Basic

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

abbrev Quanta [source]

The quanta associated with the representations in a `SU(5) x U(1)` F-theory. This contains the value of the charges and the flux intergers `(M, N)` for the 5-bar matter content and the 10d matter content, and the charges of the `Hd` and `Hu` particles (there values of `(M,N)` are not included as they are forced to be `(0, 1)` and `(0, -1)` respectively.

```
abbrev Quanta (𝒵 : Type := ℤ) : Type
```

instance DecidableEq (Quanta 𝒵) [source]

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

def toCharges [source]

The underlying `Charges` of a `Quanta`.

```
def toCharges [DecidableEq 𝒵] (x : Quanta 𝒵) : Charges 𝒵
```

def reduce [source]

The reduce of `Quanta` is a new `Quanta` with all the fluxes corresponding to the same charge (i.e. represenation) added together.

```
def reduce [DecidableEq 𝒵] (x : Quanta 𝒵) : Quanta 𝒵
```

def HdAnomalyCoefficent [source]

The pair of anomaly cancellation coefficents associated with the `Hd` particle.

```
def HdAnomalyCoefficent [CommRing 𝒵] (qHd : Option 𝒵) : 𝒵 × 𝒵
```

lemma HdAnomalyCoefficent_map [source]

```
@[simp]
lemma HdAnomalyCoefficent_map {𝒵 𝒵1 : Type} [CommRing 𝒵] [CommRing 𝒵1]
    (f : 𝒵 →+* 𝒵1) (qHd : Option 𝒵) :
    HdAnomalyCoefficent (qHd.map f) = (f.prodMap f) (HdAnomalyCoefficent qHd)
```

def HuAnomalyCoefficent [source]

The pair of anomaly cancellation coefficents associated with the `Hu` particle.

```
def HuAnomalyCoefficent [CommRing ℤ] (qHu : Option ℤ) : ℤ × ℤ
```

lemma HuAnomalyCoefficent_map [source]

```
@[simp]
lemma HuAnomalyCoefficent_map {ℤ ℤ1 : Type} [CommRing ℤ] [CommRing ℤ1]
    (f : ℤ →+* ℤ1) (qHu : Option ℤ) :
    HuAnomalyCoefficent (qHu.map f) = (f.prodMap f) (HuAnomalyCoefficent qHu)
```

def AnomalyCancellation [source]

The anomaly cancellation conditions on quanta making up the fields present in the theory. This corresponds to the conditions that:

- `∑ᵢ qᵢ Nᵢ + ∑ₐ qₐ Nₐ = 0` where the first sum is over all 5-bar represenations and the second is over all 10d representations. - `∑ᵢ qᵢ² Nᵢ + 3 * ∑ₐ qₐ² Nₐ = 0` where the first sum is over all 5-bar represenations and the second is over all 10d representations.

```
def AnomalyCancellation [CommRing ℤ] (qHd qHu : Option ℤ) (F : FiveQuanta ℤ) (T : TenQuanta
    ℤ) :
    Prop
```

instance Decidable (AnomalyCancellation (ℤ [source]

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

lemma anomalyCoefficent_snd_eq_zero_of_anomalyCancellation [source]

```
lemma anomalyCoefficent_snd_eq_zero_of_anomalyCancellation [CommRing ℤ]
    {qHd qHu : Option ℤ} {F : FiveQuanta ℤ} {T : TenQuanta ℤ}
    (h : AnomalyCancellation qHd qHu F T) :
    ((HdAnomalyCoefficent qHd).2 + (HuAnomalyCoefficent qHu).2
    + (F.anomalyCoefficent).2 + (T.anomalyCoefficent).2) = 0
```

def ofChargesExpand [source]

Given a finite set of charges `c` the `Quanta` with fluxes `{(1, -1), (1, -1), (1, -1), (0, 1), (0, 1), (0, 1)}` for the 5-bar matter content and fluxes `{(1, 0), (1, 0), (1, 0)}` or `{(1, 1), (1, -1), (1, 0)}` for the 10d matter content, and finite set of charges equal to `c`.

These quanta reduce to all viable quanta.

```
def ofChargesExpand [DecidableEq ℤ] (c : Charges ℤ) : Multiset (Quanta ℤ)
```

2.33 PhysLean.StringTheory.FTheory.SU5.Quanta.FiveQuanta

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

abbrev FiveQuanta [source]

The quanta of 5-bar representations corresponding to a multiset of `(q, M, N)` for each partcile. `(M, N)` are defined in the `FluxesFive` module.

```
abbrev FiveQuanta (𝒵 : Type := ℤ) : Type
```

def toFluxesFive [source]

The underlying `FluxesFive` from a `FiveQuanta`.

```
def toFluxesFive (x : FiveQuanta 𝒵) : FluxesFive
```

def toCharges [source]

The underlying Multiset charges from a `FiveQuanta`.

```
def toCharges (x : FiveQuanta 𝒵) : Multiset 𝒵
```

def reduce [source]

The `reduce` of `FiveQuanta` is a new `FiveQuanta` with all the fluxes corresponding to the same charge (i.e. represenation) added together.

```
def reduce (x : FiveQuanta 𝒵) : FiveQuanta 𝒵
```

lemma reduce_nodup [source]

```
lemma reduce_nodup (x : FiveQuanta 𝒵) : x.reduce.Nodup
```

lemma reduce_dedup [source]

```
@[simp]
lemma reduce_dedup (x : FiveQuanta 𝒵) : x.reduce.dedup = x.reduce
```

lemma reduce_toCharges [source]

```
lemma reduce_toCharges (x : FiveQuanta 𝒵) : x.reduce.toCharges = x.toCharges.dedup
```

lemma reduce_eq_val [source]

```
lemma reduce_eq_val (x : FiveQuanta 𝒵) :
    x.reduce = (x.toCharges.toFinset.image fun q5 =>
      (q5, ((x.filter (fun f => f.1 = q5)).map (fun y => y.2)).sum)).val
```

lemma mem_reduce_iff [source]

```
lemma mem_reduce_iff (x : FiveQuanta 𝒵) (p : 𝒵 × ℤ × ℤ) :
    p ∈ x.reduce ↔ p.1 ∈ x.toCharges ∧
      p.2 = ((x.filter (fun f => f.1 = p.1)).map (fun y => y.2)).sum
```

lemma reduce_filter [source]

```
lemma reduce_filter (x : FiveQuanta ℤ) (q : ℤ) (h : q ∈ x.toCharges) :
    x.reduce.filter (fun f => f.1 = q) =
    {(q, ((x.filter (fun f => f.1 = q)).map (fun y => y.2)).sum)}
```

lemma reduce_reduce [source]

```
@[simp]
lemma reduce_reduce (x : FiveQuanta ℤ) :
    x.reduce.reduce = x.reduce
```

lemma reduce_sum_eq_sum_toCharges [source]

```
lemma reduce_sum_eq_sum_toCharges {M} [AddCommMonoid M] (x : FiveQuanta ℤ) (f : ℤ → ℤ × ℤ →+ M
    ) :
    (x.reduce.map fun (q5, x) => f q5 x).sum = (x.map fun (q5, x) => f q5 x).sum
```

lemma reduce_eq_self_of_ofCharges_nodup [source]

```
lemma reduce_eq_self_of_ofCharges_nodup (x : FiveQuanta ℤ) (h : x.toCharges.Nodup) :
    x.reduce = x
```

def anomalyCoefficent [source]

The anomaly coefficent of a `FiveQuanta` is given by the pair of integers: `(∑ᵢ qᵢ Nᵢ, ∑ᵢ qᵢ² Nᵢ)`.

The first components is for the mixed U(1)-MSSM, see equation (22) of arXiv:1401.5084. The second component is for the mixed U(1)Y-U(1)-U(1) gauge anomaly, see equation (23) of arXiv:1401.5084.

```
def anomalyCoefficent (F : FiveQuanta ℤ) : ℤ × ℤ
```

lemma anomalyCoefficent_of_map [source]

```
@[simp]
lemma anomalyCoefficent_of_map {ℤ ℤ1 : Type} [CommRing ℤ] [CommRing ℤ1]
    (f : ℤ →+* ℤ1) (F : FiveQuanta ℤ) :
    FiveQuanta.anomalyCoefficent (F.map fun y => (f y.1, y.2) : FiveQuanta ℤ1) =
    (f.prodMap f) F.anomalyCoefficent
```

lemma anomalyCoefficent_of_reduce [source]

```
lemma anomalyCoefficent_of_reduce (F : FiveQuanta ℤ) [DecidableEq ℤ] :
    F.reduce.anomalyCoefficent = F.anomalyCoefficent
```

def ofChargesExpand [source]

Given a finite set of charges `c` the `FiveQuanta` with fluxes `{(1, -1), (1, -1), (1, -1), (0, 1), (0, 1), (0, 1)}` and finite set of charges equal to `c`.

```
def ofChargesExpand (c : Finset ℤ) : Multiset (FiveQuanta ℤ)
```

lemma toFluxesFive_of_mem_ofChargesExpand [source]

```
lemma toFluxesFive_of_mem_ofChargesExpand (c : Finset ℤ)
    {x : FiveQuanta ℤ} (h : x ∈ ofChargesExpand c) :
    x.toFluxesFive = {(1, -1), (1, -1), (1, -1), (0, 1), (0, 1), (0, 1)}
```

lemma toCharges_of_mem_ofChargesExpand [source]

```
lemma toCharges_of_mem_ofChargesExpand (c : Finset ℤ)
    {x : FiveQuanta ℤ} (h : x ∈ ofChargesExpand c) :
    x.toCharges.toFinset = c
```

lemma mem_ofChargesExpand_of_toCharges_toFluxesFive [source]

```
lemma mem_ofChargesExpand_of_toCharges_toFluxesFive (c : Finset ℤ) {x : FiveQuanta ℤ}
    (h : x.toCharges.toFinset = c) (h2 : x.toFluxesFive =
      {(1, -1), (1, -1), (1, -1), (0, 1), (0, 1), (0, 1)}) :
    x ∈ ofChargesExpand c
```

lemma mem_ofChargesExpand_iff [source]

```
lemma mem_ofChargesExpand_iff(c : Finset ℤ) {x : FiveQuanta ℤ} :
    x ∈ ofChargesExpand c ↔
    x.toCharges.toFinset = c ∧ x.toFluxesFive =
      {(1, -1), (1, -1), (1, -1), (0, 1), (0, 1), (0, 1)}
```

lemma eq_sum_filter_of_mem_ofChargesExpand [source]

```
lemma eq_sum_filter_of_mem_ofChargesExpand (c : Finset ℤ) (F : FiveQuanta ℤ)
    (h : F ∈ ofChargesExpand c) :
    F = (F.filter fun x => x.2 = (1, -1)) + (F.filter fun x => x.2 = (0, 1))
```

lemma mem_ofChargesExpand_of_noExotics_hasNoZero [source]

```
lemma mem_ofChargesExpand_of_noExotics_hasNoZero (F : FiveQuanta ℤ) (c : Finset ℤ)
    (hc : F.toCharges.toFinset = c)
    (h1 : F.toFluxesFive.NoExotics) (h2 : F.toFluxesFive.HasNoZero) :
    ∃ y ∈ ofChargesExpand c, y.reduce = F.reduce
```

lemma exists_charges_of_mem_ofChargesExpand [source]

```
lemma exists_charges_of_mem_ofChargesExpand (c : Finset ℤ) (F : FiveQuanta ℤ)
    (h : F ∈ ofChargesExpand c) :
    ∃ q1 q2 q3 q4 q5 q6 : ℤ,
      F = {(q1, 1, -1), (q2, 1, -1), (q3, 1, -1), (q4, 0, 1), (q5, 0, 1), (q6, 0, 1)}
```

lemma exists_charges_le_of_mem_ofChargesExpand [source]

```
lemma exists_charges_le_of_mem_ofChargesExpand (c : Finset ℤ) (F : FiveQuanta ℤ)
    (h : F ∈ ofChargesExpand c) :
    ∃ q1 q2 q3 q4 q5 q6 : ℤ,
      F = {(q1, 1, -1), (q2, 1, -1), (q3, 1, -1), (q4, 0, 1), (q5, 0, 1), (q6, 0, 1)} ∧
      q1 ≤ q2 ∧ q2 ≤ q3 ∧ q4 ≤ q5 ∧ q5 ≤ q6
```

lemma reduce_hasNoZeros_of_mem_ofChargesExpand [source]

```
lemma reduce_hasNoZeros_of_mem_ofChargesExpand (c : Finset ℤ) (F : FiveQuanta ℤ)
    (h : F ∈ ofChargesExpand c) :
    F.reduce.toFluxesFive.HasNoZero
```

lemma reduce_noExotics_of_mem_ofChargesExpand [source]

```
lemma reduce_noExotics_of_mem_ofChargesExpand (c : Finset ℤ) (F : FiveQuanta ℤ)
    (h : F ∈ ofChargesExpand c) :
    F.reduce.toFluxesFive.NoExotics
```

lemma reduce_mem_elemsNoExotics_of_mem_ofChargesExpand [source]

```
lemma reduce_mem_elemsNoExotics_of_mem_ofChargesExpand (c : Finset ℤ) (F : FiveQuanta ℤ)
    (h : F ∈ ofChargesExpand c) :
    F.reduce.toFluxesFive ∈ FluxesFive.elemsNoExotics
```

lemma mem_ofChargesExpand_map_reduce_iff [source]

```
lemma mem_ofChargesExpand_map_reduce_iff (c : Finset ℤ) (S : FiveQuanta ℤ) :
    S ∈ (ofChargesExpand c).map reduce ↔ S.toFluxesFive ∈ FluxesFive.elemsNoExotics
      ∧ S.toCharges.toFinset = c ∧ S.reduce = S
```

2.34 PhysLean.StringTheory.FTheory.SU5.Quanta.IsViable

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

def IsViable [source]

For a given `I : CodimensionOneConfig` the condition on a `Quanta` for it to be phenomenologically viable.

```
def IsViable (I : CodimensionOneConfig) (x : Quanta) : Prop
```

lemma isViable_iff_expand_ofFinset [source]

```
lemma isViable_iff_expand_ofFinset (I : CodimensionOneConfig) (x : Quanta) :
    IsViable I x ↔
      x.toCharges.IsComplete ∧
  ¬ x.toCharges.IsPhenoConstrained ∧
  ¬ x.toCharges.YukawaGeneratesDangerousAtLevel 1 ∧
  AllowsTerm x.toCharges topYukawa ∧
  x.2.2.1.toFluxesFive.NoExotics ∧
  x.2.2.1.toFluxesFive.HasNoZero ∧
  x.2.2.2.toFluxesTen.NoExotics ∧
  x.2.2.2.toFluxesTen.HasNoZero ∧
  AnomalyCancellation x.1 x.2.1 x.2.2.1 x.2.2.2 ∧
  (x.1.toFinset ⊆ I.allowedBarFiveCharges ∧ x.2.1.toFinset ⊆ I.allowedBarFiveCharges ∧
      x.toCharges.2.2.1 ⊆ I.allowedBarFiveCharges ∧ x.toCharges.2.2.2 ⊆ I.allowedTenCharges)
      ∧
    x.2.2.1.toCharges.Nodup ∧
    x.2.2.2.toCharges.Nodup
```

instance Decidable (IsViable I x) [source]

```
instance (I : CodimensionOneConfig) (x : Quanta) : Decidable (IsViable I x)
```

lemma toCharges_five_nodup_of_isViable [source]

```
lemma toCharges_five_nodup_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    x.2.2.1.toCharges.Nodup
```

lemma toCharges_ten_nodup_of_isViable [source]

```
lemma toCharges_ten_nodup_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    x.2.2.2.toCharges.Nodup
```

lemma toCharges_mem_ofFinset_of_isViable [source]

```
lemma toCharges_mem_ofFinset_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    x.toCharges ∈ ofFinset I.allowedBarFiveCharges I.allowedTenCharges
```

lemma toFluxesFive_noExotics_of_isViable [source]

```
lemma toFluxesFive_noExotics_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    x.2.2.1.toFluxesFive.NoExotics
```

lemma toFluxesTen_noExotics_of_isViable [source]

```
lemma toFluxesTen_noExotics_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    x.2.2.2.toFluxesTen.NoExotics
```

lemma toFluxesFive_hasNoZero_of_isViable [source]

```
lemma toFluxesFive_hasNoZero_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    x.2.2.1.toFluxesFive.HasNoZero
```

lemma toFluxesTen_hasNoZero_of_isViable [source]

```
lemma toFluxesTen_hasNoZero_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    x.2.2.2.toFluxesTen.HasNoZero
```

lemma Q10_charges_mem_allowedBarTenCharges_of_isViable [source]

```
lemma Q10_charges_mem_allowedBarTenCharges_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    ∀ s ∈ x.2.2.2.toCharges, s ∈ I.allowedTenCharges
```

lemma Q5_charges_mem_allowedBarFiveCharges_of_isViable [source]

```
lemma Q5_charges_mem_allowedBarFiveCharges_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    ∀ s ∈ x.2.2.1.toCharges, s ∈ I.allowedBarFiveCharges
```

lemma topYukawa_allowsTerm_of_isViable [source]

```
lemma topYukawa_allowsTerm_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    AllowsTerm x.toCharges topYukawa
```

lemma not_isPhenoConstrained_of_isViable [source]

```
lemma not_isPhenoConstrained_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    ¬ x.toCharges.IsPhenoConstrained
```

lemma toCharges_isComplete_of_isViable [source]

```
lemma toCharges_isComplete_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    x.toCharges.IsComplete
```

lemma anomalyCancellation_of_isViable [source]

```
lemma anomalyCancellation_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    AnomalyCancellation x.1 x.2.1 x.2.2.1 x.2.2.2
```

lemma reduce_five_eq_self_of_isViable [source]

```
lemma reduce_five_eq_self_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    x.2.2.1.reduce = x.2.2.1
```

lemma reduce_ten_eq_self_of_isViable [source]

```
lemma reduce_ten_eq_self_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    x.2.2.2.reduce = x.2.2.2
```

lemma reduce_eq_self_of_isViable [source]

```
lemma reduce_eq_self_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    x.reduce = x
```

lemma mem_ofChargesExpand_of_isViable [source]

```
lemma mem_ofChargesExpand_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
```

```
    x ∈ (ofChargesExpand x.toCharges).map reduce
```

lemma toCharges_isAnomalyFree_of_isViable [source]

```
lemma toCharges_isAnomalyFree_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    IsAnomalyFree x.toCharges
```

lemma toCharges_mem_viableCharges_filter_isAnomalyFree_of_isViable [source]

```
lemma toCharges_mem_viableCharges_filter_isAnomalyFree_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    x.toCharges ∈ (viableCharges I).filter IsAnomalyFree
```

def viableElems [source]

Given a `CodimensionOneConfig` the `Quanta` which statisfy the condition `IsViable`.

```
def viableElems : CodimensionOneConfig → Multiset Quanta
  | .same => {(some 2, some (-2), {(-1, 1, 2), (1, 2, -2)}, {(-1, 3, 0)}),
      (some 2, some (-2), {(-1, 0, 2), (1, 3, -2)}, {(-1, 3, 0)}),
      (some (-2), some 2, {(-1, 2, -2), (1, 1, 2)}, {(1, 3, 0)}),
      (some (-2), some 2, {(-1, 3, -2), (1, 0, 2)}, {(1, 3, 0)})}
  | .nearestNeighbor => {(some 6, some (-14), {(-9, 1, 2), (1, 2, -2)}, {(-7, 3, 0)}),
    (some 6, some (-14), {(-9, 0, 2), (1, 3, -2)}, {(-7, 3, 0)})}
  | .nextToNearestNeighbor => {}
```

lemma isViable_of_mem_viableElems [source]

```
lemma isViable_of_mem_viableElems
    (I : CodimensionOneConfig) (x : Quanta) (h : x ∈ viableElems I) :
    IsViable I x
```

lemma viableElems_card_eq [source]

```
lemma viableElems_card_eq (I : CodimensionOneConfig) :
    (viableElems I).card = match I with
    | .same => 4
    | .nearestNeighbor => 2
    | .nextToNearestNeighbor => 0
```

lemma mem_viableElems_of_isViable [source]

```
lemma mem_viableElems_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    x ∈ viableElems I
```

lemma isViable_iff_mem_viableElems [source]

```
lemma isViable_iff_mem_viableElems
    (I : CodimensionOneConfig) (x : Quanta) :
    IsViable I x ↔ x ∈ viableElems I
```

lemma yukawaSingletsRegenerateDangerousInsertion_two_of_isViable [source]

Every viable Quanta regenerates a dangerous coupling at two insertions of the Yukawa singlets.

```
lemma yukawaSingletsRegenerateDangerousInsertion_two_of_isViable
    (I : CodimensionOneConfig) (x : Quanta) (h : IsViable I x) :
    (toCharges x).YukawaGeneratesDangerousAtLevel 2
```

2.35 PhysLean.StringTheory.FTheory.SU5.Quanta.TenQuanta

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

abbrev TenQuanta [source]

The quanta of w0d representations corresponding to a multiset of `(q, M, N)` for each partcile. `(M, N)` are defined in the `FluxesFive` module.

```
abbrev TenQuanta (𝓩 : Type := ℤ) : Type
```

def toFluxesTen [source]

The underlying `FluxesTen` from a `TenQuanta`.

```
def toFluxesTen (x : TenQuanta 𝓩) : FluxesTen
```

def toCharges [source]

The underlying Multiset charges from a `TenQuanta`.

```
def toCharges (x : TenQuanta 𝓩) : Multiset 𝓩
```

def reduce [source]

The `reduce` of `TenQuanta` is a new `TenQuanta` with all the fluxes corresponding to the same charge (i.e. represenation) added together.

```
def reduce (x : TenQuanta 𝓩) : TenQuanta 𝓩
```

lemma reduce_nodup [source]

```
lemma reduce_nodup (x : TenQuanta 𝓩) : x.reduce.Nodup
```

lemma reduce_dedup [source]

```
@[simp]
lemma reduce_dedup (x : TenQuanta 𝓩) : x.reduce.dedup = x.reduce
```

lemma reduce_toCharges [source]

```
lemma reduce_toCharges (x : TenQuanta 𝓩) : x.reduce.toCharges = x.toCharges.dedup
```

lemma reduce_eq_val [source]

```
lemma reduce_eq_val (x : TenQuanta 𝒵) :
    x.reduce = (x.toCharges.toFinset.image fun q10 =>
      (q10, ((x.filter (fun f => f.1 = q10)).map (fun y => y.2)).sum)).val
```

lemma mem_reduce_iff [source]

```
lemma mem_reduce_iff (x : TenQuanta 𝒵) (p : 𝒵 × ℤ × ℤ) :
    p ∈ x.reduce ↔ p.1 ∈ x.toCharges ∧
      p.2 = ((x.filter (fun f => f.1 = p.1)).map (fun y => y.2)).sum
```

lemma reduce_filter [source]

```
lemma reduce_filter (x : TenQuanta 𝒵) (q : 𝒵) (h : q ∈ x.toCharges) :
    x.reduce.filter (fun f => f.1 = q) =
    {(q, ((x.filter (fun f => f.1 = q)).map (fun y => y.2)).sum)}
```

lemma reduce_reduce [source]

```
@[simp]
lemma reduce_reduce (x : TenQuanta 𝒵) :
    x.reduce.reduce = x.reduce
```

lemma reduce_sum_eq_sum_toCharges [source]

```
lemma reduce_sum_eq_sum_toCharges {M} [AddCommMonoid M] (x : TenQuanta 𝒵) (f : 𝒵 → ℤ × ℤ →+ M)
     :
    (x.reduce.map fun (q, x) => f q x).sum = (x.map fun (q, x) => f q x).sum
```

lemma reduce_eq_self_of_ofCharges_nodup [source]

```
lemma reduce_eq_self_of_ofCharges_nodup (x : TenQuanta 𝒵) (h : x.toCharges.Nodup) :
    x.reduce = x
```

def anomalyCoefficent [source]

The anomaly coefficent of a `TenQuanta` is given by the pair of integers: `(∑ᵢ qᵢ Nᵢ, 3 * ∑ᵢ qᵢ² Nᵢ)`.

The first components is for the mixed U(1)-MSSM, see equation (22) of arXiv:1401.5084. The second component is for the mixed U(1)Y-U(1)-U(1) gauge anomaly, see equation (23) of arXiv:1401.5084.

```
def anomalyCoefficent (F : TenQuanta 𝒵) : 𝒵 × 𝒵
```

lemma anomalyCoefficent_of_map [source]

```
@[simp]
lemma anomalyCoefficent_of_map {𝒵 𝒵1 : Type} [CommRing 𝒵] [CommRing 𝒵1]
    (f : 𝒵 →+* 𝒵1) (F : TenQuanta 𝒵) :
    TenQuanta.anomalyCoefficent (F.map fun y => (f y.1, y.2) : TenQuanta 𝒵1) =
    (f.prodMap f) F.anomalyCoefficent
```

lemma anomalyCoefficent_of_reduce [source]

```
lemma anomalyCoefficent_of_reduce [DecidableEq ℤ] (F : TenQuanta ℤ) :
    F.reduce.anomalyCoefficent = F.anomalyCoefficent
```

def ofChargesExpand [source]

Given a finite set of charges `c` the `TenQuanta` with fluxes `{(1, 0), (1, 0), (1, 0)}` and `{(1, 1), (1, -1), (1, 0)}` and finite set of charges equal to `c`.

```
def ofChargesExpand (c : Finset ℤ) : Multiset (TenQuanta ℤ)
```

lemma toFluxesFive_of_mem_ofChargesExpand [source]

```
lemma toFluxesFive_of_mem_ofChargesExpand (c : Finset ℤ)
    {x : TenQuanta ℤ} (h : x ∈ ofChargesExpand c) :
    x.toFluxesTen = {(1, 0), (1, 0), (1, 0)}
    ∨ x.toFluxesTen = {(1, 1), (1, -1), (1, 0)}
```

lemma toCharges_of_mem_ofChargesExpand [source]

```
lemma toCharges_of_mem_ofChargesExpand (c : Finset ℤ)
    {x : TenQuanta ℤ} (h : x ∈ ofChargesExpand c) :
    x.toCharges.toFinset = c
```

lemma mem_ofChargesExpand_of_toCharges_toFluxesTen [source]

```
lemma mem_ofChargesExpand_of_toCharges_toFluxesTen (c : Finset ℤ) {x : TenQuanta ℤ}
    (h : x.toCharges.toFinset = c) (h2 : x.toFluxesTen =
      {(1, 0), (1, 0), (1, 0)} ∨ x.toFluxesTen = {(1, 1), (1, -1), (1, 0)}) :
    x ∈ ofChargesExpand c
```

lemma mem_ofChargesExpand_iff [source]

```
lemma mem_ofChargesExpand_iff (c : Finset ℤ) {x : TenQuanta ℤ} :
    x ∈ ofChargesExpand c ↔
    x.toCharges.toFinset = c ∧ (x.toFluxesTen = {(1, 0), (1, 0), (1, 0)}
    ∨ x.toFluxesTen = {(1, 1), (1, -1), (1, 0)})
```

lemma mem_ofChargesExpand_of_noExotics_hasNoZero [source]

```
lemma mem_ofChargesExpand_of_noExotics_hasNoZero (F : TenQuanta ℤ) (c : Finset ℤ)
    (hc : F.toCharges.toFinset = c)
    (h1 : F.toFluxesTen.NoExotics) (h2 : F.toFluxesTen.HasNoZero) :
    ∃ y ∈ ofChargesExpand c, y.reduce = F.reduce
```

lemma reduce_hasNoZeros_of_mem_ofChargesExpand [source]

```
lemma reduce_hasNoZeros_of_mem_ofChargesExpand (c : Finset ℤ) (F : TenQuanta ℤ)
    (h : F ∈ ofChargesExpand c) :
    F.reduce.toFluxesTen.HasNoZero
```

lemma reduce_noExotics_of_mem_ofChargesExpand [source]

```
lemma reduce_noExotics_of_mem_ofChargesExpand (c : Finset 𝒵) (F : TenQuanta 𝒵)
    (h : F ∈ ofChargesExpand c) :
    F.reduce.toFluxesTen.NoExotics
```

lemma reduce_mem_elemsNoExotics_of_mem_ofChargesExpand [source]

```
lemma reduce_mem_elemsNoExotics_of_mem_ofChargesExpand (c : Finset 𝒵) (F : TenQuanta 𝒵)
    (h : F ∈ ofChargesExpand c) :
    F.reduce.toFluxesTen ∈ FluxesTen.elemsNoExotics
```

lemma mem_ofChargesExpand_map_reduce_iff [source]

```
lemma mem_ofChargesExpand_map_reduce_iff (c : Finset 𝒵) (S : TenQuanta 𝒵) :
    S ∈ (ofChargesExpand c).map reduce ↔ S.toFluxesTen ∈ FluxesTen.elemsNoExotics
      ∧ S.toCharges.toFinset = c ∧ S.reduce = S
```

2.36 PhysLean.Thermodynamics.Temperature.Basic

PhysLean/Thermodynamics/Temperature/Basic.lean on GitHub.

instance Coe Temperature ℝ≥0 [source]

Coercion to `ℝ≥0`.

```
instance : Coe Temperature ℝ≥0
```

lemma beta_pos [source]

Positivity of `β` from positivity of temperature.

```
lemma beta_pos (T : Temperature) (hT_pos : 0 < T.val) : 0 < (T.β : ℝ)
```

lemma eventually_pos_ofβ [source]

Eventually, `ofβ β` is positive as $\beta \to \infty$`.

```
lemma eventually_pos_ofβ : ∀ᶠ b : ℝ≥0 in atTop, ((Temperature.ofβ b : Temperature) : ℝ) > 0
```

lemma tendsto_const_inv_mul_atTop [source]

General helper: for any `a > 0`, we have `1 / (a * b) → 0` as `b → ∞` in `ℝ≥0`.

```
private lemma tendsto_const_inv_mul_atTop (a : ℝ) (ha : 0 < a) :
    Tendsto (fun b : ℝ≥0 => (1 : ℝ) / (a * (b : ℝ))) atTop (𝓝 (0 : ℝ))
```

lemma tendsto_toReal_ofβ_atTop [source]

Core convergence: as $\beta \to \infty$, `toReal (ofβ β) → 0` in $\mathbb{R}$.

```
lemma tendsto_toReal_ofβ_atTop :
    Tendsto (fun b : ℝ≥0 => (Temperature.ofβ b : ℝ))
      atTop (𝓝 (0 : ℝ))
```

lemma tendsto_ofβ_atTop [source]

As $\beta \to \infty$, $T = of\beta\ \beta \to 0+$ in $\mathbb{R}$ (within Ioi 0).

```
lemma tendsto_ofβ_atTop :
    Tendsto (fun b : ℝ≥0 => (Temperature.ofβ b : ℝ))
      atTop (nhdsWithin 0 (Set.Ioi 0))
```

2.37 PhysLean.Thermodynamics.Temperature.TemperatureUnits

PhysLean/Thermodynamics/Temperature/TemperatureUnits.lean on GitHub.

instance HDiv TemperatureUnit TemperatureUnit ℝ≥0 [source]

```
noncomputable instance : HDiv TemperatureUnit TemperatureUnit ℝ≥0 where
```

lemma div_neq_zero [source]

```
@[simp]
lemma div_neq_zero (x y : TemperatureUnit) : ¬ x / y = (0 : ℝ≥0)
```

lemma div_mul_div_coe [source]

```
@[simp]
lemma div_mul_div_coe (x y z : TemperatureUnit) :
    (x / y : ℝ) * (y /z : ℝ) = x /z
```

lemma self_div_scale [source]

```
@[simp]
lemma self_div_scale (x : TemperatureUnit) (r : ℝ) (hr : 0 < r) :
    x / scale r x hr = ((1/r, _root_.div_nonneg (by simp) (le_of_lt hr)) : ℝ≥0)
```

2.38 PhysLean.Units.Basic

PhysLean/Units/Basic.lean on GitHub.

lemma Dimension.ext [source]

```
@[ext]
lemma ext {d1 d2 : Dimension}
    (h1 : d1.length = d2.length)
    (h2 : d1.time = d2.time)
    (h3 : d1.mass = d2.mass)
    (h4 : d1.charge = d2.charge)
    (h5 : d1.temperature = d2.temperature) :
    d1 = d2
```

lemma length_mul [source]

```
@[simp]
lemma length_mul (d1 d2 : Dimension) :
    (d1 * d2).length = d1.length + d2.length
```

lemma mass_mul [source]

```
@[simp]
lemma mass_mul (d1 d2 : Dimension) :
    (d1 * d2).mass = d1.mass + d2.mass
```

lemma charge_mul [source]

```
@[simp]
lemma charge_mul (d1 d2 : Dimension) :
    (d1 * d2).charge = d1.charge + d2.charge
```

lemma temperature_mul [source]

```
@[simp]
lemma temperature_mul (d1 d2 : Dimension) :
    (d1 * d2).temperature = d1.temperature + d2.temperature
```

instance CommGroup Dimension [source]

```
instance : CommGroup Dimension where
```

lemma inv_length [source]

```
@[simp]
lemma inv_length (d : Dimension) : d⁻¹.length = -d.length
```

lemma inv_time [source]

```
@[simp]
lemma inv_time (d : Dimension) : d⁻¹.time = -d.time
```

lemma inv_mass [source]

```
@[simp]
lemma inv_mass (d : Dimension) : d⁻¹.mass = -d.mass
```

lemma inv_charge [source]

```
@[simp]
lemma inv_charge (d : Dimension) : d⁻¹.charge = -d.charge
```

lemma inv_temperature [source]

```
@[simp]
lemma inv_temperature (d : Dimension) : d⁻¹.temperature = -d.temperature
```

lemma L⌈_length [source]

```
@[simp]
lemma L⌈_length : L⌈.length = 1
```

lemma L⌈_time [source]

```
@[simp]
lemma L⌈_time : L⌈.time = 0
```

lemma L⌈_mass [source]

```
@[simp]
lemma L⌈_mass : L⌈.mass = 0
```

lemma L⌈_charge [source]

```
@[simp]
lemma L⌈_charge : L⌈.charge = 0
```

lemma L⌈_temperature [source]

```
@[simp]
lemma L⌈_temperature : L⌈.temperature = 0
```

lemma T⌈_length [source]

```
@[simp]
lemma T⌈_length : T⌈.length = 0
```

lemma T⌈_time [source]

```
@[simp]
lemma T⌈_time : T⌈.time = 1
```

lemma T⌈_mass [source]

```
@[simp]
lemma T⌈_mass : T⌈.mass = 0
```

lemma T⌈_charge [source]

```
@[simp]
lemma T⌈_charge : T⌈.charge = 0
```

lemma T⌈_temperature [source]

```
@[simp]
lemma T⌈_temperature : T⌈.temperature = 0
```

structure UnitChoices [source]

The choice of units.

```
@[ext]
structure UnitChoices where
  length : LengthUnit
```

```
  time : TimeUnit
  mass : MassUnit
  charge : ChargeUnit
  temperature : TemperatureUnit
```

def dimScale [source]

Given two choices of units `u1` and `u2` and a dimension `d`, the element of `ℝ≥0` corresponding to the scaling (by definition) of a quantity of dimension `d` when changing from units `u1` to `u2`.

```
noncomputable def dimScale (u1 u2 : UnitChoices) (d : Dimension) : ℝ≥0
```

lemma dimScale_self [source]

```
@[simp]
lemma dimScale_self (u : UnitChoices) (d : Dimension) :
    dimScale u u d = 1
```

lemma dimScale_zero [source]

```
@[simp]
lemma dimScale_zero (u1 u2 : UnitChoices) :
    dimScale u1 u2 0 = 1
```

lemma dimScale_transitive [source]

```
lemma dimScale_transitive (u1 u2 u3 : UnitChoices) (d : Dimension) :
    dimScale u1 u2 d * dimScale u2 u3 d = dimScale u1 u3 d
```

lemma dimScale_mul [source]

```
@[simp]
lemma dimScale_mul (u1 u2 : UnitChoices) (d1 d2 : Dimension) :
    dimScale u1 u2 (d1 * d2) = dimScale u1 u2 d1 * dimScale u1 u2 d2
```

lemma dimScale_mul_symm [source]

```
@[simp]
lemma dimScale_mul_symm (u1 u2 : UnitChoices) (d : Dimension) :
    dimScale u1 u2 d * dimScale u2 u1 d = 1
```

lemma dimScale_coe_mul_symm [source]

```
@[simp]
lemma dimScale_coe_mul_symm (u1 u2 : UnitChoices) (d : Dimension) :
    (toReal (dimScale u1 u2 d)) * (toReal (dimScale u2 u1 d)) = 1
```

lemma dimScale_neq_zero [source]

```
@[simp]
lemma dimScale_neq_zero (u1 u2 : UnitChoices) (d : Dimension) :
```

```
    dimScale u1 u2 d ≠ 0
```

lemma dimScale_inv [source]

```
lemma dimScale_inv (u1 u2 : UnitChoices) (d : Dimension) :
    dimScale u1 u2 d⁻¹ = (dimScale u1 u2 d)⁻¹
```

lemma dimScale_symm [source]

```
lemma dimScale_symm (u1 u2 : UnitChoices) (d : Dimension) :
    dimScale u1 u2 d = (dimScale u2 u1 d)⁻¹
```

lemma dimScale_of_inv_eq_swap [source]

```
lemma dimScale_of_inv_eq_swap (u1 u2 : UnitChoices) (d : Dimension) :
    dimScale u1 u2 d⁻¹ = dimScale u2 u1 d
```

def SI [source]

The choice of units corresponding to SI units, that is - meters, - seconds, - kilograms, - columbs, - kelvin.

```
noncomputable def SI : UnitChoices where
```

lemma SI_length [source]

```
@[simp]
lemma SI_length : SI.length = LengthUnit.meters
```

lemma SI_time [source]

```
@[simp]
lemma SI_time : SI.time = TimeUnit.seconds
```

lemma SI_mass [source]

```
@[simp]
lemma SI_mass : SI.mass = MassUnit.kilograms
```

lemma SI_charge [source]

```
@[simp]
lemma SI_charge : SI.charge = ChargeUnit.coulombs
```

lemma SI_temperature [source]

```
@[simp]
lemma SI_temperature : SI.temperature = TemperatureUnit.kelvin
```

class CarriesDimension [source]

A type `M` carries a dimension `d` if every element of `M` is supposed to have this dimension. For example, the type `Time` will carry a dimension `T⌈`.

```
class CarriesDimension (M : Type) extends MulAction ℝ≥0 M where
  d : Dimension
```

class ModuleCarriesDimension [source]

A module `M` carries a dimension `d` if every element of `M` is supposed to have this dimension. This is defined in additon to `CarriesDimension` to prevent a type-casting dimond.

```
class ModuleCarriesDimension (M : Type) [AddCommMonoid M] [Module ℝ M] where
  d : Dimension
```

instance CarriesDimension M [source]

```
instance {M : Type} [AddCommMonoid M] [Module ℝ M] [ModuleCarriesDimension M] :
    CarriesDimension M where
```

def HasDimension [source]

A quantity of type `M` which depends on a choice of units `UnitChoices` is said to be of dimension `d` if it scales by `UnitChoices.dimScale u1 u2 d` under a change in units.

```
def HasDimension {M : Type} [CarriesDimension M] (f : UnitChoices → M) : Prop
```

lemma hasDimension_iff [source]

```
lemma hasDimension_iff {M : Type} [CarriesDimension M] (f : UnitChoices → M) :
    HasDimension f ↔ ∀ u1 u2 : UnitChoices, f u2 =
    UnitChoices.dimScale u1 u2 (CarriesDimension.d M) • f u1
```

def Dimensionful [source]

The subtype of functions `UnitChoices → M`, for which `M` carries a dimension, which `HasDimension`.

```
def Dimensionful (M : Type) [CarriesDimension M]
```

lemma Dimensionful.ext [source]

```
@[ext]
lemma Dimensionful.ext {M : Type} [CarriesDimension M] (f1 f2 : Dimensionful M)
    (h : f1.val = f2.val) : f1 = f2
```

instance MulAction ℝ≥0 (Dimensionful M) [source]

```
instance {M : Type} [CarriesDimension M] : MulAction ℝ≥0 (Dimensionful M) where
```

lemma smul_apply [source]

```
@[simp]
lemma Dimensionful.smul_apply {M : Type} [CarriesDimension M]
    (a : ℝ≥0) (f : Dimensionful M) (u : UnitChoices) :
    (a • f).1 u = a • f.1 u
```

def toDimensionful [source]

For `M` carying a dimension `d`, the equivalence between `M` and `Dimension M`, given a choice of units.

```
noncomputable def CarriesDimension.toDimensionful {M : Type} [CarriesDimension M]
    (u : UnitChoices) :
    M ≃ Dimensionful M where
```

lemma toDimensionful_apply_apply [source]

```
lemma CarriesDimension.toDimensionful_apply_apply
    {M : Type} [CarriesDimension M] (u1 u2 : UnitChoices) (m : M) :
    (toDimensionful u1 m).1 u2 = (u1.dimScale u2 (CarriesDimension.d M)) • m
```

class UnitDependent [source]

A type carries the instance `UnitDependent M` if it depends on a choice of units. This dependence is manifested in `scaleUnit u1 u2` which describes how elements of `M` change under a scaling of units which would take `u1` to `u2`.

This type is used for functions, and propositions etc.

```
class UnitDependent (M : Type) where
  scaleUnit : UnitChoices → UnitChoices → M → M
  scaleUnit_trans : ∀ u1 u2 u3 m, scaleUnit u2 u3 (scaleUnit u1 u2 m) = scaleUnit u1 u3 m
  scaleUnit_trans' : ∀ u1 u2 u3 m, scaleUnit u1 u2 (scaleUnit u2 u3 m) = scaleUnit u1 u3 m
  scaleUnit_id : ∀ u m, scaleUnit u u m = m
```

class MulUnitDependent [source]

A type `M` with an instance of `UnitDependent M` such that `scaleUnit u1 u2` is compatible with the `MulAction ℝ≥0 M` instance on `M`.

```
class MulUnitDependent (M : Type) [MulAction ℝ≥0 M] extends
    UnitDependent M where
  scaleUnit_mul : ∀ u1 u2 (r : ℝ≥0) m,
    scaleUnit u1 u2 (r • m) = r • scaleUnit u1 u2 m
```

class LinearUnitDependent [source]

A type `M` with an instance of `UnitDependent M` such that `scaleUnit u1 u2` is compatible with the `Module ℝ M` instance on `M`.

```
class LinearUnitDependent (M : Type) [AddCommMonoid M] [Module ℝ M] extends UnitDependent M
    where
  scaleUnit_add : ∀ u1 u2 m1 m2,
    scaleUnit u1 u2 (m1 + m2) = scaleUnit u1 u2 m1 + scaleUnit u1 u2 m2
```

```
  scaleUnit_smul : ∀ u1 u2 (r : ℝ) m,
    scaleUnit u1 u2 (r • m) = r • scaleUnit u1 u2 m
```

class ContinuousLinearUnitDependent [source]

A type `M` with an instance of `UnitDependent M` such that `scaleUnit u1 u2` is compatible with the `Module ℝ M` instance on `M`, and is continuous.

```
class ContinuousLinearUnitDependent (M : Type) [AddCommMonoid M] [Module ℝ M]
    [TopologicalSpace M] extends LinearUnitDependent M where
  scaleUnit_cont : ∀ u1 u2, Continuous (scaleUnit u1 u2)
```

lemma scaleUnit_symm_apply [source]

```
@[simp]
lemma UnitDependent.scaleUnit_symm_apply {M : Type} [UnitDependent M]
    (u1 u2 : UnitChoices) (m : M) :
    scaleUnit u2 u1 (scaleUnit u1 u2 m) = m
```

lemma scaleUnit_injective [source]

```
@[simp]
lemma UnitDependent.scaleUnit_injective {M : Type} [UnitDependent M]
    (u1 u2 : UnitChoices) (m1 m2 : M) :
    scaleUnit u1 u2 m1 = scaleUnit u1 u2 m2 ↔ m1 = m2
```

def scaleUnitEquiv [source]

For an `M` with an instance of `UnitDependent M`, `scaleUnit u1 u2` as an equivalence.

```
def UnitDependent.scaleUnitEquiv {M : Type} [UnitDependent M]
    (u1 u2 : UnitChoices) : M ≃ M where
```

def scaleUnitLinear [source]

For an `M` with an instance of `LinearUnitDependent M`, `scaleUnit u1 u2` as a linear map.

```
def LinearUnitDependent.scaleUnitLinear
    {M : Type} [AddCommMonoid M] [Module ℝ M] [LinearUnitDependent M]
    (u1 u2 : UnitChoices) :
    M →ₗ[ℝ] M where
```

def scaleUnitLinearEquiv [source]

For an `M` with an instance of `LinearUnitDependent M`, `scaleUnit u1 u2` as a linear equivalence.

```
def LinearUnitDependent.scaleUnitLinearEquiv {M : Type} [AddCommMonoid M]
    [Module ℝ M] [LinearUnitDependent M] (u1 u2 : UnitChoices) :
    M ≃ₗ[ℝ] M
```

def scaleUnitContLinear [source]

For an `M` with an instance of `ContinuousLinearUnitDependent M`, `scaleUnit u1 u2` as a continuous linear map.

```
def ContinuousLinearUnitDependent.scaleUnitContLinear {M : Type} [AddCommMonoid M] [Module ℝ M]
    [TopologicalSpace M] [ContinuousLinearUnitDependent M]
    (u1 u2 : UnitChoices) : M →L[ℝ] M where
```

def scaleUnitContLinearEquiv [source]

For an `M` with an instance of `ContinuousLinearUnitDependent M`, `scaleUnit u1 u2` as a continuous linear equivalence.

```
def ContinuousLinearUnitDependent.scaleUnitContLinearEquiv {M : Type} [AddCommMonoid M] [Module
     ℝ M]
    [TopologicalSpace M] [ContinuousLinearUnitDependent M]
    (u1 u2 : UnitChoices) : M ≃L[ℝ] M
```

lemma scaleUnitContLinearEquiv_apply [source]

```
@[simp]
lemma ContinuousLinearUnitDependent.scaleUnitContLinearEquiv_apply
    {M : Type} [AddCommGroup M] [Module ℝ M] [TopologicalSpace M]
    [ContinuousLinearUnitDependent M]
    (u1 u2 : UnitChoices) (m : M) :
    (ContinuousLinearUnitDependent.scaleUnitContLinearEquiv u1 u2) m =
      scaleUnit u1 u2 m
```

lemma scaleUnitContLinearEquiv_symm_apply [source]

```
@[simp]
lemma ContinuousLinearUnitDependent.scaleUnitContLinearEquiv_symm_apply
    {M : Type} [AddCommGroup M] [Module ℝ M] [TopologicalSpace M]
    [ContinuousLinearUnitDependent M]
    (u1 u2 : UnitChoices) (m : M) :
    (ContinuousLinearUnitDependent.scaleUnitContLinearEquiv u1 u2).symm m =
      scaleUnit u2 u1 m
```

instance UnitDependent UnitChoices [source]

```
noncomputable instance : UnitDependent UnitChoices where
```

lemma scaleUnit_apply_fst [source]

```
@[simp]
lemma UnitChoices.scaleUnit_apply_fst (u1 u2 : UnitChoices) :
    (scaleUnit u1 u2 u1) = u2
```

lemma dimScale_scaleUnit [source]

```
@[simp]
lemma UnitChoices.dimScale_scaleUnit {u1 u2 u : UnitChoices} (d : Dimension) :
    u.dimScale (scaleUnit u1 u2 u) d = u1.dimScale u2 d
```

lemma of_scaleUnit [source]

```
lemma Dimensionful.of_scaleUnit {M : Type} [CarriesDimension M] {u1 u2 u : UnitChoices}
    (c : Dimensionful M) :
    c.1 (scaleUnit u1 u2 u) =
    u1.dimScale u2 (CarriesDimension.d M) • c.1 (u)
```

instance MulUnitDependent M1 [source]

```
noncomputable instance {M1 : Type} [CarriesDimension M1] : MulUnitDependent M1 where
```

lemma scaleUnit_apply [source]

```
lemma CarriesDimension.scaleUnit_apply {M : Type} [CarriesDimension M]
    (u1 u2 : UnitChoices) (m : M) :
    scaleUnit u1 u2 m = (u1.dimScale u2 (d M)) • m
```

lemma scaleUnit_apply' [source]

```
lemma CarriesDimension.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)
```

instance LinearUnitDependent M [source]

```
noncomputable instance {M : Type} [AddCommMonoid M] [Module ℝ M]
    [ModuleCarriesDimension M] : LinearUnitDependent M where
```

instance ContinuousLinearUnitDependent M [source]

```
noncomputable instance {M : Type} [AddCommMonoid M] [Module ℝ M]
    [ModuleCarriesDimension M] [TopologicalSpace M]
    [ContinuousConstSMul ℝ M] : ContinuousLinearUnitDependent M where
```

instance UnitDependent (M1 → M2) [source]

```
noncomputable instance {M1 M2 : Type} [UnitDependent M2] :
    UnitDependent (M1 → M2) where
```

instance LinearUnitDependent (M1 →ₗ[ℝ] M2) [source]

```
noncomputable instance {M1 M2 : Type} [AddCommMonoid M1] [Module ℝ M1]
    [AddCommMonoid M2] [Module ℝ M2] [LinearUnitDependent M2] :
    LinearUnitDependent (M1 →ₗ[ℝ] M2) where
```

instance ContinuousLinearUnitDependent (M1 →L[ℝ] M2) [source]

```
noncomputable instance {M1 M2 : Type} [AddCommGroup M1] [Module ℝ M1]
    [TopologicalSpace M1]
    [AddCommGroup M2] [Module ℝ M2] [TopologicalSpace M2] [ContinuousConstSMul ℝ M2]
    [IsTopologicalAddGroup M2]
```

```
    [ContinuousLinearUnitDependent M2] :
    ContinuousLinearUnitDependent (M1 →L[ℝ] M2) where
```

instance UnitDependent (M1 → M2) [source]

```
noncomputable instance {M1 M2 : Type} [UnitDependent M1] :
    UnitDependent (M1 → M2) where
```

instance instUnitDependentTwoSided [source]

```
noncomputable instance instUnitDependentTwoSided
    {M1 M2 : Type} [UnitDependent M1] [UnitDependent M2] :
    UnitDependent (M1 → M2) where
```

instance instUnitDependentTwoSidedMul [source]

```
noncomputable instance instUnitDependentTwoSidedMul
    {M1 M2 : Type} [UnitDependent M1] [MulAction ℝ≥0 M2] [MulUnitDependent M2] :
    MulUnitDependent (M1 → M2) where
```

instance instContinuousLinearUnitDependentMap [source]

```
noncomputable instance instContinuousLinearUnitDependentMap
    {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] :
    ContinuousLinearUnitDependent (M1 →L[ℝ] M2) where
```

def IsDimensionallyInvariant [source]

A term of type `M` carrying an instance of `UnitDependent M` is said to be dimensionally invariant if under a change of units it remains the same.

This corresponds to the statement that term is dimensionally correct.

```
def IsDimensionallyInvariant {M : Type} [UnitDependent M] (m : M) : Prop
```

lemma isDimensionallyInvariant_iff [source]

```
lemma isDimensionallyInvariant_iff {M : Type} [UnitDependent M] (m : M) :
    IsDimensionallyInvariant m ↔ ∀ u1 u2 : UnitChoices,
      scaleUnit u1 u2 m = m
```

class DMul [source]

The multiplication of an element of `M1` with an element of `M2` to get an element of `M3` in such a way that dimensions are preserved.

```
class DMul (M1 M2 M3 : Type) [CarriesDimension M1] [CarriesDimension M2] [CarriesDimension M3]
    extends HMul M1 M2 M3 where
  mul_dim : ∀ (m1 : Dimensionful M1) (m2 : Dimensionful M2),
```

```
    HasDimension (fun u => hMul (m1.1 u) (m2.1 u))
```

lemma hMul_scaleUnit [source]

```
@[simp]
lemma DMul.hMul_scaleUnit {M1 M2 M3 : Type} [CarriesDimension M1] [CarriesDimension M2]
    [CarriesDimension M3]
    [DMul M1 M2 M3] (m1 : M1) (m2 : M2) (u1 u2 : UnitChoices) :
    (scaleUnit u1 u2 m1) * (scaleUnit u1 u2 m2) =
    scaleUnit u1 u2 (m1 * m2)
```

def DimSet [source]

Given a type `M` that depends on units, e.g. the function type `M1 → M2` between two types carrying a dimension, the subtype of `M` which scales according to the dimension `d`.

```
def DimSet (M : Type) [MulAction ℝ≥0 M] [MulUnitDependent M] (d : Dimension) :
    Set M
```

instance MulAction ℝ≥0 (DimSet M d) [source]

```
instance (M : Type) [MulAction ℝ≥0 M] [MulUnitDependent M] (d : Dimension) :
    MulAction ℝ≥0 (DimSet M d) where
```

instance CarriesDimension (DimSet M d) [source]

```
instance (M : Type) [MulAction ℝ≥0 M] [MulUnitDependent M] (d : Dimension) :
    CarriesDimension (DimSet M d) where
```

lemma scaleUnit_dimSet_val [source]

```
@[simp]
lemma scaleUnit_dimSet_val {M : Type} [MulAction ℝ≥0 M] [MulUnitDependent M] (d : Dimension)
    (m : DimSet M d) (u1 u2 : UnitChoices) :
    (scaleUnit u1 u2 m).1 = scaleUnit u1 u2 m.1
```

2.39 PhysLean.Units.Examples

PhysLean/Units/Examples.lean on GitHub.

def meters400 [source]

The length corresponding to 400 meters.

```
noncomputable def meters400 : Dimensionful (WithDim L⌈ ℝ)
```

def speedOfLight [source]

The speed of light as a dimensionful quantity.

```
noncomputable def speedOfLight : Dimensionful (WithDim (L⌈ * T⌈⁻¹) ℝ)
```

def EnergyMass [source]

The equation `E = m c^2`, in this equation we `E` and `m` are implicitly in the units `u`, while the speed of light is explicitly written in those units.

```
def EnergyMass (m : WithDim M⌈ ℝ) (E : WithDim (M⌈ * L⌈ * L⌈ * T⌈⁻¹ * T⌈⁻¹) ℝ)
    (u : UnitChoices) : Prop
```

def EnergyMass' [source]

The equation `E = m c^2`, in this version everything is written explicitly in terms of a choice of units.

```
def EnergyMass' (m : Dimensionful (WithDim M⌈ ℝ))
    (E : Dimensionful (WithDim (M⌈ * L⌈ * L⌈ * T⌈⁻¹ * T⌈⁻¹) ℝ))
    (u : UnitChoices) : Prop
```

lemma energyMass_isDimensionallyInvariant [source]

The lemma that the proposition `EnergyMass` is dimensionally correct

```
lemma energyMass_isDimensionallyInvariant :
    IsDimensionallyInvariant EnergyMass
```

lemma example1_energyMass [source]

```
lemma example1_energyMass : EnergyMass ⟨2⟩ ⟨2 * 299792458 ^ 2⟩ SI
```

lemma example2_energyMass [source]

```
lemma example2_energyMass (u2 : UnitChoices) :
    EnergyMass (scaleUnit SI u2 ⟨2⟩)
      (scaleUnit SI u2 ⟨2 * 299792458 ^ 2⟩) u2
```

2.40 PhysLean.Units.FDeriv

PhysLean/Units/FDeriv.lean on GitHub.

lemma fderiv_isDimensionallyInvariant [source]

If a function is dimensionally valid then so is it's derivative.

```
lemma fderiv_isDimensionallyInvariant {M1 M2 : Type} [NormedAddCommGroup M1] [NormedSpace ℝ M1]
    [ContinuousConstSMul ℝ M1] [ModuleCarriesDimension M1]
    [NormedAddCommGroup M2] [NormedSpace ℝ M2]
    [SMulCommClass ℝ ℝ M2] [ContinuousConstSMul ℝ M2]
    [IsTopologicalAddGroup M2]
    [ModuleCarriesDimension M2]
    (f : M1 → M2) (hf : IsDimensionallyInvariant f) (f_diff : Differentiable ℝ f) :
    IsDimensionallyInvariant (fderiv ℝ f)
```

2.41 PhysLean.Units.Integral

PhysLean/Units/Integral.lean on GitHub.

instance MulUnitDependent (MeasureTheory.Measure M) [source]

```
noncomputable instance (M : Type)
    [NormedAddCommGroup M] [NormedSpace ℝ M] [ModuleCarriesDimension M]
    [MeasurableSpace M] [self : MeasurableConstSMul ℝ M] :
      MulUnitDependent (MeasureTheory.Measure M) where
```

lemma scaleUnit_measure [source]

```
lemma scaleUnit_measure (u1 u2 : UnitChoices) (μ : MeasureTheory.Measure M) :
    scaleUnit u1 u2 μ = μ.map (fun m => scaleUnit u1 u2 m)
```

lemma integral_isDimensionallyInvariant [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 invariant.

```
lemma integral_isDimensionallyInvariant (d : Dimension) :
    IsDimensionallyInvariant (fun (μ : DimSet (MeasureTheory.Measure M) d)
      (f : DimSet (M → G) (CarriesDimension.d G * d⁻¹)) ↦ ∫ x, f.1 x ∂μ.1)
```

2.42 PhysLean.Units.WithDim.Area

PhysLean/Units/WithDim/Area.lean on GitHub.

abbrev DimArea [source]

The type of areas in the absence of a choice of unit.

```
abbrev DimArea : Type
```

def squareMeter [source]

The dimensional area corresponding to 1 square meter.

```
noncomputable def squareMeter : DimArea
```

def squareFoot [source]

The dimensional area corresponding to 1 square foot.

```
noncomputable def squareFoot : DimArea
```

def squareMile [source]

The dimensional area corresponding to 1 square mile.

```
noncomputable def squareMile : DimArea
```

def are [source]

The dimensional area corresponding to 1 are (100 square meters).

```
noncomputable def are : DimArea
```

def hectare [source]

The dimensional area corresponding to 1 hectare (10,000 square meters).

```
noncomputable def hectare : DimArea
```

def acre [source]

The dimensional area corresponding to 1 acre (1/640 square miles).

```
noncomputable def acre : DimArea
```

lemma squareMeter_in_SI [source]

```
@[simp]
lemma squareMeter_in_SI : squareMeter.1 SI = ⟨1⟩
```

lemma squareFoot_in_SI [source]

```
@[simp]
lemma squareFoot_in_SI : squareFoot.1 SI = ⟨0.09290304⟩
```

lemma squareMile_in_SI [source]

```
@[simp]
lemma squareMile_in_SI : squareMile.1 SI = ⟨2589988.110336⟩
```

lemma are_in_SI [source]

```
@[simp]
lemma are_in_SI : are.1 SI = ⟨100⟩
```

lemma hectare_in_SI [source]

```
@[simp]
lemma hectare_in_SI : hectare.1 SI = ⟨10000⟩
```

lemma acre_in_SI [source]

```
@[simp]
lemma acre_in_SI : acre.1 SI = ⟨4046.8564224⟩
```

lemma acre_eq_mul_squareFeet [source]

One acre is exactly `43560` square feet.

```
lemma acre_eq_mul_squareFeet : acre = (43560 : ℝ≥0) • squareFoot
```

2.43 PhysLean.Units.WithDim.Basic

PhysLean/Units/WithDim/Basic.lean on GitHub.

structure WithDim [source]

The type `M` carrying an instance of a dimension `d`.

```
structure WithDim (d : Dimension) (M : Type) [MulAction ℝ≥0 M] where
  val : M
```

lemma ext [source]

```
@[ext]
lemma ext {d M} [MulAction ℝ≥0 M] (x1 x2 : WithDim d M) (h : x1.val = x2.val) : x1 = x2
```

instance MulAction ℝ≥0 (WithDim d M) [source]

```
instance (d : Dimension) (M : Type) [MulAction ℝ≥0 M] : MulAction ℝ≥0 (WithDim d M) where
```

lemma smul_val [source]

```
@[simp]
lemma smul_val {d : Dimension} {M : Type} [MulAction ℝ≥0 M] (a : ℝ≥0) (m : WithDim d M) :
    (a • m).val = a • m.val
```

instance CarriesDimension (WithDim d M) [source]

```
instance (d : Dimension) (M : Type) [inst : MulAction ℝ≥0 M] :
    CarriesDimension (WithDim d M) where
```

lemma carriesDimension_d [source]

```
@[simp]
lemma carriesDimension_d (d : Dimension) (M : Type) [MulAction ℝ≥0 M] :
    CarriesDimension.d (WithDim d M) = d
```

instance HMul (WithDim d1 ℝ) (WithDim d2 ℝ) (WithDim (d1 * d2) ℝ) [source]

```
instance {d1 d2 : Dimension} :
    HMul (WithDim d1 ℝ) (WithDim d2 ℝ) (WithDim (d1 * d2) ℝ) where
```

lemma withDim_hMul_val [source]

```
@[simp]
lemma withDim_hMul_val {d1 d2 : Dimension} (m1 : WithDim d1 ℝ) (m2 : WithDim d2 ℝ) :
    (m1 * m2).val = m1.val * m2.val
```

instance DMul (WithDim d1 ℝ) (WithDim d2 ℝ) (WithDim (d1 * d2) ℝ) [source]

```
instance {d1 d2 : Dimension} :
    DMul (WithDim d1 ℝ) (WithDim d2 ℝ) (WithDim (d1 * d2) ℝ) where
```

2.44 PhysLean.Units.WithDim.Energy

PhysLean/Units/WithDim/Energy.lean on GitHub.

abbrev DimEnergy [source]

Energy as a dimensional quantity with dimension MLT^{-2}.

```
abbrev DimEnergy : Type
```

def joule [source]

The dimensional energy corresponding to 1 joule, J.

```
noncomputable def joule : DimEnergy
```

def electronVolt [source]

The dimensional energy corresponding to 1 electron volt, 1.602176634×10−19 J.

```
noncomputable def electronVolt : DimEnergy
```

def calorie [source]

The dimensional energy corresponding to 1 calorie, 4.184 J.

```
noncomputable def calorie : DimEnergy
```

def kilowattHour [source]

The dimensional energy corresponding to 1 kilowatt-hours, (3,600,000 J).

```
noncomputable def kilowattHour : DimEnergy
```

2.45 PhysLean.Units.WithDim.Pressure

PhysLean/Units/WithDim/Pressure.lean on GitHub.

abbrev DimPressure [source]

Pressure as a dimensional quantity with dimension $ML^{-1}T^{-2}$.

```
abbrev DimPressure : Type
```

def pascal [source]

The dimensional pressure corresponding to 1 pascal, Pa.

```
noncomputable def pascal : DimPressure
```

def millimeterOfMercury [source]

The dimensional pressure corresponding to 1 millimeter of mercury (133.322387415 pascals).

```
noncomputable def millimeterOfMercury : DimPressure
```

def bar [source]

The dimensional pressure corresponding to 1 bar (100,000 pascals).

```
noncomputable def bar : DimPressure
```

def standardAtmosphere [source]

The dimensional pressure corresponding to 1 standard atmosphere (101,325 pascals).

```
noncomputable def standardAtmosphere : DimPressure
```

def torr [source]

The dimensional pressure corresponding to 1 torr (1/760 of standard atmosphere pressure).

```
noncomputable def torr : DimPressure
```

def psi [source]

The dimensional pressure corresponding to 1 pound per square inch.

```
noncomputable def psi : DimPressure
```

2.46 PhysLean.Units.WithDim.Speed

PhysLean/Units/WithDim/Speed.lean on GitHub.

abbrev DimSpeed [source]

The type of speeds in the absence of a choice of unit.

```
abbrev DimSpeed : Type
```

def oneMeterPerSecond [source]

The dimensional speed corresponding to 1 meter per second.

```
noncomputable def oneMeterPerSecond : DimSpeed
```

def oneMilePerHour [source]

The dimensional speed corresponding to 1 mile per hour.

```
noncomputable def oneMilePerHour : DimSpeed
```

def oneKilometerPerHour [source]

The dimensional speed corresponding to 1 kilometer per hour.

```
noncomputable def oneKilometerPerHour : DimSpeed
```

def oneKnot [source]

The dimensional speed corresponding to 1 knot, aka, one nautical mile per hour.

```
noncomputable def oneKnot : DimSpeed
```

lemma oneMeterPerSecond_in_SI [source]

```
@[simp]
lemma oneMeterPerSecond_in_SI : oneMeterPerSecond.1 SI = ⟨1⟩
```

lemma oneMilePerHour_in_SI [source]

```
@[simp]
lemma oneMilePerHour_in_SI : oneMilePerHour.1 SI = ⟨0.44704⟩
```

lemma oneKilometerPerHour_in_SI [source]

```
@[simp]
lemma oneKilometerPerHour_in_SI :
    oneKilometerPerHour.1 SI = ⟨5/18⟩
```

lemma oneKnot_in_SI [source]

```
@[simp]
lemma oneKnot_in_SI : oneKnot.1 SI = ⟨463/900⟩
```

lemma oneKnot_eq_mul_oneKilometerPerHour [source]

```
lemma oneKnot_eq_mul_oneKilometerPerHour :
    oneKnot = (1.852 : ℝ≥0) • oneKilometerPerHour
```

lemma oneKilometerPerHour_eq_mul_oneKnot [source]

```
lemma oneKilometerPerHour_eq_mul_oneKnot:
    oneKilometerPerHour = (250/463 : ℝ≥0) • oneKnot
```

lemma oneMeterPerSecond_eq_mul_oneMilePerHour [source]

```
lemma oneMeterPerSecond_eq_mul_oneMilePerHour :
    oneMeterPerSecond = (3125/1397 : ℝ≥0) • oneMilePerHour
```