Finite-Dimensional Algebras and Scalar Towers

Arthur Freitas Ramos David Barros Hulak
Ruy J. G. B. de Queiroz

November 6, 2026

Abstract

For a finite-dimensional vector space over a finite-dimensional field extension, the dimension over the smaller field is the product of the two dimensions. The result includes vector subspaces and the zero space. For compatible towers of two fields and a finite-dimensional commutative unital algebra, trace and norm over the bottom field are the iterated trace and norm through the intermediate field. Examples include $\mathbb{F}_2 \subseteq \mathbb{F}_4$ and $\mathbb{R} \subseteq \mathbb{C}$. The development follows the standard finite extension treatment in [1].

Contents

1 Restriction of scalars for carrier-based vector spaces **2**

2 Existing scalar infrastructure **5**

3 Scalar towers **8**

4 Restriction of scalars **8**

5 Composition of bases **11**

6 Dimension in a tower **14**

7 Regular representation **15**

8 Algebra trace **22**

9 Algebra norm **25**

10 Examples **33**
10.1 Finite-field towers 33
10.2 The four-element field over $\mathbb{F}_2$ 35
10.3 The quadratic extension of the reals by the imaginary unit 39

theory *Scalar-Restriction-Bridge*
imports *HOL−New-Algebra.Steinitz*
begin

1 Restriction of scalars for carrier-based vector spaces

The upper carrier is an arbitrary vector space, not a field. In particular, it may be the zero space. The lower vector space describes the intermediate field over the smaller field, using the same scalar operations. These parameters can also describe subfields of a noncommutative ambient ring and a vector subspace of that ring.

locale *scalar-restriction-tower* =
  lower: *Vector-Space K sadd smult szero sone sadd szero F smult* +
  upper: *Vector-Space F sadd smult szero sone vadd vzero E scale*
  for *K F* :: *'s set*
    and *sadd smult* :: *'s* ⇒ *'s* ⇒ *'s*
    and *szero sone* :: *'s*
    and *vadd* :: *'v* ⇒ *'v* ⇒ *'v*
    and *vzero* :: *'v*
    and *E* :: *'v set*
    and *scale* :: *'s* ⇒ *'v* ⇒ *'v* +
  assumes *base-subset*: *K* ⊆ *F*
begin

sublocale *direct*: *Vector-Space K sadd smult szero sone vadd vzero E scale*
⟨*proof*⟩

Restricting the field changes the allowed coefficients, but not the expression for a linear combination.

lemma *restricted-lincomb*:
  direct.lincomb c A = *upper.lincomb c A*
  ⟨*proof*⟩

definition *product-basis* :: *'s set* ⇒ *'v set* ⇒ *'v set*
  where *product-basis B C* = (λ(*b*,*c*). *scale b c*) ‘ (*B* × *C*)

lemma *product-basis-finite*:
  assumes *finite B finite C*
  shows *finite* (*product-basis B C*)
  ⟨*proof*⟩

lemma *product-basis-subset*:
  assumes *B* ⊆ *F C* ⊆ *E*
  shows *product-basis B C* ⊆ *E*
  ⟨*proof*⟩

lemma *product-basis-empty* [*simp*]:
  product-basis B {} = {}
  ⟨*proof*⟩

lemma *lower-basis-nonzero*:

assumes B: *lower.basis* B **and** b: $b \in B$
shows $b \neq$ *szero*
⟨*proof*⟩

lemma *upper-basis-scaled-unique*:
assumes C: *upper.basis* C **and** c: $c \in C$ **and** d: $d \in C$
and b: $b \in F$ **and** e: $e \in F$
and *eq*: *scale* b c = *scale* e d
shows b = (*if* $c = d$ *then* e *else szero*)
⟨*proof*⟩

lemma *product-basis-injective*:
assumes B: *lower.basis* B **and** C: *upper.basis* C
shows *inj-on* ($\lambda(b,c)$. *scale* b c) ($B \times C$)
⟨*proof*⟩

lemma *scalar-lower-combination-in-product-span*:
assumes *BF*: $B \subseteq F$ **and** *CE*: $C \subseteq E$ **and** c: $c \in C$
and *fin*: *finite* T **and** *TB*: $T \subseteq B$
and *coeff*: $\bigwedge b.\ b \in T \Longrightarrow f\ b \in K$
shows *scale* (*lower.lincomb* f T) $c \in$ *direct.span* (*product-basis* B C)
⟨*proof*⟩

lemma *fincomp-in-direct-span*:
assumes *PE*: $P \subseteq E$ **and** *fin*: *finite* A
and *terms*: $\bigwedge a.\ a \in A \Longrightarrow g\ a \in$ *direct.span* P
shows *upper.vadd.fincomp* g $A \in$ *direct.span* P
⟨*proof*⟩

lemma *product-basis-spanning*:
assumes B: *lower.basis* B **and** C: *upper.basis* C
shows *direct.spanning* (*product-basis* B C)
⟨*proof*⟩

lemma *scale-lower-lincomb*:
assumes *BF*: $B \subseteq F$ **and** *cE*: $c \in E$
and *fK*: $\bigwedge b.\ b \in B \Longrightarrow f\ b \in K$
shows *scale* (*lower.lincomb* f B) c =
upper.vadd.fincomp ($\lambda b.$ *scale* (*smult* (f b) b) c) B
⟨*proof*⟩

lemma *vector-fincomp-cartesian*:
assumes *finB*: *finite* B **and** *finC*: *finite* C
and *gE*: $\bigwedge b\ c.\ [\![b \in B;\ c \in C]\!] \Longrightarrow g\ (b,c) \in E$
shows *upper.vadd.fincomp* g ($B \times C$) =
upper.vadd.fincomp ($\lambda c.$ *upper.vadd.fincomp* ($\lambda b.$ g (b,c)) B) C
⟨*proof*⟩

lemma *product-lincomb-nested*:

assumes B: *lower.basis B* **and** C: *upper.basis C*
and *fK*: $\bigwedge v.\ v \in$ *product-basis B C* $\Longrightarrow$ *f v* $\in K$
shows *direct.lincomb f* (*product-basis B C*) =
upper.lincomb ($\lambda c.$ *lower.lincomb* ($\lambda b.$ *f* (*scale b c*)) *B*) *C*
⟨*proof*⟩

lemma *product-basis-independent*:
assumes B: *lower.basis B* **and** C: *upper.basis C*
shows *direct.lin-indep* (*product-basis B C*)
⟨*proof*⟩

theorem *product-basis*:
assumes B: *lower.basis B* **and** C: *upper.basis C*
shows *direct.basis* (*product-basis B C*)
⟨*proof*⟩

lemma *product-basis-in-direct-span*:
assumes B: *lower.basis B* **and** C: *upper.basis C*
and b: $b \in B$ **and** c: $c \in C$
shows *scale b c* $\in$ *direct.span* (*product-basis B C*)
⟨*proof*⟩

Cardinality uniqueness turns a product basis into the dimension formula.

lemma *dimension-from-product-basis*:
assumes B: *lower.basis B* **and** C: *upper.basis C*
and P: *direct.basis* (*product-basis B C*)
and *inj*: *inj-on* ($\lambda(b,c).$ *scale b c*) ($B \times C$)
shows *direct.dimension* = *lower.dimension* $*$ *upper.dimension*
⟨*proof*⟩

theorem *finite-dimension-tower*:
assumes *lower-finite*: $\exists B.$ *lower.basis B*
and *upper-finite*: $\exists C.$ *upper.basis C*
shows $\exists P.$ *direct.basis P*
and *direct.dimension* = *lower.dimension* $*$ *upper.dimension*
⟨*proof*⟩

The zero space has an empty basis over either field. This case must not be excluded by an index type or by requiring the upper carrier to be a field.

lemma *zero-space-tower*:
assumes $E = \{vzero\}$
shows *direct.basis* {} *upper.basis* {}
and *direct.dimension* = *lower.dimension* $*$ *upper.dimension*
⟨*proof*⟩

end

end
theory *Compatible-Scalar-Actions*

imports
HOL−Analysis.Determinants
begin

2 Existing scalar infrastructure

Isabelle's type-based algebra library represents a module by the locale *module*, parameterized by an explicit scalar action. Consequently, several scalar actions may coexist without introducing new type classes. Linear maps are already represented by *module-hom*, and over fields by the definitionally equivalent locale *vector-space*. We use its existing operations for subspaces, spans, independence, and coordinate representations. The Cartesian matrix library supplies trace and determinant.

locale *compatible-scalar-actions* =
RS: *Modules.module rS* +
SA: *Modules.module sA* +
RA: *Modules.module rA*
for *rS* :: *'r*::*comm-ring-1* ⇒ *'s*::*comm-ring-1* ⇒ *'s*
and *sA* :: *'s* ⇒ *'a*::*comm-ring-1* ⇒ *'a*
and *rA* :: *'r* ⇒ *'a* ⇒ *'a* +
assumes *scalar-compat* [*simp*]:
rA r (*sA s a*) = *sA* (*rS r s*) *a*
begin

abbreviation *scalar-embed* :: *'r* ⇒ *'s*
where *scalar-embed r* ≡ *rS r 1*

lemma *direct-eq-upper-embed* [*simp*]:
rA r a = *sA* (*scalar-embed r*) *a*
⟨*proof*⟩

lemma *module-hom-restrict-scalars*:
assumes *module-hom sA sA f*
shows *module-hom rA rA f*
⟨*proof*⟩

end

locale *compatible-vector-spaces* =
compatible-scalar-actions rS sA rA
for *rS* :: *'r*::*field* ⇒ *'s*::*field* ⇒ *'s*
and *sA* :: *'s* ⇒ *'a*::*comm-ring-1* ⇒ *'a*
and *rA* :: *'r* ⇒ *'a* ⇒ *'a*
begin

lemma *linear-restrict-scalars*:
assumes *Vector-Spaces.linear sA sA f*

shows *Vector-Spaces.linear rA rA f*
⟨*proof*⟩

end

locale *commutative-algebra* =
Modules.module scale
for *scale* :: *'r*::*comm-ring-1* ⇒ *'a*::*comm-ring-1* ⇒ *'a* +
assumes *scale-mult-right* [*algebra-simps*]:
scale r x ∗ *y* = *scale r* (*x* ∗ *y*)
begin

abbreviation *scalar-embed* :: *'r* ⇒ *'a*
where *scalar-embed r* ≡ *scale r 1*

lemma *scale-eq-embed-mult*:
scale r x = *scalar-embed r* ∗ *x*
⟨*proof*⟩

lemma *scale-mult-left* [*algebra-simps*]:
x ∗ *scale r y* = *scale r* (*x* ∗ *y*)
⟨*proof*⟩

lemma *scalar-embed-zero* [*simp*]:
scalar-embed 0 = *0*
⟨*proof*⟩

lemma *scalar-embed-one* [*simp*]:
scalar-embed 1 = *1*
⟨*proof*⟩

lemma *scalar-embed-add* [*simp*]:
scalar-embed (*r* + *s*) = *scalar-embed r* + *scalar-embed s*
⟨*proof*⟩

lemma *scalar-embed-minus* [*simp*]:
scalar-embed (− *r*) = − *scalar-embed r*
⟨*proof*⟩

lemma *scalar-embed-diff* [*simp*]:
scalar-embed (*r* − *s*) = *scalar-embed r* − *scalar-embed s*
⟨*proof*⟩

lemma *scalar-embed-mult* [*simp*]:
scalar-embed (*r* ∗ *s*) = *scalar-embed r* ∗ *scalar-embed s*
⟨*proof*⟩

lemma *scalar-embed-of-nat* [*simp*]:
scalar-embed (*of-nat n*) = *of-nat n*

⟨*proof*⟩

lemma *scalar-embed-of-int* [*simp*]:
scalar-embed (*of-int z*) = *of-int z*
⟨*proof*⟩

lemma *scalar-embed-power* [*simp*]:
scalar-embed (*r* ^ *n*) = *scalar-embed r* ^ *n*
⟨*proof*⟩

end

locale *algebra-tower* =
RS: *commutative-algebra rS* +
SA: *commutative-algebra sA* +
RA: *commutative-algebra rA*
for *rS* :: *'r*::*comm-ring-1* ⇒ *'s*::*comm-ring-1* ⇒ *'s*
and *sA* :: *'s* ⇒ *'a*::*comm-ring-1* ⇒ *'a*
and *rA* :: *'r* ⇒ *'a* ⇒ *'a* +
assumes *scalar-compat* [*simp*]:
rA r (*sA s a*) = *sA* (*rS r s*) *a*
begin

sublocale *compatible-scalar-actions rS sA rA*
⟨*proof*⟩

lemma *composed-scalar-embed* [*simp*]:
rA r 1 = *sA* (*rS r 1*) *1*
⟨*proof*⟩

lemma *scalar-compat-mult* [*simp*]:
rA r (*sA s a* ∗ *b*) = *sA* (*rS r s*) (*a* ∗ *b*)
⟨*proof*⟩

lemma *scalar-compat-linear*:
assumes *module-hom sA sA f*
shows *f* (*rA r a*) = *rA r* (*f a*)
⟨*proof*⟩

end

end
theory *Algebra-Towers*
imports *Compatible-Scalar-Actions*
begin

3 Scalar towers

The locale *algebra-tower* packages three existing module actions and the single additional compatibility law required by a tower. The remaining theories derive functorial, basis, dimension, trace, and norm interfaces from it.

end
theory *Restriction-of-Scalars*
  imports *Algebra-Towers*
begin

4 Restriction of scalars

Linear maps in *HOL.Modules* are plain functions equipped with a locale proposition. Restricting scalars therefore changes only the proposition used to view a function, never the function itself.

definition *restrict-scalars* :: $('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$
  where *restrict-scalars f x* = *f x*

lemma *restrict-scalars-apply* [*simp*]:
  restrict-scalars f x = *f x*
  ⟨*proof*⟩

definition *subspace-rel* :: $'a$::*ab-group-add set* ⇒ $('a \times 'a)$ *set*
  where *subspace-rel U* = {(*x*, *y*). *x* − *y* ∈ *U*}

definition *subspace-coset* :: $'a$::*ab-group-add set* ⇒ $'a$ ⇒ $'a$ *set*
  where *subspace-coset U x* = ((+) *x*) ‘ *U*

lemma *subspace-coset-class*:
  subspace-coset U x = {*y*. (*y*, *x*) ∈ *subspace-rel U*}
⟨*proof*⟩

context *Modules.module*
begin

lemma *subspace-rel-refl*:
  assumes *subspace U*
  shows (*x*, *x*) ∈ *subspace-rel U*
  ⟨*proof*⟩

lemma *subspace-rel-sym*:
  assumes *U*: *subspace U*
    and *xy*: (*x*, *y*) ∈ *subspace-rel U*
  shows (*y*, *x*) ∈ *subspace-rel U*
⟨*proof*⟩

lemma *subspace-rel-trans*:
assumes *U*: *subspace U*
and *xy*: $(x, y) \in$ *subspace-rel U*
and *yz*: $(y, z) \in$ *subspace-rel U*
shows $(x, z) \in$ *subspace-rel U*
⟨*proof*⟩

lemma *subspace-rel-equiv*:
assumes *U*: *subspace U*
shows *equiv UNIV* (*subspace-rel U*)
⟨*proof*⟩

end

locale *scalar-tower-pair* =
source: *compatible-scalar-actions rS sA rA* +
target: *compatible-scalar-actions rS sB rB*
for *rS* :: *'r*::*comm-ring-1* ⇒ *'s*::*comm-ring-1* ⇒ *'s*
and *sA* :: *'s* ⇒ *'a*::*comm-ring-1* ⇒ *'a*
and *rA* :: *'r* ⇒ *'a* ⇒ *'a*
and *sB* :: *'s* ⇒ *'b*::*comm-ring-1* ⇒ *'b*
and *rB* :: *'r* ⇒ *'b* ⇒ *'b*
begin

lemma *restrict-scalars-module-hom*:
assumes *module-hom sA sB f*
shows *module-hom rA rB* (*restrict-scalars f*)
⟨*proof*⟩

lemma *restrict-scalars-id* [*simp*]:
restrict-scalars id = *id*
⟨*proof*⟩

lemma *restrict-scalars-comp* [*simp*]:
restrict-scalars (*g* ∘ *f*) =
restrict-scalars g ∘ *restrict-scalars f*
⟨*proof*⟩

lemma *restrict-scalars-inj* [*simp*]:
inj (*restrict-scalars f*) ⟷ *inj f*
⟨*proof*⟩

lemma *restrict-scalars-surj* [*simp*]:
surj (*restrict-scalars f*) ⟷ *surj f*
⟨*proof*⟩

lemma *restrict-scalars-bij* [*simp*]:
bij (*restrict-scalars f*) ⟷ *bij f*
⟨*proof*⟩

lemma *restrict-scalars-kernel* [*simp*]:
$\{x.\ \textit{restrict-scalars}\ f\ x = 0\} = \{x.\ f\ x = 0\}$
⟨*proof*⟩

lemma *restrict-scalars-image* [*simp*]:
restrict-scalars f ‘ $X = f$ ‘ X
⟨*proof*⟩

lemma *restrict-scalars-range* [*simp*]:
range (*restrict-scalars* f) = *range* f
⟨*proof*⟩

lemma *source-subspace-restrict-scalars*:
assumes U: *source.SA.subspace* U
shows *source.RA.subspace* U
⟨*proof*⟩

lemma *target-subspace-restrict-scalars*:
assumes U: *target.SA.subspace* U
shows *target.RA.subspace* U
⟨*proof*⟩

lemma *restrict-scalars-kernel-subspace*:
assumes f: *module-hom sA sB f*
shows *source.RA.subspace* $\{x.\ \textit{restrict-scalars}\ f\ x = 0\}$
⟨*proof*⟩

lemma *restrict-scalars-image-subspace*:
assumes f: *module-hom sA sB f*
and U: *source.SA.subspace* U
shows *target.RA.subspace* (*restrict-scalars* f ‘ U)
⟨*proof*⟩

lemma *restrict-scalars-quotient*:
assumes f: *module-hom sA sB f*
and *maps*: f ‘ $U \subseteq V$
and *xy*: $(x,\ y) \in$ *subspace-rel* U
shows (*restrict-scalars* $f\ x$, *restrict-scalars* $f\ y$) ∈ *subspace-rel* V
⟨*proof*⟩

lemma *restrict-scalars-coset-image*:
assumes *module-hom sA sB f*
shows *restrict-scalars* f ‘ *subspace-coset* $U\ x$ =
subspace-coset (f ‘ U) ($f\ x$)
⟨*proof*⟩

end

locale *scalar-tower-pair-fields* =
  scalar-tower-pair rS sA rA sB rB
  for *rS* :: *'r*::*field* ⇒ *'s*::*field* ⇒ *'s*
    and *sA* :: *'s* ⇒ *'a*::*comm-ring-1* ⇒ *'a*
    and *rA* :: *'r* ⇒ *'a* ⇒ *'a*
    and *sB* :: *'s* ⇒ *'b*::*comm-ring-1* ⇒ *'b*
    and *rB* :: *'r* ⇒ *'b* ⇒ *'b*
begin

lemma *restrict-scalars-linear*:
  assumes *Vector-Spaces.linear sA sB f*
  shows *Vector-Spaces.linear rA rB* (*restrict-scalars f*)
  ⟨*proof*⟩

end

end
theory *Tower-Bases*
  imports *Restriction-of-Scalars*
begin

5 Composition of bases

An indexed basis below is only a convenient locale wrapper around Isabelle's existing set-valued notions of independence and span. It adds no competing basis predicate.

locale *indexed-basis* =
  Modules.module scale
  for *scale* :: *'r*::*comm-ring-1* ⇒ *'a*::*ab-group-add* ⇒ *'a* +
  fixes *basis* :: *'i*::*finite* ⇒ *'a*
  assumes *basis-inj*: *inj basis*
    and *independent-basis*: *independent* (*range basis*)
    and *span-basis*: *span* (*range basis*) = *UNIV*
begin

definition *coordinate* :: *'a* ⇒ *'i* ⇒ *'r*
  where *coordinate x i* = *representation* (*range basis*) *x* (*basis i*)

lemma *basis-mem-span* [*simp*]:
  basis i ∈ *span* (*range basis*)
  ⟨*proof*⟩

lemma *all-mem-span* [*simp*]:
  x ∈ *span* (*range basis*)
  ⟨*proof*⟩

lemma *coordinate-zero* [*simp*]:
  coordinate 0 i = *0*

⟨*proof*⟩

lemma *coordinate-add* [*simp*]:
coordinate (*x* + *y*) *i* = *coordinate x i* + *coordinate y i*
⟨*proof*⟩

lemma *coordinate-diff* [*simp*]:
coordinate (*x* − *y*) *i* = *coordinate x i* − *coordinate y i*
⟨*proof*⟩

lemma *coordinate-scale* [*simp*]:
coordinate (*scale r x*) *i* = *r* ∗ *coordinate x i*
⟨*proof*⟩

lemma *coordinate-sum*:
coordinate (∑ *x*∈*X*. *f x*) *i* =
(∑ *x*∈*X*. *coordinate* (*f x*) *i*)
⟨*proof*⟩

lemma *coordinate-basis* [*simp*]:
coordinate (*basis i*) *j* = (*if j* = *i then 1 else 0*)
⟨*proof*⟩

lemma *basis-expansion*:
(∑ *i*∈*UNIV*. *scale* (*coordinate x i*) (*basis i*)) = *x*
⟨*proof*⟩

lemma *coordinate-eq-iff*:
x = *y* ⟷ (∀ *i*. *coordinate x i* = *coordinate y i*)
⟨*proof*⟩

end

locale *indexed-algebra-tower* =
algebra-tower rS sA rA
for *rS* :: *'r*::*comm-ring-1* ⇒ *'s*::*comm-ring-1* ⇒ *'s*
and *sA* :: *'s* ⇒ *'a*::*comm-ring-1* ⇒ *'a*
and *rA* :: *'r* ⇒ *'a* ⇒ *'a* +
fixes *lower-basis* :: *'i*::*finite* ⇒ *'s*
and *upper-basis* :: *'j*::*finite* ⇒ *'a*
assumes *lower-basis-inj*: *inj lower-basis*
and *lower-basis-independent*: *RS.independent* (*range lower-basis*)
and *lower-basis-span*: *RS.span* (*range lower-basis*) = *UNIV*
and *upper-basis-inj*: *inj upper-basis*
and *upper-basis-independent*: *SA.independent* (*range upper-basis*)
and *upper-basis-span*: *SA.span* (*range upper-basis*) = *UNIV*
begin

sublocale *lower*: *indexed-basis rS lower-basis*

⟨*proof*⟩

sublocale *upper*: *indexed-basis sA upper-basis*
⟨*proof*⟩

definition *tower-basis* :: $('i \times 'j) \Rightarrow 'a$
where *tower-basis ij* =
sA (*lower-basis* (*fst ij*)) (*upper-basis* (*snd ij*))

definition *tower-coordinate* :: $'a \Rightarrow ('i \times 'j) \Rightarrow 'r$
where *tower-coordinate x ij* =
lower.coordinate (*upper.coordinate x* (*snd ij*)) (*fst ij*)

lemma *tower-coordinate-zero* [*simp*]:
tower-coordinate 0 ij = *0*
⟨*proof*⟩

lemma *tower-coordinate-add* [*simp*]:
tower-coordinate (*x* + *y*) *ij* =
tower-coordinate x ij + *tower-coordinate y ij*
⟨*proof*⟩

lemma *tower-coordinate-diff* [*simp*]:
tower-coordinate (*x* − *y*) *ij* =
tower-coordinate x ij − *tower-coordinate y ij*
⟨*proof*⟩

lemma *upper-coordinate-lower-scale*:
upper.coordinate (*rA r x*) *j* =
rS r (*upper.coordinate x j*)
⟨*proof*⟩

lemma *tower-coordinate-scale* [*simp*]:
tower-coordinate (*rA r x*) *ij* = *r* ∗ *tower-coordinate x ij*
⟨*proof*⟩

lemma *tower-coordinate-sum*:
tower-coordinate ($\sum x{\in}X.\ f\ x$) *ij* =
($\sum x{\in}X.$ *tower-coordinate* (*f x*) *ij*)
⟨*proof*⟩

lemma *tower-coordinate-basis* [*simp*]:
tower-coordinate (*tower-basis ij*) *kl* =
(*if kl* = *ij then 1 else 0*)
⟨*proof*⟩

lemma *tower-basis-inj*:
inj tower-basis
⟨*proof*⟩

lemma *tower-basis-span*:
RA.span (*range tower-basis*) = *UNIV*
⟨*proof*⟩

lemma *tower-basis-independent*:
RA.independent (*range tower-basis*)
⟨*proof*⟩

theorem *basis-tower*:
RA.independent (*range tower-basis*) ∧
RA.span (*range tower-basis*) = *UNIV*
⟨*proof*⟩

sublocale *composed*: *indexed-basis rA tower-basis*
⟨*proof*⟩

lemma *composed-coordinate*:
composed.coordinate x ij = *tower-coordinate x ij*
⟨*proof*⟩

end

end
theory *Finite-Dimensional-Towers*
imports *Tower-Bases*
Scalar-Restriction-Bridge
begin

6 Dimension in a tower

context *indexed-algebra-tower*
begin

theorem *finite-rank-tower*:
card (*range tower-basis*) = *card* (*range lower-basis*) ∗ *card* (*range upper-basis*)
⟨*proof*⟩

end

locale *indexed-field-tower* =
indexed-algebra-tower rS sA rA lower-basis upper-basis
for *rS* :: *'r*::*field* ⇒ *'s*::*field* ⇒ *'s*
and *sA* :: *'s* ⇒ *'a*::*comm-ring-1* ⇒ *'a*
and *rA* :: *'r* ⇒ *'a* ⇒ *'a*
and *lower-basis* :: *'i*::*finite* ⇒ *'s*
and *upper-basis* :: *'j*::*finite* ⇒ *'a*
begin

sublocale *lower-space*: *vector-space rS*
⟨*proof*⟩

sublocale *upper-space*: *vector-space sA*
⟨*proof*⟩

sublocale *direct-space*: *vector-space rA*
⟨*proof*⟩

theorem *finrank-tower*:
direct-space.dim UNIV =
lower-space.dim UNIV ∗ *upper-space.dim UNIV*
⟨*proof*⟩

end

The carrier-based formulation permits the top carrier to be a vector subspace, including the zero space; it does not assume that the top carrier is a field. The two finite-basis assumptions express finite dimensionality of the intermediate field over the base and of the vector space over the intermediate field.

context *scalar-restriction-tower*
begin

corollary *embedded-field-dimension-tower*:
assumes ∃ *B*. *lower.basis B* ∃ *C*. *upper.basis C*
shows ∃ *P*. *direct.basis P*
and *direct.dimension* = *lower.dimension* ∗ *upper.dimension*
⟨*proof*⟩

end

end
theory *Regular-Representation*
imports *Finite-Dimensional-Towers*
begin

7 Regular representation

context *commutative-algebra*
begin

definition *left-multiplication* :: *'a* ⇒ *'a* ⇒ *'a*
where *left-multiplication a x* = *a* ∗ *x*

lemma *left-multiplication-module-hom*:
module-hom scale scale (*left-multiplication a*)
⟨*proof*⟩

lemma *left-multiplication-zero* [*simp*]:
left-multiplication 0 x = 0
⟨*proof*⟩

lemma *left-multiplication-one* [*simp*]:
left-multiplication 1 = id
⟨*proof*⟩

lemma *left-multiplication-add*:
left-multiplication (a + b) x =
left-multiplication a x + left-multiplication b x
⟨*proof*⟩

lemma *left-multiplication-mult*:
*left-multiplication (a * b) =*
left-multiplication a ∘ left-multiplication b
⟨*proof*⟩

definition *regular-representation* :: *'a ⇒ 'a ⇒ 'a*
where *regular-representation a = left-multiplication a*

lemma *regular-representation-module-hom*:
module-hom scale scale (*regular-representation a*)
⟨*proof*⟩

lemma *regular-representation-add*:
regular-representation (a + b) x =
regular-representation a x + regular-representation b x
⟨*proof*⟩

lemma *regular-representation-mult*:
*regular-representation (a * b) =*
regular-representation a ∘ regular-representation b
⟨*proof*⟩

lemma *regular-representation-one* [*simp*]:
regular-representation 1 = id
⟨*proof*⟩

end

context *indexed-basis*
begin

definition *matrix-of* :: *('a ⇒ 'a) ⇒ 'r^'i^'i*
where *matrix-of f = (χ i j. coordinate (f (basis j)) i)*

lemma *matrix-of-component* [*simp*]:

matrix-of f $ i $ j = coordinate (f (basis j)) i
⟨*proof*⟩

lemma *matrix-of-id* [*simp*]:
matrix-of id = mat 1
⟨*proof*⟩

lemma *matrix-of-zero* [*simp*]:
matrix-of (λ-. *0*) = *0*
⟨*proof*⟩

lemma *matrix-of-add*:
matrix-of (λx. *f x + g x*) = *matrix-of f + matrix-of g*
⟨*proof*⟩

lemma *matrix-of-comp*:
assumes *f*: *module-hom scale scale f*
and *g*: *module-hom scale scale g*
shows *matrix-of* (*g* ∘ *f*) = *matrix-of g ** *matrix-of f*
⟨*proof*⟩

end

locale *indexed-basis-pair* =
first: *indexed-basis scale first-basis* +
second: *indexed-basis scale second-basis*
for *scale* :: *'r*::*comm-ring-1* ⇒ *'a*::*ab-group-add* ⇒ *'a*
and *first-basis* :: *'i*::*finite* ⇒ *'a*
and *second-basis* :: *'i* ⇒ *'a*
begin

definition *to-second* :: *'r^'i^'i*
where *to-second* =
(χ *i j*. *second.coordinate* (*first-basis j*) *i*)

definition *to-first* :: *'r^'i^'i*
where *to-first* =
(χ *i j*. *first.coordinate* (*second-basis j*) *i*)

definition *first-to-second-matrix* :: (*'a* ⇒ *'a*) ⇒ *'r^'i^'i*
where *first-to-second-matrix f* =
(χ *i j*. *second.coordinate* (*f* (*first-basis j*)) *i*)

lemma *to-second-component* [*simp*]:
to-second $ i $ j = second.coordinate (*first-basis j*) *i*
⟨*proof*⟩

lemma *to-first-component* [*simp*]:
to-first $ i $ j = first.coordinate (*second-basis j*) *i*

⟨*proof*⟩

lemma *first-to-second-component* [*simp*]:
first-to-second-matrix f $ *i* $ *j* =
second.coordinate (*f* (*first-basis j*)) *i*
⟨*proof*⟩

lemma *to-second-to-first*:
to-second ∗∗ *to-first* = *mat 1*
⟨*proof*⟩

lemma *to-first-to-second*:
to-first ∗∗ *to-second* = *mat 1*
⟨*proof*⟩

lemma *first-to-second-matrix-left*:
assumes *f*: *module-hom scale scale f*
shows *first-to-second-matrix f* =
to-second ∗∗ *first.matrix-of f*
⟨*proof*⟩

lemma *second-matrix-from-mixed*:
assumes *f*: *module-hom scale scale f*
shows *second.matrix-of f* =
first-to-second-matrix f ∗∗ *to-first*
⟨*proof*⟩

theorem *matrix-change-basis*:
assumes *module-hom scale scale f*
shows *second.matrix-of f* =
(*to-second* ∗∗ *first.matrix-of f*) ∗∗ *to-first*
⟨*proof*⟩

end

locale *indexed-algebra* =
commutative-algebra scale
for *scale* :: *'r*::*comm-ring-1* ⇒ *'a*::*comm-ring-1* ⇒ *'a* +
fixes *basis* :: *'i*::*finite* ⇒ *'a*
assumes *basis-inj*: *inj basis*
and *basis-independent*: *independent* (*range basis*)
and *basis-span*: *span* (*range basis*) = *UNIV*
begin

sublocale *coordinates*: *indexed-basis scale basis*
⟨*proof*⟩

definition *regular-matrix* :: *'a* ⇒ *'r*^*'i*^*'i*
where *regular-matrix a* =

coordinates.matrix-of (*left-multiplication a*)

lemma *regular-matrix-zero* [*simp*]:
regular-matrix 0 = 0
⟨*proof*⟩

lemma *regular-matrix-add*:
regular-matrix (*a + b*) = *regular-matrix a + regular-matrix b*
⟨*proof*⟩

lemma *regular-matrix-scale*:
regular-matrix (*scale r a*) =
(*χ i j. r ∗ regular-matrix a $ i $ j*)
⟨*proof*⟩

lemma *regular-matrix-one* [*simp*]:
regular-matrix 1 = mat 1
⟨*proof*⟩

lemma *regular-matrix-scalar-embed*:
regular-matrix (*scale r 1*) = *mat r*
⟨*proof*⟩

lemma *regular-matrix-mult*:
regular-matrix (*a ∗ b*) = *regular-matrix a ∗∗ regular-matrix b*
⟨*proof*⟩

end

Finite basis index types are bookkeeping rather than mathematical data. The following locale transports an indexed algebra structure along an explicit bijection between finite index types, so later trace and determinant results can compare bases with genuinely different index types.

locale *indexed-algebra-reindex* =
original: *indexed-algebra scale original-basis*
for *scale* :: *'r*::*comm-ring-1* ⇒ *'a*::*comm-ring-1* ⇒ *'a*
and *original-basis* :: *'j*::*finite* ⇒ *'a* +
fixes *index-iso* :: *'i*::*finite* ⇒ *'j*
assumes *index-iso-bij*: *bij index-iso*
begin

definition *reindexed-basis* :: *'i* ⇒ *'a*
where *reindexed-basis i = original-basis* (*index-iso i*)

lemma *reindexed-basis-range*:
range reindexed-basis = range original-basis
⟨*proof*⟩

lemma *reindexed-basis-inj*:

inj reindexed-basis
⟨*proof*⟩

lemma *reindexed-basis-independent*:
original.independent (*range reindexed-basis*)
⟨*proof*⟩

lemma *reindexed-basis-span*:
original.span (*range reindexed-basis*) = *UNIV*
⟨*proof*⟩

sublocale *reindexed*: *indexed-algebra scale reindexed-basis*
⟨*proof*⟩

lemma *coordinate-reindexed*:
reindexed.coordinates.coordinate x i =
original.coordinates.coordinate x (*index-iso i*)
⟨*proof*⟩

lemma *regular-matrix-reindexed-component* [*simp*]:
reindexed.regular-matrix a $ *i* $ *k* =
original.regular-matrix a $ *index-iso i* $ *index-iso k*
⟨*proof*⟩

end

locale *indexed-algebra-isomorphism* =
source: *indexed-algebra source-scale source-basis* +
target: *indexed-algebra target-scale target-basis*
for *source-scale* :: *'r*::*comm-ring-1* ⇒ *'a*::*comm-ring-1* ⇒ *'a*
and *source-basis* :: *'i*::*finite* ⇒ *'a*
and *target-scale* :: *'r* ⇒ *'b*::*comm-ring-1* ⇒ *'b*
and *target-basis* :: *'i* ⇒ *'b* +
fixes *iso* :: *'a* ⇒ *'b*
assumes *iso-module-hom*: *module-hom source-scale target-scale iso*
and *iso-mult*: *iso* (*x* ∗ *y*) = *iso x* ∗ *iso y*
and *iso-one*: *iso 1* = *1*
and *iso-bij*: *bij iso*
and *iso-basis*: *iso* (*source-basis i*) = *target-basis i*
begin

interpretation *iso*: *module-hom source-scale target-scale iso*
⟨*proof*⟩

lemma *iso-basis-expansion*:
(∑ *i*∈*UNIV*. *target-scale* (*source.coordinates.coordinate x i*) (*target-basis i*)) =
iso x
⟨*proof*⟩

lemma *coordinate-iso*:
target.coordinates.coordinate (*iso x*) *i* =
source.coordinates.coordinate x i
⟨*proof*⟩

lemma *regular-matrix-iso*:
target.regular-matrix (*iso a*) = *source.regular-matrix a*
⟨*proof*⟩

end

locale *indexed-algebra-isomorphism-general* =
source: *indexed-algebra source-scale source-basis* +
target: *indexed-algebra target-scale target-basis*
for *source-scale* :: *'r*::*comm-ring-1* ⇒ *'a*::*comm-ring-1* ⇒ *'a*
and *source-basis* :: *'i*::*finite* ⇒ *'a*
and *target-scale* :: *'r* ⇒ *'b*::*comm-ring-1* ⇒ *'b*
and *target-basis* :: *'j*::*finite* ⇒ *'b* +
fixes *iso* :: *'a* ⇒ *'b*
and *index-iso* :: *'i* ⇒ *'j*
assumes *iso-module-hom*: *module-hom source-scale target-scale iso*
and *iso-mult*: *iso* (*x* ∗ *y*) = *iso x* ∗ *iso y*
and *iso-one*: *iso 1* = *1*
and *iso-bij*: *bij iso*
and *index-iso-bij*: *bij index-iso*
begin

interpretation *iso*: *module-hom source-scale target-scale iso*
⟨*proof*⟩

definition *transported-basis* :: *'i* ⇒ *'b*
where *transported-basis i* = *iso* (*source-basis i*)

lemma *transported-basis-range*:
range transported-basis = *iso* ‘ *range source-basis*
⟨*proof*⟩

lemma *transported-basis-inj*:
inj transported-basis
⟨*proof*⟩

lemma *transported-basis-independent*:
target.independent (*range transported-basis*)
⟨*proof*⟩

lemma *transported-basis-span*:
target.span (*range transported-basis*) = *UNIV*
⟨*proof*⟩

sublocale *transported*: *indexed-algebra target-scale transported-basis*
⟨*proof*⟩

sublocale *basis-iso*: *indexed-algebra-isomorphism*
source-scale source-basis target-scale transported-basis iso
⟨*proof*⟩

sublocale *reindexed-target*:
indexed-algebra-reindex target-scale target-basis index-iso
⟨*proof*⟩

end

end
theory *Algebra-Trace*
imports *Regular-Representation*
begin

8 Algebra trace

context *indexed-algebra*
begin

definition *algebra-trace* :: $'a \Rightarrow 'r$
where *algebra-trace a* = *trace* (*regular-matrix a*)

lemma *algebra-trace-zero* [*simp*]:
algebra-trace 0 = *0*
⟨*proof*⟩

lemma *algebra-trace-add*:
algebra-trace (*a* + *b*) = *algebra-trace a* + *algebra-trace b*
⟨*proof*⟩

lemma *algebra-trace-sum*:
algebra-trace ($\sum x{\in}X.$ *f x*) =
($\sum x{\in}X.$ *algebra-trace* (*f x*))
⟨*proof*⟩

lemma *algebra-trace-scale*:
algebra-trace (*scale r a*) = *r* ∗ *algebra-trace a*
⟨*proof*⟩

theorem *algebra-trace-linear*:
module-hom scale ((∗) :: $'r \Rightarrow 'r \Rightarrow 'r$) *algebra-trace*
⟨*proof*⟩

lemma *algebra-trace-scalar*:
algebra-trace (*scale r 1*) = *of-nat CARD*($'i$) ∗ *r*

⟨*proof*⟩

end

context *indexed-basis-pair*
begin

theorem *trace-matrix-change-basis*:
assumes *module-hom scale scale f*
shows *trace* (*second.matrix-of f*) = *trace* (*first.matrix-of f*)
⟨*proof*⟩

end

context *indexed-algebra-reindex*
begin

theorem *algebra-trace-reindexed*:
reindexed.algebra-trace a = *original.algebra-trace a*
⟨*proof*⟩

end

context *indexed-algebra*
begin

theorem *algebra-trace-basis-independent*:
fixes *other-basis* :: *'i* ⇒ *'a*
assumes *other-inj*: *inj other-basis*
and *other-independent*: *independent* (*range other-basis*)
and *other-span*: *span* (*range other-basis*) = *UNIV*
shows *algebra-trace a* =
indexed-algebra.algebra-trace scale other-basis a
⟨*proof*⟩

theorem *algebra-trace-basis-independent-bij*:
fixes *other-basis* :: *'j*::*finite* ⇒ *'a*
and *index-iso* :: *'i* ⇒ *'j*
assumes *other-inj*: *inj other-basis*
and *other-independent*: *independent* (*range other-basis*)
and *other-span*: *span* (*range other-basis*) = *UNIV*
and *index-iso-bij*: *bij index-iso*
shows *algebra-trace a* =
indexed-algebra.algebra-trace scale other-basis a
⟨*proof*⟩

end

context *indexed-algebra-isomorphism*

begin

theorem *algebra-trace-iso*:
target.algebra-trace (*iso a*) = *source.algebra-trace a*
⟨*proof*⟩

end

context *indexed-algebra-isomorphism-general*
begin

theorem *algebra-trace-iso*:
target.algebra-trace (*iso a*) = *source.algebra-trace a*
⟨*proof*⟩

end

context *indexed-algebra-tower*
begin

sublocale *lower-algebra*: *indexed-algebra rS lower-basis*
⟨*proof*⟩

sublocale *upper-algebra*: *indexed-algebra sA upper-basis*
⟨*proof*⟩

sublocale *tower-algebra*: *indexed-algebra rA tower-basis*
⟨*proof*⟩

lemma *restricted-matrix-diagonal*:
assumes *f*: *module-hom sA sA f*
shows *composed.matrix-of f* $ (*i*, *j*) $ (*i*, *j*) = *lower-algebra.regular-matrix*
(*upper.matrix-of f* $ *j* $ *j*) $ *i* $ *i*
⟨*proof*⟩

theorem *trace-restrict-scalars*:
assumes *f*: *module-hom sA sA f*
shows *trace* (*composed.matrix-of f*) =
lower-algebra.algebra-trace
(*trace* (*upper.matrix-of f*))
⟨*proof*⟩

theorem *algebra-trace-trans*:
tower-algebra.algebra-trace a =
lower-algebra.algebra-trace (*upper-algebra.algebra-trace a*)
⟨*proof*⟩

end

end
theory *Algebra-Norm*
imports *Algebra-Trace*
begin

9 Algebra norm

Simultaneously reindexing the rows and columns of a matrix along a bijection does not change its determinant. Isabelle's *map-permutation* transports the permutations in the Leibniz formula and preserves their signs.

lemma *bij-betw-map-permutations*:
assumes *h*: *bij-betw h A B*
shows *bij-betw* (*map-permutation A h*)
{*p*. *p permutes A*} {*q*. *q permutes B*}
⟨*proof*⟩

theorem *det-reindex-bij*:
fixes *A* :: *'r*::*comm-ring-1*^*'j*::*finite*^*'j*
and *index-iso* :: *'i*::*finite* ⇒ *'j*
assumes *index-iso-bij*: *bij index-iso*
shows *det* (*χ i k*. *A* $ *index-iso i* $ *index-iso k*) = *det A*
⟨*proof*⟩

The Cartesian-space library provides matrix induction by elementary operations only for real matrices. The proof uses no order or topology, so we expose the same argument over an arbitrary field.

lemma *induct-matrix-row-operations-field*:
fixes *P* :: *'a*::*field*^*'n*^*'n* ⇒ *bool*
assumes *zero-row*: ⋀*A i*. *row i A* = *0* ⟹ *P A*
and *diagonal*: ⋀*A*. (⋀*i j*. *i* ≠ *j* ⟹ *A*$*i*$*j* = *0*) ⟹ *P A*
and *swap-cols*: ⋀*A m n*. ⟦*P A*; *m* ≠ *n*⟧ ⟹
P (*χ i j*. *A* $ *i* $ *Transposition.transpose m n j*)
and *row-op*: ⋀*A m n c*. ⟦*P A*; *m* ≠ *n*⟧ ⟹
P (*χ i*. *if i* = *m then row m A* + *c* ∗*s row n A else row i A*)
shows *P A*
⟨*proof*⟩

lemma *induct-matrix-elementary-field*:
fixes *P* :: *'a*::*field*^*'n*^*'n* ⇒ *bool*
assumes *mult*: ⋀*A B*. ⟦*P A*; *P B*⟧ ⟹ *P* (*A* ∗∗ *B*)
and *zero-row*: ⋀*A i*. *row i A* = *0* ⟹ *P A*
and *diagonal*: ⋀*A*. (⋀*i j*. *i* ≠ *j* ⟹ *A*$*i*$*j* = *0*) ⟹ *P A*
and *swap*: ⋀*m n*. *m* ≠ *n* ⟹
P (*χ i j*. *mat 1* $ *i* $ *Transposition.transpose m n j*)
and *add*: ⋀*m n c*. *m* ≠ *n* ⟹
P (*χ i j*.
if i = *m* ∧ *j* = *n then c else of-bool* (*i* = *j*))
shows *P A*

⟨*proof*⟩

definition *block-embed* ::
  'j::*finite* ⇒ *'r*::*zero-neq-one*^*'i*::*finite*^*'i* ⇒
    'r^(*'i* × *'j*)^(*'i* × *'j*)
 where *block-embed j B* =
  (χ *ik hl*.
    if snd ik = *j* ∧ *snd hl* = *j*
    then B $ *fst ik* $ *fst hl*
    else of-bool (*ik* = *hl*))

lemma *sum-UNIV-prod*:
 fixes *f* :: (*'a*::*finite* × *'b*::*finite*) ⇒ *'c*::*comm-monoid-add*
 shows (∑ *x*∈*UNIV*. *f x*) =
  (∑ *a*∈*UNIV*. ∑ *b*∈*UNIV*. *f* (*a*, *b*))
 ⟨*proof*⟩

lemma *cartesian-matrix-mult-component*:
 fixes *B C* :: *'r*::*comm-ring-1*^*'i*::*finite*^*'i*
 shows (*B* ** *C*) $ *i* $ *h* = (∑ *a*∈*UNIV*. *B* $ *i* $ *a* * *C* $ *a* $ *h*)
 ⟨*proof*⟩

lemma *block-embed-component* [*simp*]:
 fixes *j* :: *'j*::*finite*
  and *B* :: *'r*::*zero-neq-one*^*'i*::*finite*^*'i*
 shows *block-embed j B* $ (*i*, *k*) $ (*h*, *l*) =
  (*if k* = *j* ∧ *l* = *j then B* $ *i* $ *h*
   else of-bool ((*i*, *k*) = (*h*, *l*)))
 ⟨*proof*⟩

lemma *block-embed-one* [*simp*]:
 fixes *j* :: *'j*::*finite*
 shows *block-embed j* (*mat 1* :: *'r*::*comm-ring-1*^*'i*^*'i*) =
  (*mat 1* :: *'r*^(*'i* × *'j*)^(*'i* × *'j*))
 ⟨*proof*⟩

lemma *sum-of-bool-left*:
 fixes *f* :: *'a*::*finite* ⇒ *'r*::*comm-ring-1*
 shows (∑ *x*∈*UNIV*. *of-bool* (*x* = *p*) * *f x*) = *f p*
 ⟨*proof*⟩

lemma *sum-of-bool-right*:
 fixes *f* :: *'a*::*finite* ⇒ *'r*::*comm-ring-1*
 shows (∑ *x*∈*UNIV*. *f x* * *of-bool* (*x* = *p*)) = *f p*
 ⟨*proof*⟩

lemma *prod-if-eq*:
 fixes *j* :: *'a*::*finite*
  and *x* :: *'r*::*comm-monoid-mult*

shows $(\prod k{\in}UNIV.\ \mathit{if}\ k = j\ \mathit{then}\ x\ \mathit{else}\ 1) = x$
⟨*proof*⟩

lemma *block-embed-row-outside*:
fixes $j :: 'j{::}\mathit{finite}$
and $B :: 'r{::}\mathit{zero\text{-}neq\text{-}one}\hat{}'i{::}\mathit{finite}\hat{}'i$
assumes $k \neq j$
shows $\mathit{block\text{-}embed}\ j\ B\ \$\ (i,\ k)\ \$\ (h,\ l) =$
$\mathit{of\text{-}bool}\ ((i,\ k) = (h,\ l))$
⟨*proof*⟩

lemma *block-embed-column-outside*:
fixes $j :: 'j{::}\mathit{finite}$
and $B :: 'r{::}\mathit{zero\text{-}neq\text{-}one}\hat{}'i{::}\mathit{finite}\hat{}'i$
assumes $l \neq j$
shows $\mathit{block\text{-}embed}\ j\ B\ \$\ (i,\ k)\ \$\ (h,\ l) =$
$\mathit{of\text{-}bool}\ ((i,\ k) = (h,\ l))$
⟨*proof*⟩

lemma *block-embed-mult*:
fixes $j :: 'j{::}\mathit{finite}$
and $B\ C :: 'r{::}\mathit{comm\text{-}ring\text{-}1}\hat{}'i{::}\mathit{finite}\hat{}'i$
shows $\mathit{block\text{-}embed}\ j\ (B\ **\ C) = \mathit{block\text{-}embed}\ j\ B\ **\ \mathit{block\text{-}embed}\ j\ C$
⟨*proof*⟩

theorem *det-block-embed*:
fixes $j :: 'j{::}\mathit{finite}$
and $B :: 'r{::}\mathit{field}\hat{}'i{::}\mathit{finite}\hat{}'i$
shows $\det\ (\mathit{block\text{-}embed}\ j\ B) = \det\ B$
⟨*proof*⟩

definition *block-swap-entries* ::
$'i{::}\mathit{finite}\ \mathit{set} \Rightarrow 'j{::}\mathit{finite} \Rightarrow 'j \Rightarrow$
$'r{::}\mathit{comm\text{-}ring\text{-}1}\hat{}('i \times 'j)\hat{}('i \times 'j)$
where $\mathit{block\text{-}swap\text{-}entries}\ X\ m\ n =$
$(\chi\ ik\ hl.\ \mathit{mat}\ 1\ \$\ ik\ \$$
$(\mathit{if}\ \mathit{fst}\ hl \in X$
$\mathit{then}\ (\mathit{fst}\ hl,\ \mathit{Transposition.transpose}\ m\ n\ (\mathit{snd}\ hl))$
$\mathit{else}\ hl))$

definition *block-swap* ::
$'j{::}\mathit{finite} \Rightarrow 'j \Rightarrow$
$'r{::}\mathit{comm\text{-}ring\text{-}1}\hat{}('i{::}\mathit{finite} \times 'j)\hat{}('i \times 'j)$
where $\mathit{block\text{-}swap}\ m\ n = \mathit{block\text{-}swap\text{-}entries}\ UNIV\ m\ n$

lemma *block-swap-entries-empty* [*simp*]:
$\mathit{block\text{-}swap\text{-}entries}\ \{\}\ m\ n =$
$(\mathit{mat}\ 1 :: 'r{::}\mathit{comm\text{-}ring\text{-}1}\hat{}('i{::}\mathit{finite} \times 'j{::}\mathit{finite})\hat{}$
$('i \times 'j))$

⟨*proof*⟩

lemma *block-swap-entries-insert*:
fixes $X :: 'i$::*finite set*
and $m\ n :: 'j$::*finite*
assumes $a \notin X$
shows *block-swap-entries* (*insert a X*) *m n* =
(χ *ik hl*. *block-swap-entries X m n* $ *ik* $
Transposition.transpose (*a*, *m*) (*a*, *n*) *hl*)
⟨*proof*⟩

lemma *det-block-swap-entries*:
fixes $X :: 'i$::*finite set*
and $m\ n :: 'j$::*finite*
assumes $m \neq n$
shows *det* (*block-swap-entries X m n* ::
$'r$::*field*^($'i \times 'j$)^($'i \times 'j$)) =
(−*1*) ^ *card X*
⟨*proof*⟩

theorem *det-block-swap*:
fixes $m\ n :: 'j$::*finite*
assumes $m \neq n$
shows *det* (*block-swap m n* ::
$'r$::*field*^($'i$::*finite* $\times$ $'j$)^($'i \times 'j$)) =
(−*1*) ^ *CARD*($'i$)
⟨*proof*⟩

definition *block-shear-entries* ::
($'i$::*finite* $\times$ $'i$) *set* ⇒ $'j$::*finite* ⇒ $'j$ ⇒
$'r$::*comm-ring-1*^$'i$^$'i$ ⇒
$'r$^($'i \times 'j$)^($'i \times 'j$)
where *block-shear-entries X m n B* =
(χ *ik hl*.
if snd ik = *m* ∧ *snd hl* = *n* ∧ (*fst ik*, *fst hl*) ∈ *X*
then B $ *fst ik* $ *fst hl*
else of-bool (*ik* = *hl*))

definition *block-shear* ::
$'j$::*finite* ⇒ $'j$ ⇒ $'r$::*comm-ring-1*^$'i$::*finite*^$'i$ ⇒
$'r$^($'i \times 'j$)^($'i \times 'j$)
where *block-shear m n B* = *block-shear-entries UNIV m n B*

lemma *block-shear-entries-empty* [*simp*]:
block-shear-entries {} *m n B* =
(*mat 1* :: $'r$::*comm-ring-1*^($'i$::*finite* $\times$ $'j$::*finite*)^
($'i \times 'j$))
⟨*proof*⟩

lemma *block-shear-entries-insert*:
fixes $X :: ('i::finite \times 'i)\ set$
and $m\ n :: 'j::finite$
and $B :: 'r::comm\text{-}ring\text{-}1\hat{}'i\hat{}'i$
assumes *pair-out*: $(i, h) \notin X$
and *mn*: $m \neq n$
shows *block-shear-entries* $(insert\ (i, h)\ X)\ m\ n\ B =$
$(\chi\ ik.$
if $ik = (i, m)$
then row (i, m) (*block-shear-entries* $X\ m\ n\ B$) +
$(B\ \$\ i\ \$\ h) *s$ *row* (h, n) (*block-shear-entries* $X\ m\ n\ B$)
else row ik (*block-shear-entries* $X\ m\ n\ B$))
⟨*proof*⟩

lemma *det-block-shear-entries*:
fixes $X :: ('i::finite \times 'i)\ set$
and $m\ n :: 'j::finite$
and $B :: 'r::field\hat{}'i\hat{}'i$
assumes $m \neq n$
shows *det* (*block-shear-entries* $X\ m\ n\ B$) = *1*
⟨*proof*⟩

theorem *det-block-shear*:
fixes $m\ n :: 'j::finite$
and $B :: 'r::field\hat{}'i::finite\hat{}'i$
assumes $m \neq n$
shows *det* (*block-shear m n B*) = *1*
⟨*proof*⟩

context *indexed-algebra*
begin

lemma *regular-matrix-sum*:
regular-matrix $(\sum x{\in}X.\ f\ x) =$
$(\sum x{\in}X.$ *regular-matrix* $(f\ x))$
⟨*proof*⟩

definition *algebra-norm* :: $'a \Rightarrow 'r$
where *algebra-norm a* = *det* (*regular-matrix a*)

lemma *algebra-norm-zero* [*simp*]:
algebra-norm 0 = *0*
⟨*proof*⟩

lemma *algebra-norm-one* [*simp*]:
algebra-norm 1 = *1*
⟨*proof*⟩

theorem *algebra-norm-mult*:

algebra-norm $(a * b)$ = *algebra-norm* a $*$ *algebra-norm* b
⟨*proof*⟩

lemma *algebra-norm-scalar*:
algebra-norm (*scale* r 1) = $r \hat{} CARD('i)$
⟨*proof*⟩

end

context *indexed-algebra-reindex*
begin

theorem *algebra-norm-reindexed*:
reindexed.algebra-norm a = *original.algebra-norm* a
⟨*proof*⟩

end

context *indexed-basis-pair*
begin

theorem *det-matrix-change-basis*:
assumes *module-hom scale scale f*
shows *det* (*second.matrix-of f*) = *det* (*first.matrix-of f*)
⟨*proof*⟩

end

context *indexed-algebra*
begin

theorem *algebra-norm-basis-independent*:
fixes *other-basis* :: $'i \Rightarrow 'a$
assumes *other-inj*: *inj other-basis*
and *other-independent*: *independent* (*range other-basis*)
and *other-span*: *span* (*range other-basis*) = *UNIV*
shows *algebra-norm a* =
indexed-algebra.algebra-norm scale other-basis a
⟨*proof*⟩

theorem *algebra-norm-basis-independent-bij*:
fixes *other-basis* :: $'j$::*finite* $\Rightarrow 'a$
and *index-iso* :: $'i \Rightarrow 'j$
assumes *other-inj*: *inj other-basis*
and *other-independent*: *independent* (*range other-basis*)
and *other-span*: *span* (*range other-basis*) = *UNIV*
and *index-iso-bij*: *bij index-iso*
shows *algebra-norm a* =
indexed-algebra.algebra-norm scale other-basis a

⟨*proof*⟩

end

context *indexed-algebra-isomorphism*
begin

theorem *algebra-norm-iso*:
target.algebra-norm (*iso a*) = *source.algebra-norm a*
⟨*proof*⟩

end

context *indexed-algebra-isomorphism-general*
begin

theorem *algebra-norm-iso*:
target.algebra-norm (*iso a*) = *source.algebra-norm a*
⟨*proof*⟩

end

locale *indexed-field-algebra-tower* =
indexed-algebra-tower rS sA rA lower-basis upper-basis
for *rS* :: *'r*::*field* ⇒ *'s*::*field* ⇒ *'s*
and *sA* :: *'s* ⇒ *'a*::*comm-ring-1* ⇒ *'a*
and *rA* :: *'r* ⇒ *'a* ⇒ *'a*
and *lower-basis* :: *'i*::*finite* ⇒ *'s*
and *upper-basis* :: *'j*::*finite* ⇒ *'a*
begin

definition *restricted-matrix* ::
's^'j^'j ⇒ *'r^('i* × *'j)^('i* × *'j)*
where *restricted-matrix M* =
(*χ ik hl*.
lower-algebra.regular-matrix (*M* $ *snd ik* $ *snd hl*) $
fst ik $ *fst hl*)

lemma *restricted-matrix-component* [*simp*]:
restricted-matrix M $ (*i*, *j*) $ (*h*, *l*) =
lower-algebra.regular-matrix (*M* $ *j* $ *l*) $ *i* $ *h*
⟨*proof*⟩

lemma *restricted-matrix-zero* [*simp*]:
restricted-matrix 0 = *0*
⟨*proof*⟩

lemma *restricted-matrix-one* [*simp*]:
restricted-matrix (*mat 1*) = *mat 1*

⟨*proof*⟩

lemma *restricted-matrix-mult*:
restricted-matrix (*B* ** *C*) =
restricted-matrix B ** *restricted-matrix C*
⟨*proof*⟩

definition *diagonal-factor* :: *'j* ⇒ *'s* ⇒ *'s^'j^'j*
where *diagonal-factor j x* =
(χ *k l*. *if k* = *j* ∧ *l* = *j then x else of-bool* (*k* = *l*))

definition *partial-diagonal* :: *'j set* ⇒ *'s^'j^'j* ⇒ *'s^'j^'j*
where *partial-diagonal X M* =
(χ *k l*.
if k = *l*
then if k ∈ *X then M* $ *k* $ *k else 1*
else 0)

lemma *det-diagonal-factor*:
det (*diagonal-factor j x*) = *x*
⟨*proof*⟩

lemma *diagonal-factor-component-alt*:
diagonal-factor j x $ *k* $ *l* =
of-bool (*k* = *l*) ∗ (*if k* = *j then x else 1*)
⟨*proof*⟩

lemma *restricted-diagonal-factor*:
restricted-matrix (*diagonal-factor j x*) =
block-embed j (*lower-algebra.regular-matrix x*)
⟨*proof*⟩

lemma *partial-diagonal-empty* [*simp*]:
partial-diagonal {} *M* = *mat 1*
⟨*proof*⟩

lemma *partial-diagonal-UNIV*:
assumes *diagonal*: ⋀*k l*. *k* ≠ *l* ⟹ *M* $ *k* $ *l* = *0*
shows *partial-diagonal UNIV M* = *M*
⟨*proof*⟩

lemma *partial-diagonal-insert*:
assumes *j* ∉ *X*
shows *partial-diagonal* (*insert j X*) *M* =
diagonal-factor j (*M* $ *j* $ *j*) ** *partial-diagonal X M*
⟨*proof*⟩

lemma *restricted-swap*:
assumes *m* ≠ *n*

shows *restricted-matrix*
(χ *j l*. *mat 1* $ *j* $ *Transposition.transpose m n l*) =
block-swap m n
⟨*proof*⟩

lemma *restricted-shear*:
assumes $m \neq n$
shows *restricted-matrix*
(χ *j l*. *if j* = *m* ∧ *l* = *n then c else of-bool* (*j* = *l*)) =
block-shear m n (*lower-algebra.regular-matrix c*)
⟨*proof*⟩

theorem *det-restrict-scalars*:
det (*restricted-matrix M*) =
lower-algebra.algebra-norm (*det M*)
⟨*proof*⟩

theorem *matrix-restrict-scalars*:
assumes *f*: *module-hom sA sA f*
shows *composed.matrix-of f* =
restricted-matrix (*upper.matrix-of f*)
⟨*proof*⟩

theorem *algebra-norm-trans*:
tower-algebra.algebra-norm a =
lower-algebra.algebra-norm (*upper-algebra.algebra-norm a*)
⟨*proof*⟩

end

end
theory *Algebra-Towers-Examples*
imports
Algebra-Norm
HOL−Computational-Algebra.Primes
begin

10 Examples

10.1 Finite-field towers

datatype *gf4* = *GF4 bool bool*

instantiation *gf4* :: *field*
begin

definition *zero-gf4* :: *gf4*
where *zero-gf4* = *GF4 False False*

definition *one-gf4* :: *gf4*
  where *one-gf4* = *GF4 True False*

fun *plus-gf4* :: *gf4* ⇒ *gf4* ⇒ *gf4*
  where *plus-gf4* (*GF4 a b*) (*GF4 c d*) =
    GF4 (*a* ≠ *c*) (*b* ≠ *d*)

fun *uminus-gf4* :: *gf4* ⇒ *gf4*
  where *uminus-gf4* (*GF4 a b*) = *GF4 a b*

definition *minus-gf4* :: *gf4* ⇒ *gf4* ⇒ *gf4*
  where *minus-gf4 x y* = *x* + *y*

fun *times-gf4* :: *gf4* ⇒ *gf4* ⇒ *gf4*
  where *times-gf4* (*GF4 a b*) (*GF4 c d*) =
    GF4 ((*a* ∧ *c*) ≠ (*b* ∧ *d*))
      (((*a* ∧ *d*) ≠ (*b* ∧ *c*)) ≠ (*b* ∧ *d*))

fun *inverse-gf4* :: *gf4* ⇒ *gf4*
  where
    inverse-gf4 (*GF4 False False*) = *GF4 False False*
  | *inverse-gf4* (*GF4 True False*) = *GF4 True False*
  | *inverse-gf4* (*GF4 False True*) = *GF4 True True*
  | *inverse-gf4* (*GF4 True True*) = *GF4 False True*

definition *divide-gf4* :: *gf4* ⇒ *gf4* ⇒ *gf4*
  where *divide-gf4 x y* = *x* ∗ *inverse y*

lemma *inverse-times-gf4*:
  assumes *x* ≠ (*0* :: *gf4*)
  shows *inverse x* ∗ *x* = *1*
⟨*proof*⟩

instance
  ⟨*proof*⟩

end

instance *gf4* :: *finite-field*
⟨*proof*⟩

locale *finite-field-tower-example* =
  indexed-field-algebra-tower rS sA rA lower-basis upper-basis
  for *rS* :: *'r*::*finite-field* ⇒ *'s*::*finite-field* ⇒ *'s*
    and *sA* :: *'s* ⇒ *'a*::*finite-field* ⇒ *'a*
    and *rA* :: *'r* ⇒ *'a* ⇒ *'a*
    and *lower-basis* :: *'i*::*finite* ⇒ *'s*
    and *upper-basis* :: *'j*::*finite* ⇒ *'a*
begin

sublocale *dimensions*: *indexed-field-tower*
rS sA rA lower-basis upper-basis
⟨*proof*⟩

theorem *finite-field-tower-dimension*:
dimensions.direct-space.dim UNIV =
dimensions.lower-space.dim UNIV ∗
dimensions.upper-space.dim UNIV
⟨*proof*⟩

theorem *finite-field-tower-trace*:
tower-algebra.algebra-trace a =
lower-algebra.algebra-trace (*upper-algebra.algebra-trace a*)
⟨*proof*⟩

theorem *finite-field-tower-norm*:
tower-algebra.algebra-norm a =
lower-algebra.algebra-norm (*upper-algebra.algebra-norm a*)
⟨*proof*⟩

end

The preceding locale covers arbitrary nested finite fields without imposing a separability assumption. The next example supplies a concrete nontrivial extension: the four-element field is represented as *GF4 a b*, corresponding to $a + b\alpha$ over $\mathbb{F}_2$, with $\alpha^2 = \alpha + 1$.

10.2 The four-element field over $\mathbb{F}_2$

lemma *gf2-cases* [*case-names zero one*]:
fixes *r* :: *gf2*
obtains *r* = *0* | *r* = *1*
⟨*proof*⟩

lemma *gf2-one-add-one* [*simp*]:
(*1* :: *gf2*) + *1* = *0*
⟨*proof*⟩

lemma *gf2-two* [*simp*]:
(*2* :: *gf2*) = *0*
⟨*proof*⟩

definition *gf2-gf4-embed* :: *gf2* ⇒ *gf4*
where *gf2-gf4-embed r* = (*if r* = *0 then 0 else 1*)

definition *gf2-gf4-scale* :: *gf2* ⇒ *gf4* ⇒ *gf4*
where *gf2-gf4-scale r x* = *gf2-gf4-embed r* ∗ *x*

definition *gf4-self-scale* :: *gf4* ⇒ *gf4* ⇒ *gf4*
  where *gf4-self-scale r x* = *r* ∗ *x*

lemma *gf2-gf4-embed-zero* [*simp*]:
  gf2-gf4-embed 0 = *0*
  ⟨*proof*⟩

lemma *gf2-gf4-embed-one* [*simp*]:
  gf2-gf4-embed 1 = *1*
  ⟨*proof*⟩

lemma *gf2-gf4-embed-add* [*simp*]:
  gf2-gf4-embed (*r* + *s*) =
    gf2-gf4-embed r + *gf2-gf4-embed s*
  ⟨*proof*⟩

lemma *gf2-gf4-embed-mult* [*simp*]:
  gf2-gf4-embed (*r* ∗ *s*) =
    gf2-gf4-embed r ∗ *gf2-gf4-embed s*
  ⟨*proof*⟩

interpretation *gf2-gf4*: *commutative-algebra gf2-gf4-scale*
  ⟨*proof*⟩

interpretation *gf4-self*: *commutative-algebra gf4-self-scale*
  ⟨*proof*⟩

abbreviation *gf4-alpha* :: *gf4*
  where *gf4-alpha* ≡ *GF4 False True*

lemma *gf4-one-ne-alpha* [*simp*]:
  (*1* :: *gf4*) ≠ *gf4-alpha*
  ⟨*proof*⟩

definition *gf4-basis* :: *2* ⇒ *gf4*
  where *gf4-basis i* = (*if i* = *1 then 1 else gf4-alpha*)

definition *gf4-unit-basis* :: *1* ⇒ *gf4*
  where *gf4-unit-basis i* = *1*

definition *gf4-tower-basis* :: (*2* × *1*) ⇒ *gf4*
  where *gf4-tower-basis ij* =
    gf4-self-scale (*gf4-basis* (*fst ij*))
      (*gf4-unit-basis* (*snd ij*))

lemma *gf4-basis-range*:
  range gf4-basis = {*1*, *gf4-alpha*}
⟨*proof*⟩

lemma *gf4-basis-inj*:
inj gf4-basis
⟨*proof*⟩

lemma *gf2-gf4-scale-one-zero-iff* [*simp*]:
gf2-gf4-scale r 1 = 0 ⟷ r = 0
⟨*proof*⟩

lemma *gf2-gf4-scale-alpha-zero-iff* [*simp*]:
gf2-gf4-scale r gf4-alpha = 0 ⟷ r = 0
⟨*proof*⟩

lemma *gf4-linear-combination-zero-iff* [*simp*]:
gf2-gf4-scale r 1 +
gf2-gf4-scale s gf4-alpha = 0 ⟷
r = 0 ∧ s = 0
⟨*proof*⟩

lemma *gf4-basis-independent*:
gf2-gf4.independent (*range gf4-basis*)
⟨*proof*⟩

lemma *gf4-basis-span*:
gf2-gf4.span (*range gf4-basis*) = *UNIV*
⟨*proof*⟩

lemma *gf4-unit-basis-inj*:
inj gf4-unit-basis
⟨*proof*⟩

lemma *gf4-unit-basis-range*:
range gf4-unit-basis = {*1*}
⟨*proof*⟩

lemma *gf4-unit-basis-independent*:
gf4-self.independent (*range gf4-unit-basis*)
⟨*proof*⟩

lemma *gf4-unit-basis-span*:
gf4-self.span (*range gf4-unit-basis*) = *UNIV*
⟨*proof*⟩

lemma *gf4-unit-basis-independent-single*:
gf4-self.independent {*gf4-unit-basis* (*1* :: *1*)}
⟨*proof*⟩

lemma *gf4-unit-basis-span-single*:
gf4-self.span {*gf4-unit-basis* (*1* :: *1*)} = *UNIV*
⟨*proof*⟩

interpretation *gf4-over-gf2*:
  indexed-algebra gf2-gf4-scale gf4-basis
⟨*proof*⟩

interpretation *gf4-over-itself*:
  indexed-algebra gf4-self-scale gf4-unit-basis
⟨*proof*⟩

interpretation *gf4-over-gf2-space*: *vector-space gf2-gf4-scale*
⟨*proof*⟩

definition *gf2-of-bool* :: *bool* ⇒ *gf2*
  where *gf2-of-bool b* = (*if b then 1 else 0*)

The Boolean coordinates of an element determine its coefficients in this basis of the four-element field over the two-element field.

lemma *gf4-expansion-coefficients*:
  gf2-gf4-scale r 1 +
    gf2-gf4-scale s gf4-alpha = *GF4 a b* ⟷
   r = *gf2-of-bool a* ∧ *s* = *gf2-of-bool b*
⟨*proof*⟩

lemma *gf4-coordinate-one* [*simp*]:
  gf4-over-gf2.coordinates.coordinate (*GF4 a b*) (*1* :: *2*) =
    gf2-of-bool a
⟨*proof*⟩

lemma *gf4-coordinate-two* [*simp*]:
  gf4-over-gf2.coordinates.coordinate (*GF4 a b*) (*2* :: *2*) =
    gf2-of-bool b
⟨*proof*⟩

lemma *gf4-regular-matrix-11* [*simp*]:
  gf4-over-gf2.regular-matrix (*GF4 a b*) $
    (*1* :: *2*) $ (*1* :: *2*) = *gf2-of-bool a*
⟨*proof*⟩

lemma *gf4-regular-matrix-12* [*simp*]:
  gf4-over-gf2.regular-matrix (*GF4 a b*) $
    (*1* :: *2*) $ (*2* :: *2*) = *gf2-of-bool b*
⟨*proof*⟩

lemma *gf4-regular-matrix-21* [*simp*]:
  gf4-over-gf2.regular-matrix (*GF4 a b*) $
    (*2* :: *2*) $ (*1* :: *2*) = *gf2-of-bool b*
⟨*proof*⟩

lemma *gf4-regular-matrix-22* [*simp*]:

gf4-over-gf2.regular-matrix (*GF4 a b*) $
    (*2* :: *2*) $ (*2* :: *2*) = *gf2-of-bool* (*a* ≠ *b*)
⟨*proof*⟩

lemma *gf4-algebra-trace*:
gf4-over-gf2.algebra-trace (*GF4 a b*) = *gf2-of-bool b*
⟨*proof*⟩

lemma *gf4-algebra-norm*:
gf4-over-gf2.algebra-norm (*GF4 a b*) =
  gf2-of-bool (*a* ∨ *b*)
⟨*proof*⟩

theorem *gf4-over-gf2-trace-norm*:
gf4-over-gf2.algebra-trace gf4-alpha = *1* ∧
  gf4-over-gf2.algebra-norm gf4-alpha = *1*
⟨*proof*⟩

theorem *gf4-over-gf2-dimension*:
gf4-over-gf2-space.dim UNIV = *2*
⟨*proof*⟩

theorem *concrete-finite-field-tower-laws*:
indexed-algebra.algebra-trace
    gf2-gf4-scale gf4-tower-basis x =
  gf4-over-gf2.algebra-trace
    (*gf4-over-itself.algebra-trace x*) ∧
 indexed-algebra.algebra-norm
    gf2-gf4-scale gf4-tower-basis x =
  gf4-over-gf2.algebra-norm
    (*gf4-over-itself.algebra-norm x*)
⟨*proof*⟩

10.3 The quadratic extension of the reals by the imaginary unit

definition *real-complex-scale* :: *real* ⇒ *complex* ⇒ *complex*
  where *real-complex-scale r z* = *of-real r* ∗ *z*

definition *real-complex-direct-scale* :: *real* ⇒ *complex* ⇒ *complex*
  where *real-complex-direct-scale r z* = *of-real r* ∗ *z*

definition *complex-complex-scale* :: *complex* ⇒ *complex* ⇒ *complex*
  where *complex-complex-scale z w* = *z* ∗ *w*

definition *complex-basis* :: *2* ⇒ *complex*
  where *complex-basis i* = (*if i* = *1 then 1 else* i)

definition *complex-unit-basis* :: *1* ⇒ *complex*

where *complex-unit-basis i = 1*

definition *complex-tower-basis* :: *(2 × 1) ⇒ complex*
where *complex-tower-basis ij =*
complex-complex-scale (*complex-basis* (*fst ij*))
(*complex-unit-basis* (*snd ij*))

interpretation *real-complex*: *commutative-algebra real-complex-scale*
⟨*proof*⟩

interpretation *complex-complex*: *commutative-algebra complex-complex-scale*
⟨*proof*⟩

lemma *complex-basis-range*:
range complex-basis = {*1*, i}
⟨*proof*⟩

lemma *complex-basis-inj*:
inj complex-basis
⟨*proof*⟩

lemma *complex-basis-independent*:
real-complex.independent (*range complex-basis*)
⟨*proof*⟩

lemma *complex-basis-span*:
real-complex.span (*range complex-basis*) = *UNIV*
⟨*proof*⟩

lemma *complex-unit-basis-inj*:
inj complex-unit-basis
⟨*proof*⟩

lemma *complex-unit-basis-range*:
range complex-unit-basis = {*1*}
⟨*proof*⟩

lemma *complex-unit-basis-independent*:
complex-complex.independent (*range complex-unit-basis*)
⟨*proof*⟩

lemma *complex-unit-basis-span*:
complex-complex.span (*range complex-unit-basis*) = *UNIV*
⟨*proof*⟩

lemma *complex-unit-basis-independent-single*:
complex-complex.independent {*complex-unit-basis* (*1* :: *1*)}
⟨*proof*⟩

lemma *complex-unit-basis-span-single*:
complex-complex.span {*complex-unit-basis* (*1* :: *1*)} = *UNIV*
⟨*proof*⟩

interpretation *complex-real-algebra*:
indexed-algebra real-complex-scale complex-basis
⟨*proof*⟩

interpretation *complex-scalar-algebra*:
indexed-algebra complex-complex-scale complex-unit-basis
⟨*proof*⟩

lemma *complex-coordinate-one* [*simp*]:
complex-real-algebra.coordinates.coordinate z (*1* :: *2*) = *Re z*
⟨*proof*⟩

lemma *complex-coordinate-two* [*simp*]:
complex-real-algebra.coordinates.coordinate z (*2* :: *2*) = *Im z*
⟨*proof*⟩

lemma *complex-regular-matrix-11* [*simp*]:
complex-real-algebra.regular-matrix z $ (*1* :: *2*) $ (*1* :: *2*) =
Re z
⟨*proof*⟩

lemma *complex-regular-matrix-12* [*simp*]:
complex-real-algebra.regular-matrix z $ (*1* :: *2*) $ (*2* :: *2*) =
− *Im z*
⟨*proof*⟩

lemma *complex-regular-matrix-21* [*simp*]:
complex-real-algebra.regular-matrix z $ (*2* :: *2*) $ (*1* :: *2*) =
Im z
⟨*proof*⟩

lemma *complex-regular-matrix-22* [*simp*]:
complex-real-algebra.regular-matrix z $ (*2* :: *2*) $ (*2* :: *2*) =
Re z
⟨*proof*⟩

lemma *complex-algebra-trace*:
complex-real-algebra.algebra-trace z = *2* ∗ *Re z*
⟨*proof*⟩

lemma *complex-algebra-norm*:
complex-real-algebra.algebra-norm z =
(*Re z*)^2 + (*Im z*)^2
⟨*proof*⟩

theorem *quadratic-extension-trace-norm*:
  complex-real-algebra.algebra-trace $z = 2 * Re\ z \wedge$
    complex-real-algebra.algebra-norm $z =$
      $(Re\ z)^2 + (Im\ z)^2$
  ⟨*proof*⟩

theorem *complex-quadratic-tower-laws*:
  indexed-algebra.algebra-trace
      real-complex-direct-scale complex-tower-basis $z =$
    complex-real-algebra.algebra-trace
      (*complex-scalar-algebra.algebra-trace* z) $\wedge$
   indexed-algebra.algebra-norm
      real-complex-direct-scale complex-tower-basis $z =$
    complex-real-algebra.algebra-norm
      (*complex-scalar-algebra.algebra-norm* z)
⟨*proof*⟩

end

Acknowledgments

AI assistance was used for proof engineering.

References

[1] S. Lang. *Algebra*. Springer, New York, third edition, 2002. DOI: `https://doi.org/10.1007/978-1-4613-0041-0`.