

Representations of Finite Groups

Jeremy Sylvestre
University of Alberta, Augustana Campus
jeremy.sylvestre@ualberta.ca

May 12, 2022

Abstract

We provide a formal framework for the theory of representations of finite groups, as modules over the group ring. Along the way, we develop the general theory of groups (relying on the *group_add* class for the basics), modules, and vector spaces, to the extent required for theory of group representations. We then provide formal proofs of several important introductory theorems in the subject, including Maschke's theorem, Schur's lemma, and Frobenius reciprocity. We also prove that every irreducible representation is isomorphic to a submodule of the group ring, leading to the fact that for a finite group there are only finitely many isomorphism classes of irreducible representations. In all of this, no restriction is made on the characteristic of the ring or field of scalars until the definition of a group representation, and then the only restriction made is that the characteristic must not divide the order of the group.

Contents

1 Preliminaries	5
1.1 Logic	5
1.2 Sets	5
1.3 Lists	5
1.3.1 <i>zip</i>	5
1.3.2 <i>concat</i>	6
1.3.3 <i>strip-while</i>	6
1.3.4 <i>sum-list</i>	6
1.3.5 <i>listset</i>	7
1.4 Functions	8
1.4.1 Miscellaneous facts	8
1.4.2 Support of a function	8
1.4.3 Convolution	9
1.5 Almost-everywhere-zero functions	10

1.5.1	Definition and basic properties	10
1.5.2	Delta (impulse) functions	10
1.5.3	Convolution of almost-everywhere-zero functions	11
1.5.4	Type definition, instantiations, and instances	12
1.5.5	Transfer facts	15
1.5.6	Almost-everywhere-zero functions with constrained support	16
1.6	Polynomials	18
1.7	Algebra of sets	19
1.7.1	General facts	19
1.7.2	Additive independence of sets	19
1.7.3	Inner direct sums	20
2	Groups	21
2.1	Locales and basic facts	21
2.1.1	Locale <i>Group</i> and finite variant <i>FinGroup</i>	21
2.1.2	Abelian variant locale <i>AbGroup</i>	23
2.2	Right cosets	23
2.3	Group homomorphisms	25
2.3.1	Preliminaries	25
2.3.2	Locales	25
2.3.3	Basic facts	26
2.3.4	Basic facts about endomorphisms	27
2.3.5	Basic facts about isomorphisms	28
2.3.6	Hom-sets	28
2.4	Facts about collections of groups	29
2.5	Inner direct sums of Abelian groups	30
2.5.1	General facts	30
2.5.2	Element decomposition and projection	30
2.6	Rings	32
2.6.1	Preliminaries	32
2.6.2	Locale and basic facts	32
2.7	The group ring	33
2.7.1	Definition and basic facts	33
2.7.2	Projecting almost-everywhere-zero functions onto a group ring	34
3	Modules	35
3.1	Locales and basic facts	35
3.1.1	Locales	35
3.1.2	Basic facts	36
3.1.3	Module and submodule instances	38
3.2	Linear algebra in modules	39
3.2.1	Linear combinations: <i>lincomb</i>	39

3.2.2	Spanning: $R\text{Span}$ and Span	41
3.2.3	Finitely generated modules	44
3.2.4	R -linear independence	45
3.2.5	Linear independence over $UNIV$	47
3.2.6	Rank	48
3.3	Module homomorphisms	49
3.3.1	Locales	49
3.3.2	Basic facts	50
3.3.3	Basic facts about endomorphisms	51
3.3.4	Basic facts about isomorphisms	52
3.4	Inner direct sums of RModules	52
4	Vector Spaces	52
4.1	Locales and basic facts	52
4.2	Linear algebra in vector spaces	54
4.2.1	Linear independence and spanning	54
4.2.2	Basis for a vector space: <i>basis-for</i>	55
4.3	Finite dimensional spaces	56
4.4	Vector space homomorphisms	57
4.4.1	Locales	57
4.4.2	Basic facts	58
4.4.3	Hom-sets	59
4.4.4	Basic facts about endomorphisms	60
4.4.5	Polynomials of endomorphisms	63
4.4.6	Existence of eigenvectors of endomorphisms of finite-dimensional vector spaces	64
5	Modules Over a Group Ring	64
5.1	Almost-everywhere-zero functions as scalars	64
5.2	Locale and basic facts	65
5.3	Modules over a group ring as a vector spaces	68
5.4	Homomorphisms of modules over a group ring	69
5.4.1	Locales	69
5.4.2	Basic facts	69
5.4.3	Basic facts about endomorphisms	72
5.4.4	Basic facts about isomorphisms	72
5.4.5	Hom-sets	74
5.5	Induced modules	75
5.5.1	Additive function spaces	75
5.5.2	Spaces of functions which transform under scalar multiplication by almost-everywhere-zero functions	76
5.5.3	General induced spaces of functions on a group ring	76
5.5.4	The right regular action	78
5.5.5	Locale and basic facts	78

6	Representations of Finite Groups	81
6.1	Locale and basic facts	81
6.2	Irreducible representations	82
6.3	Maschke's theorem	83
6.3.1	Averaged projection onto a G -subspace	83
6.3.2	The theorem	84
6.3.3	Consequence: complete reducibility	84
6.3.4	Consequence: decomposition relative to a homomorphism	85
6.4	Schur's lemma	85
6.5	The group ring as a representation space	85
6.5.1	The group ring is a representation space	85
6.5.2	Irreducible representations are constituents of the group ring	86
6.6	Isomorphism classes of irreducible representations	86
6.7	Induced representations	88
6.7.1	Locale and basic facts	88
6.7.2	A basis for the induced space	89
6.7.3	The induced space is a representation space	90
6.8	Frobenius reciprocity	91
6.8.1	Locale and basic facts	91
6.8.2	The required isomorphism of Hom-sets	92
6.8.3	The inverse map of Hom-sets	93
6.8.4	Demonstration of bijectivity	94
6.8.5	The theorem	94
7	Bibliography	95

Note: A number of the proofs in this theory were modelled on or inspired by proofs in the books listed in the bibliography.

theory *Rep-Fin-Groups*

imports

HOL-Library.Function-Algebras

HOL-Library.Set-Algebras

HOL-Computational-Algebra.Polynomial

begin

1 Preliminaries

In this section, we establish some basic facts about logic, sets, and functions that are not available in the HOL library. As well, we develop some theory for almost-everywhere-zero functions in preparation of the definition of the group ring.

1.1 Logic

lemma *conjcases* [*case-names BothTrue OneTrue OtherTrue BothFalse*] :
 assumes *BothTrue*: $P \wedge Q \implies R$
 and *OneTrue*: $P \wedge \neg Q \implies R$
 and *OtherTrue*: $\neg P \wedge Q \implies R$
 and *BothFalse*: $\neg P \wedge \neg Q \implies R$
 shows R
 \langle *proof* \rangle

1.2 Sets

lemma *empty-set-diff-single* : $A - \{x\} = \{\} \implies A = \{\} \vee A = \{x\}$
 \langle *proof* \rangle

lemma *seteqI* : $(\bigwedge a. a \in A \implies a \in B) \implies (\bigwedge b. b \in B \implies b \in A) \implies A = B$
 \langle *proof* \rangle

lemma *prod-ballI* : $(\bigwedge a b. (a,b) \in A \times B \implies P a b) \implies \forall (a,b) \in A \times B. P a b$
 \langle *proof* \rangle

lemma *good-card-imp-finite* : $of_nat (card A) \neq (0::'a::semiring-1) \implies finite A$
 \langle *proof* \rangle

1.3 Lists

1.3.1 zip

lemma *zip-truncate-left* : $zip\ xs\ ys = zip\ (take\ (length\ ys)\ xs)\ ys$
 \langle *proof* \rangle

lemma *zip-truncate-right* : $zip\ xs\ ys = zip\ xs\ (take\ (length\ xs)\ ys)$
 \langle *proof* \rangle

Lemmas *zip-append1* and *zip-append2* in theory *HOL.List* have unnecessary *take (length -)* in them. Here are replacements.

lemma *zip-append-left* :
 $zip\ (xs@ys)\ zs = zip\ xs\ zs @ zip\ ys\ (drop\ (length\ xs)\ zs)$
 \langle *proof* \rangle

lemma *zip-append-right* :
 $zip\ xs\ (ys@zs) = zip\ xs\ ys @ zip\ (drop\ (length\ ys)\ xs)\ zs$

<proof>

lemma *length-concat-map-split-zip* :

$$\text{length } [f \ x \ y. (x,y) \leftarrow \text{zip } xs \ ys] = \min (\text{length } xs) (\text{length } ys)$$

<proof>

lemma *concat-map-split-eq-map-split-zip* :

$$[f \ x \ y. (x,y) \leftarrow \text{zip } xs \ ys] = \text{map } (\text{case-prod } f) (\text{zip } xs \ ys)$$

<proof>

lemma *set-zip-map2* :

$$(a,z) \in \text{set } (\text{zip } xs \ (\text{map } f \ ys)) \implies \exists b. (a,b) \in \text{set } (\text{zip } xs \ ys) \wedge z = f \ b$$

<proof>

1.3.2 concat

lemma *concat-eq* :

$$\text{list-all2 } (\lambda xs \ ys. \text{length } xs = \text{length } ys) \ xss \ yss \implies \text{concat } xss = \text{concat } yss \\ \implies xss = yss$$

<proof>

lemma *match-concat* :

fixes *bss* :: 'b list list

defines *eq-len*: $\text{eq-len} \equiv \lambda xs \ ys. \text{length } xs = \text{length } ys$

shows $\forall as :: 'a \text{ list}. \text{length } as = \text{length } (\text{concat } bss)$

$$\longrightarrow (\exists css :: 'a \text{ list list}. as = \text{concat } css \wedge \text{list-all2 } \text{eq-len } css \ bss)$$

<proof>

1.3.3 strip-while

lemma *strip-while-0-nnil* :

$$as \neq [] \implies \text{set } as \neq 0 \implies \text{strip-while } ((=) \ 0) \ as \neq []$$

<proof>

1.3.4 sum-list

lemma *const-sum-list* :

$$\forall x \in \text{set } xs. f \ x = a \implies \text{sum-list } (\text{map } f \ xs) = a * (\text{length } xs)$$

<proof>

lemma *sum-list-prod-cong* :

$$\forall (x,y) \in \text{set } xys. f \ x \ y = g \ x \ y$$

$$\implies (\sum (x,y) \leftarrow xys. f \ x \ y) = (\sum (x,y) \leftarrow xys. g \ x \ y)$$

<proof>

lemma *sum-list-prod-map2* :

$$(\sum (a,y) \leftarrow \text{zip } as \ (\text{map } f \ bs). g \ a \ y) = (\sum (a,b) \leftarrow \text{zip } as \ bs. g \ a \ (f \ b))$$

<proof>

lemma *sum-list-fun-apply* : $(\sum x \leftarrow xs. f \ x) \ y = (\sum x \leftarrow xs. f \ x \ y)$

<proof>

lemma *sum-list-prod-fun-apply* : $(\sum (x,y) \leftarrow xys. f x y) z = (\sum (x,y) \leftarrow xys. f x y z)$
<proof>

lemma (*in comm-monoid-add*) *sum-list-plus* :
 $length\ xs = length\ ys$
 $\implies sum-list\ xs + sum-list\ ys = sum-list\ [a+b. (a,b) \leftarrow zip\ xs\ ys]$
<proof>

lemma *sum-list-const-mult-prod* :
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'r :: semiring-0$
shows $r * (\sum (x,y) \leftarrow xys. f x y) = (\sum (x,y) \leftarrow xys. r * (f x y))$
<proof>

lemma *sum-list-mult-const-prod* :
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'r :: semiring-0$
shows $(\sum (x,y) \leftarrow xys. f x y) * r = (\sum (x,y) \leftarrow xys. (f x y) * r)$
<proof>

lemma *sum-list-update* :
fixes $xs :: 'a :: ab-group-add\ list$
shows $n < length\ xs \implies sum-list\ (xs[n := y]) = sum-list\ xs - xs!n + y$
<proof>

lemma *sum-list-replicate0* : $sum-list\ (replicate\ n\ 0) = 0$
<proof>

1.3.5 listset

lemma *listset-ConsI* : $x \in X \implies xs \in listset\ Xs \implies x\#\!xs \in listset\ (X\#\!Xs)$
<proof>

lemma *listset-ConsD* : $x\#\!xs \in listset\ (A\ \#\!As) \implies x \in A \wedge xs \in listset\ As$
<proof>

lemma *listset-Cons-conv* :
 $xs \in listset\ (A\ \#\!As) \implies (\exists y\ ys. y \in A \wedge ys \in listset\ As \wedge xs = y\#\!ys)$
<proof>

lemma *listset-length* : $xs \in listset\ Xs \implies length\ xs = length\ Xs$
<proof>

lemma *set-sum-list-element* :
 $x \in (\sum A \leftarrow As. A) \implies \exists as \in listset\ As. x = (\sum a \leftarrow as. a)$
<proof>

lemma *set-sum-list-element-Cons* :
assumes $x \in (\sum X \leftarrow (A\ \#\!As). X)$

shows $\exists a \text{ as. } a \in A \wedge \text{as} \in \text{listset } As \wedge x = a + (\sum b \leftarrow \text{as. } b)$
 ⟨proof⟩

lemma *sum-list-listset* : $\text{as} \in \text{listset } As \implies \text{sum-list as} \in (\sum A \leftarrow As. A)$
 ⟨proof⟩

lemma *listsetI-nth* :
 $\text{length xs} = \text{length } Xs \implies \forall n < \text{length } xs. \text{xs}!n \in Xs!n \implies \text{xs} \in \text{listset } Xs$
 ⟨proof⟩

lemma *listsetD-nth* : $\text{xs} \in \text{listset } Xs \implies \forall n < \text{length } xs. \text{xs}!n \in Xs!n$
 ⟨proof⟩

lemma *set-listset-el-subset* :
 $\text{xs} \in \text{listset } Xs \implies \forall X \in \text{set } Xs. X \subseteq A \implies \text{set xs} \subseteq A$
 ⟨proof⟩

1.4 Functions

1.4.1 Miscellaneous facts

lemma *sum-fun-apply* : $\text{finite } A \implies (\sum a \in A. f a) x = (\sum a \in A. f a x)$
 ⟨proof⟩

lemma *sum-single-nonzero* :
 $\text{finite } A \implies (\forall x \in A. \forall y \in A. f x y = (\text{if } y = x \text{ then } g x \text{ else } 0))$
 $\implies (\forall x \in A. \text{sum } (f x) A = g x)$
 ⟨proof⟩

lemma *distrib-comp-sum-right* : $(T + T') \circ S = (T \circ S) + (T' \circ S)$
 ⟨proof⟩

1.4.2 Support of a function

definition *supp* :: $('a \Rightarrow 'b :: \text{zero}) \Rightarrow 'a \text{ set where } \text{supp } f = \{x. f x \neq 0\}$

lemma *suppI*: $f x \neq 0 \implies x \in \text{supp } f$
 ⟨proof⟩

lemma *suppI-contr*: $x \notin \text{supp } f \implies f x = 0$
 ⟨proof⟩

lemma *suppD*: $x \in \text{supp } f \implies f x \neq 0$
 ⟨proof⟩

lemma *suppD-contr*: $f x = 0 \implies x \notin \text{supp } f$
 ⟨proof⟩

lemma *zerofun-imp-empty-supp* : $\text{supp } 0 = \{\}$
 ⟨proof⟩

lemma *supp-zerofun-subset-any* : $\text{supp } 0 \subseteq A$
 ⟨*proof*⟩

lemma *supp-sum-subset-union-supp* :
fixes $f g :: 'a \Rightarrow 'b::\text{monoid-add}$
shows $\text{supp } (f + g) \subseteq \text{supp } f \cup \text{supp } g$
 ⟨*proof*⟩

lemma *supp-neg-eq-supp* :
fixes $f :: 'a \Rightarrow 'b::\text{group-add}$
shows $\text{supp } (-f) = \text{supp } f$
 ⟨*proof*⟩

lemma *supp-diff-subset-union-supp* :
fixes $f g :: 'a \Rightarrow 'b::\text{group-add}$
shows $\text{supp } (f - g) \subseteq \text{supp } f \cup \text{supp } g$
 ⟨*proof*⟩

abbreviation *restrict0* :: $('a \Rightarrow 'b::\text{zero}) \Rightarrow 'a \text{ set} \Rightarrow ('a \Rightarrow 'b)$ (**infix** \downarrow 70)
where $\text{restrict0 } f A \equiv (\lambda a. \text{if } a \in A \text{ then } f a \text{ else } 0)$

lemma *supp-restrict0* : $\text{supp } (f \downarrow A) \subseteq A$
 ⟨*proof*⟩

lemma *bij-betw-restrict0* : $\text{bij-betw } f A B \Longrightarrow \text{bij-betw } (f \downarrow A) A B$
 ⟨*proof*⟩

1.4.3 Convolution

definition *convolution* ::
 $('a::\text{group-add} \Rightarrow 'b::\{\text{comm-monoid-add}, \text{times}\}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b)$
where $\text{convolution } f g$
 $= (\lambda x. \sum y | x - y \in \text{supp } f \wedge y \in \text{supp } g. (f (x - y)) * g y)$

— More often than not, this definition will be used in the case that $'b$ is of class *mult-zero*, in which case the conditions $x - y \in \text{supp } f$ and $y \in \text{supp } g$ are obviously mathematically unnecessary. However, they also serve to ensure that the sum is taken over a finite set in the case that at least one of f and g is almost everywhere zero.

lemma *convolution-zero* :
fixes $f g :: 'a::\text{group-add} \Rightarrow 'b::\{\text{comm-monoid-add}, \text{mult-zero}\}$
shows $f = 0 \vee g = 0 \Longrightarrow \text{convolution } f g = 0$
 ⟨*proof*⟩

lemma *convolution-symm* :
fixes $f g :: 'a::\text{group-add} \Rightarrow 'b::\{\text{comm-monoid-add}, \text{times}\}$
shows $\text{convolution } f g$
 $= (\lambda x. \sum y | y \in \text{supp } f \wedge -y + x \in \text{supp } g. (f y) * g (-y + x))$

<proof>

lemma *supp-convolution-subset-sum-supp* :
 fixes $f g :: 'a::group-add \Rightarrow 'b::\{comm-monoid-add,times\}$
 shows $supp (convolution f g) \subseteq supp f + supp g$
<proof>

1.5 Almost-everywhere-zero functions

1.5.1 Definition and basic properties

definition *aezfun-set* = $\{f :: 'a \Rightarrow 'b::zero. finite (supp f)\}$

lemma *aezfun-setD*: $f \in aezfun-set \Longrightarrow finite (supp f)$
<proof>

lemma *aezfun-setI*: $finite (supp f) \Longrightarrow f \in aezfun-set$
<proof>

lemma *zerofun-is-aezfun* : $0 \in aezfun-set$
<proof>

lemma *sum-of-aezfun-is-aezfun* :
 fixes $f g :: 'a \Rightarrow 'b::monoid-add$
 shows $f \in aezfun-set \Longrightarrow g \in aezfun-set \Longrightarrow f + g \in aezfun-set$
<proof>

lemma *neg-of-aezfun-is-aezfun* :
 fixes $f :: 'a \Rightarrow 'b::group-add$
 shows $f \in aezfun-set \Longrightarrow -f \in aezfun-set$
<proof>

lemma *diff-of-aezfun-is-aezfun* :
 fixes $f g :: 'a \Rightarrow 'b::group-add$
 shows $f \in aezfun-set \Longrightarrow g \in aezfun-set \Longrightarrow f - g \in aezfun-set$
<proof>

lemma *restrict-and-extend0-aezfun-is-aezfun* :
 assumes $f \in aezfun-set$
 shows $f \downarrow A \in aezfun-set$
<proof>

1.5.2 Delta (impulse) functions

The notation is set up in the order output-input so that later when these are used to define the group ring RG, it will be in order ring-element-group-element.

definition *deltafun* :: $'b::zero \Rightarrow 'a \Rightarrow ('a \Rightarrow 'b)$ (**infix** δ 70)
 where $b \delta a = (\lambda x. if x = a then b else 0)$

lemma *deltafun-apply-eq* : $(b \delta a) a = b$
 ⟨proof⟩

lemma *deltafun-apply-neq* : $x \neq a \implies (b \delta a) x = 0$
 ⟨proof⟩

lemma *deltafun0* : $0 \delta a = 0$
 ⟨proof⟩

lemma *deltafun-plus* :
fixes $b\ c :: 'b::\text{monoid-add}$
shows $(b+c) \delta a = (b \delta a) + (c \delta a)$
 ⟨proof⟩

lemma *supp-delta0fun* : $\text{supp } (0 \delta a) = \{\}$
 ⟨proof⟩

lemma *supp-deltafun* : $b \neq 0 \implies \text{supp } (b \delta a) = \{a\}$
 ⟨proof⟩

lemma *deltafun-is-aezfun* : $b \delta a \in \text{aezfun-set}$
 ⟨proof⟩

lemma *aezfun-common-supp-spanning-set'* :
 $\text{finite } A \implies \exists \text{ as. distinct as } \wedge \text{ set as } = A$
 $\wedge (\forall f :: 'a \Rightarrow 'b::\text{semiring-1. supp } f \subseteq A$
 $\longrightarrow (\exists \text{ bs. length bs } = \text{ length as } \wedge f = (\sum (b,a) \leftarrow \text{zip bs as. } b \delta a)))$
 ⟨proof⟩

1.5.3 Convolution of almost-everywhere-zero functions

lemma *convolution-eq-sum-over-supp-right* :
fixes $g\ f :: 'a::\text{group-add} \Rightarrow 'b::\{\text{comm-monoid-add,mult-zero}\}$
assumes $g \in \text{aezfun-set}$
shows $\text{convolution } f\ g = (\lambda x. \sum_{y \in \text{supp } g. (f (x - y)) * g\ y)$
 ⟨proof⟩

lemma *convolution-symm-eq-sum-over-supp-left* :
fixes $f\ g :: 'a::\text{group-add} \Rightarrow 'b::\{\text{comm-monoid-add,mult-zero}\}$
assumes $f \in \text{aezfun-set}$
shows $\text{convolution } f\ g = (\lambda x. \sum_{y \in \text{supp } f. (f\ y) * g (-y + x))$
 ⟨proof⟩

lemma *convolution-delta-left* :
fixes $b :: 'b::\{\text{comm-monoid-add,mult-zero}\}$
and $a :: 'a::\text{group-add}$
and $f :: 'a \Rightarrow 'b$
shows $\text{convolution } (b \delta a)\ f = (\lambda x. b * f (-a + x))$

<proof>

lemma *convolution-delta-right* :

fixes $b :: 'b::\{\text{comm-monoid-add,mult-zero}\}$
and $f :: 'a::\text{group-add} \Rightarrow 'b$ **and** $a::'a$
shows $\text{convolution } f (b \delta a) = (\lambda x. f (x - a) * b)$
<proof>

lemma *convolution-delta-delta* :

fixes $b1\ b2 :: 'b::\{\text{comm-monoid-add,mult-zero}\}$
and $a1\ a2 :: 'a::\text{group-add}$
shows $\text{convolution } (b1 \delta a1) (b2 \delta a2) = (b1 * b2) \delta (a1 + a2)$
<proof>

lemma *convolution-of-aezfun-is-aezfun* :

fixes $f\ g :: 'a::\text{group-add} \Rightarrow 'b::\{\text{comm-monoid-add,times}\}$
shows $f \in \text{aezfun-set} \Longrightarrow g \in \text{aezfun-set} \Longrightarrow \text{convolution } f\ g \in \text{aezfun-set}$
<proof>

lemma *convolution-assoc* :

fixes $f\ h\ g :: 'a::\text{group-add} \Rightarrow 'b::\text{semiring-0}$
assumes $f\text{-aez} : f \in \text{aezfun-set}$ **and** $h\text{-aez} : h \in \text{aezfun-set}$
shows $\text{convolution } (\text{convolution } f\ g) h = \text{convolution } f (\text{convolution } g\ h)$
<proof>

lemma *convolution-distrib-left* :

fixes $g\ h\ f :: 'a::\text{group-add} \Rightarrow 'b::\text{semiring-0}$
assumes $g \in \text{aezfun-set}$ $h \in \text{aezfun-set}$
shows $\text{convolution } f (g + h) = \text{convolution } f\ g + \text{convolution } f\ h$
<proof>

lemma *convolution-distrib-right* :

fixes $f\ g\ h :: 'a::\text{group-add} \Rightarrow 'b::\text{semiring-0}$
assumes $f \in \text{aezfun-set}$ $g \in \text{aezfun-set}$
shows $\text{convolution } (f + g) h = \text{convolution } f\ h + \text{convolution } g\ h$
<proof>

1.5.4 Type definition, instantiations, and instances

typedef (overloaded) ($'a::\text{zero},'b$) *aezfun* = *aezfun-set* :: ($'b \Rightarrow 'a$) *set*
morphisms *aezfun* *Abs-aezfun*
<proof>

setup-lifting *type-definition-aezfun*

lemma *aezfun-finite-supp* : *finite* (*supp* (*aezfun* *a*))
<proof>

lemma *aezfun-transfer* : *aezfun* *a* = *aezfun* *b* \Longrightarrow *a* = *b* *<proof>*

```

instantiation aezfun :: (zero, type) zero
begin
  lift-definition zero-aezfun :: ('a, 'b) aezfun is 0::'b⇒'a
    ⟨proof⟩
  instance ⟨proof⟩
end

lemma zero-aezfun-transfer : Abs-aezfun ((0::'b::zero) δ (0::'a::zero)) = 0
  ⟨proof⟩

lemma zero-aezfun-apply [simp]: aezfun 0 x = 0
  ⟨proof⟩

instantiation aezfun :: (monoid-add, type) plus
begin
  lift-definition plus-aezfun :: ('a, 'b) aezfun ⇒ ('a, 'b) aezfun ⇒ ('a, 'b) aezfun
    is λf g. f + g
    ⟨proof⟩
  instance ⟨proof⟩
end

lemma plus-aezfun-apply [simp]: aezfun (a+b) x = aezfun a x + aezfun b x
  ⟨proof⟩

instance aezfun :: (monoid-add, type) semigroup-add
  ⟨proof⟩

instance aezfun :: (monoid-add, type) monoid-add
  ⟨proof⟩

lemma sum-list-aezfun-apply [simp] :
  aezfun (sum-list as) x = (∑ a←as. aezfun a x)
  ⟨proof⟩

lemma sum-list-map-aezfun-apply [simp] :
  aezfun (∑ a←as. f a) x = (∑ a←as. aezfun (f a) x)
  ⟨proof⟩

lemma sum-list-map-aezfun [simp] :
  aezfun (∑ a←as. f a) = (∑ a←as. aezfun (f a))
  ⟨proof⟩

lemma sum-list-prod-map-aezfun-apply :
  aezfun (∑ (x,y)←xys. f x y) a = (∑ (x,y)←xys. aezfun (f x y) a)
  ⟨proof⟩

lemma sum-list-prod-map-aezfun :
  aezfun (∑ (x,y)←xys. f x y) = (∑ (x,y)←xys. aezfun (f x y))

```

$\langle proof \rangle$

instance *aezfun* :: (*comm-monoid-add*, *type*) *comm-monoid-add*
 $\langle proof \rangle$

lemma *sum-aezfun-apply* [*simp*] :
 $finite\ A \implies aezfun\ (\sum A)\ x = (\sum a \in A.\ aezfun\ a\ x)$
 $\langle proof \rangle$

instantiation *aezfun* :: (*group-add*, *type*) *minus*
begin
lift-definition *minus-aezfun* :: ('a, 'b) *aezfun* \Rightarrow ('a, 'b) *aezfun* \Rightarrow ('a, 'b) *aezfun*
is $\lambda f\ g.\ f - g$
 $\langle proof \rangle$
instance $\langle proof \rangle$
end

lemma *minus-aezfun-apply* [*simp*]: *aezfun* (*a* - *b*) *x* = *aezfun* *a* *x* - *aezfun* *b* *x*
 $\langle proof \rangle$

instantiation *aezfun* :: (*group-add*, *type*) *uminus*
begin
lift-definition *uminus-aezfun* :: ('a, 'b) *aezfun* \Rightarrow ('a, 'b) *aezfun* **is** $\lambda f.\ - f$
 $\langle proof \rangle$
instance $\langle proof \rangle$
end

lemma *uminus-aezfun-apply* [*simp*]: *aezfun* (-*a*) *x* = - *aezfun* *a* *x*
 $\langle proof \rangle$

lemma *aezfun-left-minus* [*simp*] :
fixes *a* :: ('a::*group-add*, 'b) *aezfun*
shows - *a* + *a* = 0
 $\langle proof \rangle$

lemma *aezfun-diff-minus* [*simp*] :
fixes *a* *b* :: ('a::*group-add*, 'b) *aezfun*
shows *a* - *b* = *a* + - *b*
 $\langle proof \rangle$

instance *aezfun* :: (*group-add*, *type*) *group-add*
 $\langle proof \rangle$

instance *aezfun* :: (*ab-group-add*, *type*) *ab-group-add*
 $\langle proof \rangle$

instantiation *aezfun* :: ({*one*, *zero*}, *zero*) *one*
begin
lift-definition *one-aezfun* :: ('a, 'b) *aezfun* **is** 1 δ 0

<proof>
instance <proof>
end

lemma *one-aezfun-transfer* : $Abs-aezfun (1 \delta 0) = 1$
 <proof>

lemma *one-aezfun-apply* [simp]: $aezfun 1 x = (1 \delta 0) x$
 <proof>

lemma *one-aezfun-apply-eq* [simp]: $aezfun 1 0 = 1$
 <proof>

lemma *one-aezfun-apply-neg* [simp]: $x \neq 0 \implies aezfun 1 x = 0$
 <proof>

instance *aezfun* :: (zero-neq-one, zero) zero-neq-one
 <proof>

instantiation *aezfun* :: ({comm-monoid-add,times}, group-add) times
begin

lift-definition *times-aezfun* :: ('a, 'b) aezfun \Rightarrow ('a, 'b) aezfun \Rightarrow ('a, 'b) aezfun
is $\lambda f g. convolution f g$
 <proof>

instance <proof>
end

lemma *convolution-transfer* :
assumes $f \in aezfun-set \ g \in aezfun-set$
shows $Abs-aezfun (convolution f g) = Abs-aezfun f * Abs-aezfun g$
 <proof>

instance *aezfun* :: ({comm-monoid-add,mult-zero}, group-add) mult-zero
 <proof>

instance *aezfun* :: (semiring-0, group-add) semiring-0
 <proof>

instance *aezfun* :: (ring, group-add) ring <proof>

instance *aezfun* :: ({semiring-0,monoid-mult,zero-neq-one}, group-add) monoid-mult
 <proof>

instance *aezfun* :: (ring-1, group-add) ring-1 <proof>

1.5.5 Transfer facts

abbreviation *aezdeltafun* :: 'b::zero \Rightarrow 'a \Rightarrow ('b,'a) aezfun (**infix** $\delta\delta$ 70)
where $b \delta\delta a \equiv Abs-aezfun (b \delta a)$

lemma *aezdeltafun* : $aezfun (b \delta\delta a) = b \delta a$
 ⟨proof⟩

lemma *aezdeltafun-plus* : $(b+c) \delta\delta a = (b \delta\delta a) + (c \delta\delta a)$
 ⟨proof⟩

lemma *times-aezdeltafun-aezdeltafun* :
fixes $b1\ b2 :: 'b::\{comm-monoid-add,mult-zero\}$
shows $(b1 \delta\delta a1) * (b2 \delta\delta a2) = (b1 * b2) \delta\delta (a1 + a2)$
 ⟨proof⟩

lemma *aezfun-restrict-and-extend0* : $(aezfun\ x)\downarrow A \in aezfun\ set$
 ⟨proof⟩

lemma *aezdeltafun-decomp* :
fixes $b :: 'b::semiring-1$
shows $b \delta\delta a = (b \delta\delta 0) * (1 \delta\delta a)$
 ⟨proof⟩

lemma *aezdeltafun-decomp'* :
fixes $b :: 'b::semiring-1$
shows $b \delta\delta a = (1 \delta\delta a) * (b \delta\delta 0)$
 ⟨proof⟩

lemma *supp-aezfun1* :
 $supp (aezfun (1 :: ('a::zero-neq-one, 'b::zero) aezfun)) = 0$
 ⟨proof⟩

lemma *supp-aezfun-diff* :
 $supp (aezfun (x - y)) \subseteq supp (aezfun x) \cup supp (aezfun y)$
 ⟨proof⟩

lemma *supp-aezfun-times* :
 $supp (aezfun (x * y)) \subseteq supp (aezfun x) + supp (aezfun y)$
 ⟨proof⟩

1.5.6 Almost-everywhere-zero functions with constrained support

The name of the next definition anticipates *aezfun-common-supp-spanning-set* below.

definition *aezfun-setspace* :: $'a\ set \Rightarrow ('b::zero, 'a) aezfun\ set$
where $aezfun-setspace A = \{x. supp (aezfun x) \subseteq A\}$

lemma *aezfun-setspaceD* : $x \in aezfun-setspace A \Longrightarrow supp (aezfun x) \subseteq A$
 ⟨proof⟩

lemma *aezfun-setspaceI* : $supp (aezfun x) \subseteq A \Longrightarrow x \in aezfun-setspace A$

<proof>

lemma *aezfun-common-supp-spanning-set* :

assumes *finite A*

shows $\exists as. \text{distinct } as \wedge \text{set } as = A \wedge (\forall x::('b::\text{semiring-1}, 'a) \text{ aezfun} \in \text{aezfun-setspace } A. \exists bs. \text{length } bs = \text{length } as \wedge x = (\sum (b,a) \leftarrow \text{zip } bs \text{ as. } b \ \delta\delta \ a)$

<proof>

lemma *aezfun-common-supp-spanning-set-decomp* :

fixes *G :: 'g::group-add set*

assumes *finite G*

shows $\exists gs. \text{distinct } gs \wedge \text{set } gs = G \wedge (\forall x::('r::\text{semiring-1}, 'g) \text{ aezfun} \in \text{aezfun-setspace } G. \exists rs. \text{length } rs = \text{length } gs \wedge x = (\sum (r,g) \leftarrow \text{zip } rs \text{ gs. } (r \ \delta\delta \ 0) * (1 \ \delta\delta \ g))$

<proof>

lemma *aezfun-decomp-aezdeltafun* :

fixes *c :: ('r::semiring-1, 'a) aezfun*

shows $\exists ras. \text{set } (\text{map } \text{snd } ras) = \text{supp } (\text{aezfun } c) \wedge c = (\sum (r,a) \leftarrow ras. r \ \delta\delta \ a)$

<proof>

lemma *aezfun-setspace-el-decomp-aezdeltafun* :

fixes *c :: ('r::semiring-1, 'a) aezfun*

shows $c \in \text{aezfun-setspace } A$

$\implies \exists ras. \text{set } (\text{map } \text{snd } ras) \subseteq A \wedge c = (\sum (r,a) \leftarrow ras. r \ \delta\delta \ a)$

<proof>

lemma *aezdelta0fun-commutes'* :

fixes *b1 b2 :: 'b::comm-semiring-1*

shows $b1 \ \delta\delta \ a * (b2 \ \delta\delta \ 0) = b2 \ \delta\delta \ 0 * (b1 \ \delta\delta \ a)$

<proof>

lemma *aezdelta0fun-commutes* :

fixes *b :: 'b::comm-semiring-1*

shows $c * (b \ \delta\delta \ 0) = b \ \delta\delta \ 0 * c$

<proof>

The following definition constrains the support of arbitrary almost-everywhere-zero functions, as a sort of projection onto a *aezfun-setspace*.

definition *aezfun-setspace-proj* :: *'a set* $\implies ('b::zero, 'a) \text{ aezfun} \implies ('b::zero, 'a) \text{ aezfun}$

where $\text{aezfun-setspace-proj } A \ x \equiv \text{Abs-aezfun } ((\text{aezfun } x) \downarrow A)$

lemma *aezfun-setspace-projD1* :

$a \in A \implies \text{aezfun } (\text{aezfun-setspace-proj } A \ x) \ a = \text{aezfun } x \ a$

<proof>

lemma *aezfun-setspace-projD2* :

$a \notin A \implies \text{aezfun } (\text{aezfun-setspace-proj } A \ x) \ a = 0$
<proof>

lemma *aezfun-setspace-proj-in-setspace* :

$\text{aezfun-setspace-proj } A \ x \in \text{aezfun-setspace } A$
<proof>

lemma *aezfun-setspace-proj-zero* : $\text{aezfun-setspace-proj } A \ 0 = 0$

<proof>

lemma *aezfun-setspace-proj-aezdeltafun* :

$\text{aezfun-setspace-proj } A \ (b \ \delta \delta \ a) = (\text{if } a \in A \ \text{then } b \ \delta \delta \ a \ \text{else } 0)$
<proof>

lemma *aezfun-setspace-proj-add* :

$\text{aezfun-setspace-proj } A \ (x+y)$
 $= \text{aezfun-setspace-proj } A \ x + \text{aezfun-setspace-proj } A \ y$
<proof>

lemma *aezfun-setspace-proj-sum-list* :

$\text{aezfun-setspace-proj } A \ (\sum x \leftarrow xs. \ f \ x) = (\sum x \leftarrow xs. \ \text{aezfun-setspace-proj } A \ (f \ x))$
<proof>

lemma *aezfun-setspace-proj-sum-list-prod* :

$\text{aezfun-setspace-proj } A \ (\sum (x,y) \leftarrow xys. \ f \ x \ y)$
 $= (\sum (x,y) \leftarrow xys. \ \text{aezfun-setspace-proj } A \ (f \ x \ y))$
<proof>

1.6 Polynomials

lemma *nonzero-coeffs-nonzero-poly* : $as \neq [] \implies \text{set } as \neq 0 \implies \text{Poly } as \neq 0$

<proof>

lemma *const-poly-nonzero-coeff* :

assumes $\text{degree } p = 0 \ p \neq 0$

shows $\text{coeff } p \ 0 \neq 0$

<proof>

lemma *pCons-induct2* [*case-names 00 lpCons rpCons pCons2*]:

assumes $00: P \ 0 \ 0$

and $lpCons: \bigwedge a \ p. \ a \neq 0 \ \vee \ p \neq 0 \implies P \ (pCons \ a \ p) \ 0$

and $rpCons: \bigwedge b \ q. \ b \neq 0 \ \vee \ q \neq 0 \implies P \ 0 \ (pCons \ b \ q)$

and $pCons2: \bigwedge a \ p \ b \ q. \ a \neq 0 \ \vee \ p \neq 0 \implies b \neq 0 \ \vee \ q \neq 0 \implies P \ p \ q$
 $\implies P \ (pCons \ a \ p) \ (pCons \ b \ q)$

shows $P \ p \ q$

<proof>

1.7 Algebra of sets

1.7.1 General facts

lemma *zeroset-eqI*: $0 \in A \implies (\bigwedge a. a \in A \implies a = 0) \implies A = 0$
<proof>

lemma *sum-list-sets-single* : $(\sum X \leftarrow [A]. X) = A$
<proof>

lemma *sum-list-sets-double* : $(\sum X \leftarrow [A, B]. X) = A + B$
<proof>

1.7.2 Additive independence of sets

primrec *add-independentS* :: 'a::monoid-add set list \implies bool
where *add-independentS* [] = True
| *add-independentS* (A#As) = (*add-independentS* As
 $\wedge (\forall x \in (\sum B \leftarrow As. B). \forall a \in A. a + x = 0 \longrightarrow a = 0)$)

lemma *add-independentS-doubleI*:
assumes $\bigwedge b a. b \in B \implies a \in A \implies a + b = 0 \implies a = 0$
shows *add-independentS* [A,B]
<proof>

lemma *add-independentS-doubleD*:
assumes *add-independentS* [A,B]
shows $\bigwedge b a. b \in B \implies a \in A \implies a + b = 0 \implies a = 0$
<proof>

lemma *add-independentS-double-iff* :
add-independentS [A,B] = $(\forall b \in B. \forall a \in A. a + b = 0 \longrightarrow a = 0)$
<proof>

lemma *add-independentS-Cons-conv-sum-right* :
add-independentS (A#As)
= (*add-independentS* [A, $\sum B \leftarrow As. B$] \wedge *add-independentS* As)
<proof>

lemma *add-independentS-double-sum-conv-append* :
[[$\forall X \in \text{set } As. 0 \in X$; *add-independentS* As; *add-independentS* Bs;
 add-independentS [$\sum X \leftarrow As. X, \sum X \leftarrow Bs. X$]]]
 \implies *add-independentS* (As@Bs)
<proof>

lemma *add-independentS-ConsI* :
assumes *add-independentS* As
 $\bigwedge x a. [x \in (\sum X \leftarrow As. X); a \in A; a + x = 0] \implies a = 0$
shows *add-independentS* (A#As)
<proof>

lemma *add-independentS-append-reduce-right* :
 $add_independentS\ (As@Bs) \implies add_independentS\ Bs$
 $\langle proof \rangle$

lemma *add-independentS-append-reduce-left* :
 $add_independentS\ (As@Bs) \implies 0 \in (\sum X \leftarrow Bs.\ X) \implies add_independentS\ As$
 $\langle proof \rangle$

lemma *add-independentS-append-conv-double-sum* :
 $add_independentS\ (As@Bs) \implies add_independentS\ [\sum X \leftarrow As.\ X, \sum X \leftarrow Bs.\ X]$
 $\langle proof \rangle$

1.7.3 Inner direct sums

definition *inner-dirsum* :: 'a::monoid-add set list \Rightarrow 'a set
where *inner-dirsum* As = (if *add-independentS* As then $\sum A \leftarrow As.\ A$ else 0)

Some syntactic sugar for *inner-dirsum*, borrowed from theory *HOL.List*.

syntax

-inner-dirsum :: ptrn \Rightarrow 'a list \Rightarrow 'b \Rightarrow 'b
 $((\exists \oplus \leftarrow \cdot \cdot) [0, 51, 10] 10)$

translations — Beware of argument permutation!

$\oplus M \leftarrow Ms.\ b == CONST\ inner_dirsum\ (CONST\ map\ (\%M.\ b)\ Ms)$

abbreviation *inner-dirsum-double* ::

'a::monoid-add set \Rightarrow 'a set \Rightarrow 'a set (**infixr** \oplus 70)
where *inner-dirsum-double* A B $\equiv inner_dirsum\ [A,B]$

lemma *inner-dirsumI* :

$M = (\sum N \leftarrow Ns.\ N) \implies add_independentS\ Ns \implies M = (\oplus N \leftarrow Ns.\ N)$
 $\langle proof \rangle$

lemma *inner-dirsum-doubleI* :

$M = A + B \implies add_independentS\ [A,B] \implies M = A \oplus B$
 $\langle proof \rangle$

lemma *inner-dirsumD* :

$add_independentS\ Ms \implies (\oplus M \leftarrow Ms.\ M) = (\sum M \leftarrow Ms.\ M)$
 $\langle proof \rangle$

lemma *inner-dirsumD2* : $\neg add_independentS\ Ms \implies (\oplus M \leftarrow Ms.\ M) = 0$
 $\langle proof \rangle$

lemma *inner-dirsum-Nil* : $(\oplus A \leftarrow [].\ A) = 0$
 $\langle proof \rangle$

lemma *inner-dirsum-singleD* : $(\oplus N \leftarrow [M].\ N) = M$
 $\langle proof \rangle$

lemma *inner-dirsum-doubleD* : $add-independentS [M,N] \implies M \oplus N = M + N$
 ⟨proof⟩

lemma *inner-dirsum-Cons* :
 $add-independentS (A \# As) \implies (\bigoplus X \leftarrow (A \# As). X) = A \oplus (\bigoplus X \leftarrow As. X)$
 ⟨proof⟩

lemma *inner-dirsum-append* :
 $add-independentS (As @ Bs) \implies 0 \in (\sum X \leftarrow Bs. X)$
 $\implies (\bigoplus X \leftarrow (As @ Bs). X) = (\bigoplus X \leftarrow As. X) \oplus (\bigoplus X \leftarrow Bs. X)$
 ⟨proof⟩

lemma *inner-dirsum-double-left0* : $0 \oplus A = A$
 ⟨proof⟩

lemma *add-independentS-Cons-conv-dirsum-right* :
 $add-independentS (A \# As) = (add-independentS [A, \bigoplus B \leftarrow As. B]$
 $\wedge add-independentS As)$
 ⟨proof⟩

lemma *sum-list-listset-dirsum* :
 $add-independentS As \implies as \in listset As \implies sum-list as \in (\bigoplus A \leftarrow As. A)$
 ⟨proof⟩

2 Groups

2.1 Locales and basic facts

2.1.1 Locale *Group* and finite variant *FinGroup*

Define a *Group* to be a closed subset of *UNIV* for the *group-add* class.

locale *Group* =
 fixes $G :: 'g::group-add set$
 assumes *nonempty* : $G \neq \{\}$
 and *diff-closed*: $\bigwedge g h. g \in G \implies h \in G \implies g - h \in G$

lemma *trivial-Group* : *Group* 0
 ⟨proof⟩

locale *FinGroup* = *Group* G
 for $G :: 'g::group-add set$
 + assumes *finite*: *finite* G

lemma (in *FinGroup*) *Group* : *Group* G ⟨proof⟩

lemma (in *Group*) *FinGroupI* : *finite* $G \implies FinGroup G$ ⟨proof⟩

context *Group*

begin

abbreviation *Subgroup* ::

'g set \Rightarrow *bool* **where** *Subgroup* $H \equiv$ *Group* $H \wedge H \subseteq G$

lemma *SubgroupD1* : *Subgroup* $H \Longrightarrow$ *Group* H \langle *proof* \rangle

lemma *zero-closed* : $0 \in G$

\langle *proof* \rangle

lemma *obtain-nonzero*: **assumes** $G \neq 0$ **obtains** g **where** $g \in G$ **and** $g \neq 0$

\langle *proof* \rangle

lemma *zeroS-closed* : $0 \subseteq G$

\langle *proof* \rangle

lemma *neg-closed* : $g \in G \Longrightarrow -g \in G$

\langle *proof* \rangle

lemma *add-closed* : $g \in G \Longrightarrow h \in G \Longrightarrow g + h \in G$

\langle *proof* \rangle

lemma *neg-add-closed* : $g \in G \Longrightarrow h \in G \Longrightarrow -g + h \in G$

\langle *proof* \rangle

lemma *sum-list-closed* : $set (map f as) \subseteq G \Longrightarrow (\sum a \leftarrow as. f a) \in G$

\langle *proof* \rangle

lemma *sum-list-closed-prod* :

$set (map (case-prod f) xys) \subseteq G \Longrightarrow (\sum (x,y) \leftarrow xys. f x y) \in G$

\langle *proof* \rangle

lemma *set-plus-closed* : $A \subseteq G \Longrightarrow B \subseteq G \Longrightarrow A + B \subseteq G$

\langle *proof* \rangle

lemma *zip-add-closed* :

$set as \subseteq G \Longrightarrow set bs \subseteq G \Longrightarrow set [a + b. (a,b) \leftarrow zip as bs] \subseteq G$

\langle *proof* \rangle

lemma *list-diff-closed* :

$set gs \subseteq G \Longrightarrow set hs \subseteq G \Longrightarrow set [x - y. (x,y) \leftarrow zip gs hs] \subseteq G$

\langle *proof* \rangle

lemma *add-closed-converse-right* : $g+x \in G \Longrightarrow g \in G \Longrightarrow x \in G$

\langle *proof* \rangle

lemma *add-closed-inverse-right* : $x \notin G \Longrightarrow g \in G \Longrightarrow g+x \notin G$

\langle *proof* \rangle

lemma *add-closed-converse-left* : $g+x \in G \implies x \in G \implies g \in G$
 ⟨proof⟩

lemma *add-closed-inverse-left* : $g \notin G \implies x \in G \implies g+x \notin G$
 ⟨proof⟩

lemma *right-translate-bij* :
 assumes $g \in G$
 shows *bij-betw* $(\lambda x. x + g)$ G G
 ⟨proof⟩

lemma *right-translate-sum* : $g \in G \implies (\sum_{h \in G}. f h) = (\sum_{h \in G}. f (h + g))$
 ⟨proof⟩

end

2.1.2 Abelian variant locale *AbGroup*

locale *AbGroup* = *Group* G
 for $G :: 'g::ab-group-add$ set
begin

lemmas *nonempty* = *nonempty*
lemmas *zero-closed* = *zero-closed*
lemmas *diff-closed* = *diff-closed*
lemmas *add-closed* = *add-closed*
lemmas *neg-closed* = *neg-closed*

lemma *sum-closed* : *finite* $A \implies f \text{ ` } A \subseteq G \implies (\sum_{a \in A}. f a) \in G$
 ⟨proof⟩

lemma *subset-plus-right* : $A \subseteq G + A$
 ⟨proof⟩

lemma *subset-plus-left* : $A \subseteq A + G$
 ⟨proof⟩

end

2.2 Right cosets

context *Group*
begin

definition *rcoset-rel* :: $'g$ set $\Rightarrow ('g \times 'g)$ set
 where *rcoset-rel* $H \equiv \{(g, g'). g \in G \wedge g' \in G \wedge g - g' \in H\}$

lemma (in *Group*) *rcosets* :
 assumes *subgrp*: *Subgroup* H and $g: g \in G$
 shows $(\text{rcoset-rel } H) \text{ `` } \{g\} = H + \{g\}$

$\langle proof \rangle$

lemma *rcoset-equiv* :

assumes *Subgroup H*

shows *equiv G (rcoset-rel H)*

$\langle proof \rangle$

lemma *rcoset0* : *Subgroup H* \implies *(rcoset-rel H) "{0} = H*

$\langle proof \rangle$

definition *is-rcoset-replist* :: 'g set \Rightarrow 'g list \Rightarrow bool

where *is-rcoset-replist H gs*

\equiv *set gs \subseteq G \wedge distinct (map ($\lambda g.$ (rcoset-rel H) "{g}) gs)*
 \wedge *G = ($\bigcup_{g \in \text{set } gs.}$ (rcoset-rel H) "{g})*

lemma *is-rcoset-replistD-set* : *is-rcoset-replist H gs* \implies *set gs \subseteq G*

$\langle proof \rangle$

lemma *is-rcoset-replistD-distinct* :

is-rcoset-replist H gs \implies *distinct (map ($\lambda g.$ (rcoset-rel H) "{g}) gs)*

$\langle proof \rangle$

lemma *is-rcoset-replistD-cosets* :

is-rcoset-replist H gs \implies *G = ($\bigcup_{g \in \text{set } gs.}$ (rcoset-rel H) "{g})*

$\langle proof \rangle$

lemma *group-eq-subgrp-rcoset-un* :

Subgroup H \implies *is-rcoset-replist H gs* \implies *G = ($\bigcup_{g \in \text{set } gs.}$ H + {g})*

$\langle proof \rangle$

lemma *is-rcoset-replist-imp-nrelated-nth* :

assumes *Subgroup H is-rcoset-replist H gs*

shows $\bigwedge i j. i < \text{length } gs \implies j < \text{length } gs \implies i \neq j \implies gs!i - gs!j \notin H$

$\langle proof \rangle$

lemma *is-rcoset-replist-Cons* :

is-rcoset-replist H (g#gs) \longleftrightarrow

g \in G \wedge set gs \subseteq G

\wedge *(rcoset-rel H) "{g} \notin set (map ($\lambda x.$ (rcoset-rel H) "{x}) gs)*

\wedge *distinct (map ($\lambda x.$ (rcoset-rel H) "{x}) gs)*

\wedge *G = (rcoset-rel H) "{g} \cup ($\bigcup_{x \in \text{set } gs.}$ (rcoset-rel H) "{x})*

$\langle proof \rangle$

lemma *rcoset-replist-Hrep* :

assumes *Subgroup H is-rcoset-replist H gs*

shows $\exists g \in \text{set } gs. g \in H$

$\langle proof \rangle$

lemma *rcoset-replist-reorder* :

is-rcoset-replist $H (gs @ g \# gs')$ \implies *is-rcoset-replist* $H (g \# gs @ gs')$
 <proof>

lemma *rcoset-replist-replacehd* :

assumes *Subgroup* H $g' \in (\text{rcoset-rel } H) \{g\}$ *is-rcoset-replist* $H (g \# gs)$

shows *is-rcoset-replist* $H (g' \# gs)$

<proof>

end

lemma (in *FinGroup*) *ex-rcoset-replist* :

assumes *Subgroup* H

shows $\exists gs. \text{is-rcoset-replist } H gs$

<proof>

lemma (in *FinGroup*) *ex-rcoset-replist-hd0* :

assumes *Subgroup* H

shows $\exists gs. \text{is-rcoset-replist } H (0 \# gs)$

<proof>

2.3 Group homomorphisms

2.3.1 Preliminaries

definition *ker* :: $(a \Rightarrow b :: \text{zero}) \Rightarrow 'a \text{ set}$

where $\text{ker } f = \{a. f a = 0\}$

lemma *kerI* : $f a = 0 \implies a \in \text{ker } f$

<proof>

lemma *kerD* : $a \in \text{ker } f \implies f a = 0$

<proof>

lemma *ker-im-iff* : $(A \neq \{\}) \wedge A \subseteq \text{ker } f = (f ' A = 0)$

<proof>

2.3.2 Locales

The *supp* condition is not strictly necessary, but helps with equality and uniqueness arguments.

locale *GroupHom* = *Group* G

for $G :: 'g :: \text{group-add set}$

+ **fixes** $T :: 'g \Rightarrow 'h :: \text{group-add}$

assumes $\text{hom} : \bigwedge g g'. g \in G \implies g' \in G \implies T (g + g') = T g + T g'$

and $\text{supp} : \text{supp } T \subseteq G$

abbreviation (in *GroupHom*) $\text{Ker} \equiv \text{ker } T \cap G$

abbreviation (in *GroupHom*) $\text{Im}G \equiv T ' G$

locale *GroupEnd* = *GroupHom* *G* *T*
for *G* :: 'g::group-add set
and *T* :: 'g ⇒ 'g
+ **assumes** *endomorph*: $\text{Im}G \subseteq G$

locale *GroupIso* = *GroupHom* *G* *T*
for *G* :: 'g::group-add set
and *T* :: 'g ⇒ 'h::group-add
+ **fixes** *H* :: 'h set
assumes *bijjective*: *bij-betw* *T* *G* *H*

2.3.3 Basic facts

lemma (**in** *Group*) *trivial-GroupHom* : *GroupHom* *G* (0::('g⇒'h::group-add))
⟨*proof*⟩

lemma (**in** *Group*) *GroupHom-idhom* : *GroupHom* *G* (*id*↓*G*)
⟨*proof*⟩

context *GroupHom*
begin

lemma *im-zero* : $T\ 0 = 0$
⟨*proof*⟩

lemma *zero-in-Ker* : $0 \in \text{Ker}$
⟨*proof*⟩

lemma *comp-zero* : $T \circ 0 = 0$
⟨*proof*⟩

lemma *im-neg* : $T\ (-\ g) = -\ T\ g$
⟨*proof*⟩

lemma *im-diff* : $g \in G \implies g' \in G \implies T\ (g - g') = T\ g - T\ g'$
⟨*proof*⟩

lemma *eq-im-imp-diff-in-Ker* : $\llbracket g \in G; h \in G; T\ g = T\ h \rrbracket \implies g - h \in \text{Ker}$
⟨*proof*⟩

lemma *im-sum-list-prod* :
 $\text{set}\ (\text{map}\ (\text{case-prod}\ f)\ xys) \subseteq G$
 $\implies T\ (\sum_{(x,y) \leftarrow xys} f\ x\ y) = (\sum_{(x,y) \leftarrow xys} T\ (f\ x\ y))$
⟨*proof*⟩

lemma *distrib-comp-sum-left* :
 $\text{range}\ S \subseteq G \implies \text{range}\ S' \subseteq G \implies T \circ (S + S') = (T \circ S) + (T \circ S')$
⟨*proof*⟩

lemma *Ker-Im-iff* : $(\text{Ker} = G) = (\text{Im}G = 0)$
⟨proof⟩

lemma *Ker0-imp-inj-on* :
 assumes $\text{Ker} \subseteq 0$
 shows $\text{inj-on } T \ G$
⟨proof⟩

lemma *inj-on-imp-Ker0* :
 assumes $\text{inj-on } T \ G$
 shows $\text{Ker} = 0$
⟨proof⟩

lemma *nonzero-Ker-el-imp-n-inj* :
 $g \in G \implies g \neq 0 \implies T \ g = 0 \implies \neg \text{inj-on } T \ G$
⟨proof⟩

lemma *Group-Ker* : $\text{Group } \text{Ker}$
⟨proof⟩

lemma *Group-Im* : $\text{Group } \text{Im}G$
⟨proof⟩

lemma *GroupHom-restrict0-subgroup* :
 assumes $\text{Subgroup } H$
 shows $\text{GroupHom } H \ (T \downarrow H)$
⟨proof⟩

lemma *im-subgroup* :
 assumes $\text{Subgroup } H$
 shows $\text{Group.Subgroup } \text{Im}G \ (T \uparrow H)$
⟨proof⟩

lemma *GroupHom-composite-left* :
 assumes $\text{Im}G \subseteq H \ \text{GroupHom } H \ S$
 shows $\text{GroupHom } G \ (S \circ T)$
⟨proof⟩

lemma *idhom-left* : $T \uparrow G \subseteq H \implies (\text{id} \downarrow H) \circ T = T$
⟨proof⟩

end

2.3.4 Basic facts about endomorphisms

context *GroupEnd*
begin

lemmas $\text{hom} = \text{hom}$

lemma *range* : $\text{range } T \subseteq G$
<proof>

lemma *proj-decomp* :
 assumes $\bigwedge g. g \in G \implies T (T g) = T g$
 shows $G = \text{Ker } T \oplus \text{Im } T$
<proof>

end

2.3.5 Basic facts about isomorphisms

context *GroupIso*
begin

abbreviation *invT* $\equiv (\text{the-inv-into } G T) \downarrow H$

lemma *ImT* : $\text{Im } T = H$ *<proof>*

lemma *GroupH* : *Group* H *<proof>*

lemma *invT-onto* : $\text{inv } T \text{ ' } H = G$
<proof>

lemma *inj-on-invT* : *inj-on* $\text{inv } T \text{ } H$
<proof>

lemma *bij-into-invT* : *bij-into* $\text{inv } T \text{ } H \text{ } G$
<proof>

lemma *invT-into* : $h \in H \implies \text{inv } T \text{ } h \in G$
<proof>

lemma *T-invT* : $h \in H \implies T (\text{inv } T \text{ } h) = h$
<proof>

lemma *invT-eq* : $g \in G \implies T g = h \implies \text{inv } T \text{ } h = g$
<proof>

lemma *inv* : *GroupIso* $H \text{ } \text{inv } T \text{ } G$
<proof>

end

2.3.6 Hom-sets

definition *GroupHomSet* :: $'g::\text{group-add set} \Rightarrow 'h::\text{group-add set} \Rightarrow ('g \Rightarrow 'h) \text{ set}$
 where $\text{GroupHomSet } G \text{ } H \equiv \{T. \text{GroupHom } G \text{ } T\} \cap \{T. T \text{ ' } G \subseteq H\}$

lemma *GroupHomSetI* :
 $\text{GroupHom } G \ T \implies T \ ' \ G \subseteq H \implies T \in \text{GroupHomSet } G \ H$
 ⟨proof⟩

lemma *GroupHomSetD-GroupHom* :
 $T \in \text{GroupHomSet } G \ H \implies \text{GroupHom } G \ T$
 ⟨proof⟩

lemma *GroupHomSetD-Im* : $T \in \text{GroupHomSet } G \ H \implies T \ ' \ G \subseteq H$
 ⟨proof⟩

lemma (in *Group*) *Group-GroupHomSet* :
 fixes $H :: 'h::\text{ab-group-add set}$
 assumes *AbGroup* H
 shows *Group* ($\text{GroupHomSet } G \ H$)
 ⟨proof⟩

2.4 Facts about collections of groups

lemma *listset-Group-plus-closed* :

$$\llbracket \forall G \in \text{set } Gs. \text{Group } G; as \in \text{listset } Gs; bs \in \text{listset } Gs \rrbracket$$

$$\implies [a+b. (a,b) \leftarrow \text{zip } as \ bs] \in \text{listset } Gs$$

 ⟨proof⟩

lemma *AbGroup-set-plus* :
 assumes *AbGroup* H *AbGroup* G
 shows *AbGroup* ($H + G$)
 ⟨proof⟩

lemma *AbGroup-sum-list* :
 $(\forall G \in \text{set } Gs. \text{AbGroup } G) \implies \text{AbGroup } (\sum G \leftarrow Gs. G)$
 ⟨proof⟩

lemma *AbGroup-subset-sum-list* :
 $\forall G \in \text{set } Gs. \text{AbGroup } G \implies H \in \text{set } Gs \implies H \subseteq (\sum G \leftarrow Gs. G)$
 ⟨proof⟩

lemma *independent-AbGroups-pairwise-int0* :

$$\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; G \in \text{set } Gs; G' \in \text{set } Gs;$$

$$G \neq G' \rrbracket \implies G \cap G' = 0$$

 ⟨proof⟩

lemma *independent-AbGroups-pairwise-int0-double* :
 assumes *AbGroup* G *AbGroup* G' *add-independentS* [G, G']
 shows $G \cap G' = 0$
 ⟨proof⟩

2.5 Inner direct sums of Abelian groups

2.5.1 General facts

lemma *AbGroup-inner-dirsum* :

$\forall G \in \text{set } Gs. \text{AbGroup } G \implies \text{AbGroup } (\bigoplus G \leftarrow Gs. G)$
 $\langle \text{proof} \rangle$

lemma *inner-dirsum-double-leftfull-imp-right0*:

assumes $\text{Group } A \ B \neq \{\}$ $A = A \oplus B$

shows $B = 0$

$\langle \text{proof} \rangle$

lemma *AbGroup-subset-inner-dirsum* :

$\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; H \in \text{set } Gs \rrbracket$

$\implies H \subseteq (\bigoplus G \leftarrow Gs. G)$

$\langle \text{proof} \rangle$

lemma *AbGroup-nth-subset-inner-dirsum* :

$\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; n < \text{length } Gs \rrbracket$

$\implies Gs!n \subseteq (\bigoplus G \leftarrow Gs. G)$

$\langle \text{proof} \rangle$

lemma *AbGroup-inner-dirsum-el-decomp-ex1-double* :

assumes $\text{AbGroup } G \ \text{AbGroup } H \ \text{add-independentS } [G, H] \ x \in G \oplus H$

shows $\exists !gh. \text{fst } gh \in G \wedge \text{snd } gh \in H \wedge x = \text{fst } gh + \text{snd } gh$

$\langle \text{proof} \rangle$

lemma *AbGroup-inner-dirsum-el-decomp-ex1* :

$\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs \rrbracket$

$\implies \forall x \in (\bigoplus G \leftarrow Gs. G). \exists !gs \in \text{listset } Gs. x = \text{sum-list } gs$

$\langle \text{proof} \rangle$

lemma *AbGroup-inner-dirsum-pairwise-int0* :

$\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; G \in \text{set } Gs; G' \in \text{set } Gs; G \neq G' \rrbracket \implies G \cap G' = 0$

$\langle \text{proof} \rangle$

lemma *AbGroup-inner-dirsum-pairwise-int0-double* :

assumes $\text{AbGroup } G \ \text{AbGroup } G' \ \text{add-independentS } [G, G']$

shows $G \cap G' = 0$

$\langle \text{proof} \rangle$

2.5.2 Element decomposition and projection

definition *inner-dirsum-el-decomp* ::

$'g :: \text{ab-group-add set list} \Rightarrow ('g \Rightarrow 'g \text{ list}) (\bigoplus \leftarrow)$

where $\bigoplus Gs \leftarrow = (\lambda x. \text{if } x \in (\bigoplus G \leftarrow Gs. G)$

$\text{then THE } gs. gs \in \text{listset } Gs \wedge x = \text{sum-list } gs \text{ else []})$

abbreviation *inner-dirsum-el-decomp-double* ::

'g::ab-group-add set \Rightarrow 'g set \Rightarrow ('g \Rightarrow 'g list) (\oplus - \leftarrow) **where** $G \oplus H \leftarrow \equiv \oplus [G, H] \leftarrow$

abbreviation *inner-dirsum-el-decomp-nth* ::

'g::ab-group-add set list \Rightarrow nat \Rightarrow ('g \Rightarrow 'g) (\oplus \downarrow -)
where $\oplus Gs \downarrow n \equiv \text{restrict0 } (\lambda x. (\oplus Gs \leftarrow x)!n) (\oplus G \leftarrow Gs. G)$

lemma *AbGroup-inner-dirsum-el-decompI* :

$\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; x \in (\oplus G \leftarrow Gs. G) \rrbracket$
 $\implies (\oplus Gs \leftarrow x) \in \text{listset } Gs \wedge x = \text{sum-list } (\oplus Gs \leftarrow x)$
<proof>

lemma (**in** *AbGroup*) *abSubgroup-inner-dirsum-el-decomp-set* :

$\llbracket \forall H \in \text{set } Hs. \text{Subgroup } H; \text{add-independentS } Hs; x \in (\oplus H \leftarrow Hs. H) \rrbracket$
 $\implies \text{set } (\oplus Hs \leftarrow x) \subseteq G$
<proof>

lemma *AbGroup-inner-dirsum-el-decomp-eq* :

$\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; x \in (\oplus G \leftarrow Gs. G);$
 $gs \in \text{listset } Gs; x = \text{sum-list } gs \rrbracket \implies (\oplus Gs \leftarrow x) = gs$
<proof>

lemma *AbGroup-inner-dirsum-el-decomp-plus* :

assumes $\forall G \in \text{set } Gs. \text{AbGroup } G \text{ add-independentS } Gs$ $x \in (\oplus G \leftarrow Gs. G)$
 $y \in (\oplus G \leftarrow Gs. G)$
shows $(\oplus Gs \leftarrow (x+y)) = [a+b. (a,b) \leftarrow \text{zip } (\oplus Gs \leftarrow x) (\oplus Gs \leftarrow y)]$
<proof>

lemma *AbGroup-length-inner-dirsum-el-decomp* :

$\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; x \in (\oplus G \leftarrow Gs. G) \rrbracket$
 $\implies \text{length } (\oplus Gs \leftarrow x) = \text{length } Gs$
<proof>

lemma *AbGroup-inner-dirsum-el-decomp-in-nth* :

assumes $\forall G \in \text{set } Gs. \text{AbGroup } G \text{ add-independentS } Gs$ $n < \text{length } Gs$
 $x \in Gs!n$
shows $(\oplus Gs \leftarrow x) = (\text{replicate } (\text{length } Gs) 0)[n := x]$
<proof>

lemma *AbGroup-inner-dirsum-el-decomp-nth-in-nth* :

$\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; k < \text{length } Gs;$
 $n < \text{length } Gs; x \in Gs!n \rrbracket \implies (\oplus Gs \downarrow k) x = (\text{if } k = n \text{ then } x \text{ else } 0)$
<proof>

lemma *AbGroup-inner-dirsum-el-decomp-nth-id-on-nth* :

$\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; n < \text{length } Gs; x \in Gs!n \rrbracket$
 $\implies (\oplus Gs \downarrow n) x = x$
<proof>

lemma *AbGroup-inner-dirsum-el-decomp-nth-onto-nth* :
assumes $\forall G \in \text{set } Gs. \text{AbGroup } G \text{ add-independentS } Gs \ n < \text{length } Gs$
shows $(\bigoplus Gs \downarrow n) \text{ ' } (\bigoplus G \leftarrow Gs. G) = Gs!n$
<proof>

lemma *AbGroup-inner-dirsum-subset-proj-eq-0* :
assumes $Gs \neq [] \ \forall G \in \text{set } Gs. \text{AbGroup } G \text{ add-independentS } Gs$
 $X \subseteq (\bigoplus G \leftarrow Gs. G) \ \forall i < \text{length } Gs. (\bigoplus Gs \downarrow i) \text{ ' } X = 0$
shows $X = 0$
<proof>

lemma *GroupEnd-inner-dirsum-el-decomp-nth* :
assumes $\forall G \in \text{set } Gs. \text{AbGroup } G \text{ add-independentS } Gs \ n < \text{length } Gs$
shows $\text{GroupEnd } (\bigoplus G \leftarrow Gs. G) (\bigoplus Gs \downarrow n)$
<proof>

2.6 Rings

2.6.1 Preliminaries

lemma (*in ring-1*) *map-times-neg1-eq-map-uminus* : $[(-1)*r. r \leftarrow rs] = [-r. r \leftarrow rs]$
<proof>

2.6.2 Locale and basic facts

Define a *Ring1* to be a multiplicatively closed additive subgroup of *UNIV* for the *ring-1* class.

locale *Ring1* = *Group* *R*
for *R* :: 'r::ring-1 set
+ assumes *one-closed* : $1 \in R$
and *mult-closed*: $\bigwedge r \ s. r \in R \implies s \in R \implies r * s \in R$
begin

lemma *AbGroup* : *AbGroup* *R*
<proof>

lemmas *zero-closed* = *zero-closed*
lemmas *add-closed* = *add-closed*
lemmas *neg-closed* = *neg-closed*
lemmas *diff-closed* = *diff-closed*
lemmas *zip-add-closed* = *zip-add-closed*
lemmas *sum-closed* = *AbGroup.sum-closed*[*OF AbGroup*]
lemmas *sum-list-closed* = *sum-list-closed*
lemmas *sum-list-closed-prod* = *sum-list-closed-prod*
lemmas *list-diff-closed* = *list-diff-closed*

abbreviation *Subring1* :: 'r set \implies bool **where** *Subring1* *S* \equiv *Ring1* *S* \wedge $S \subseteq R$

lemma *Subring1D1* : *Subring1* *S* \implies *Ring1* *S* *<proof>*

end

lemma (in *ring-1*) *full-Ring1* : *Ring1 UNIV*
⟨*proof*⟩

2.7 The group ring

2.7.1 Definition and basic facts

Realize the group ring as the set of almost-every-zero functions from group to ring. One can recover the usual notion of group ring element by considering such a function to send group elements to their coefficients. Here the codomain of such functions is not restricted to some *Ring1* subset since we will not be interested in having the ability to change the ring of scalars for a group ring.

context *Group*
begin

abbreviation *group-ring* :: ('a::zero, 'g) *aezfun set*
where *group-ring* ≡ *aezfun-setspace* *G*

lemmas *group-ringD* = *aezfun-setspace-def*[of *G*]

lemma *RG-one-closed* : (1::('r::zero-neq-one, 'g) *aezfun*) ∈ *group-ring*
⟨*proof*⟩

lemma *RG-zero-closed* : (0::('r::zero, 'g) *aezfun*) ∈ *group-ring*
⟨*proof*⟩

lemma *RG-n0* : *group-ring* ≠ (0::('r::zero-neq-one, 'g) *aezfun set*)
⟨*proof*⟩

lemma *RG-mult-closed* :
defines *RG*: *RG* ≡ *group-ring* :: ('r::ring-1, 'g) *aezfun set*
shows $x \in RG \implies y \in RG \implies x * y \in RG$
⟨*proof*⟩

lemma *Ring1-RG* :
defines *RG*: *RG* ≡ *group-ring* :: ('r::ring-1, 'g) *aezfun set*
shows *Ring1 RG*
⟨*proof*⟩

lemma *RG-aezdeltafun-closed* :
defines *RG*: *RG* ≡ *group-ring* :: ('r::ring-1, 'g) *aezfun set*
assumes $g \in G$
shows $r \delta \delta g \in RG$
⟨*proof*⟩

lemma *RG-aezdelta0fun-closed* : (r::'r::ring-1) $\delta\delta$ 0 \in group-ring
 ⟨proof⟩

lemma *RG-sum-list-rddg-closed* :
defines *RG*: $RG \equiv$ group-ring :: ('r::ring-1, 'g) aezfun set
assumes set (map snd rgs) \subseteq G
shows $(\sum (r,g) \leftarrow$ rgs. r $\delta\delta$ g) \in RG
 ⟨proof⟩

lemmas *RG-el-decomp-aezdeltafun* = aezfun-sets-pan-el-decomp-aezdeltafun[of - G]

lemma *Subgroup-imp-Subring* :
fixes H :: 'g set
and FH :: ('r::ring-1, 'g) aezfun set
and FG :: ('r, 'g) aezfun set
defines FH \equiv Group.group-ring H
and FG \equiv group-ring
shows Subgroup H \implies Ring1.Subring1 FG FH
 ⟨proof⟩

end

lemma (in FinGroup) *group-ring-spanning-set* :
 \exists gs. distinct gs \wedge set gs = G
 \wedge (\forall f \in (group-ring :: ('b::semiring-1, 'g) aezfun set). \exists bs.
 length bs = length gs \wedge f = $(\sum (b,g) \leftarrow$ zip bs gs. (b $\delta\delta$ 0) * (1 $\delta\delta$ g)))
 ⟨proof⟩

2.7.2 Projecting almost-everywhere-zero functions onto a group ring

context Group
begin

abbreviation *RG-proj* \equiv aezfun-sets-pan-proj G

lemmas *RG-proj-in-RG* = aezfun-sets-pan-proj-in-sets-pan
lemmas *RG-proj-sum-list-prod* = aezfun-sets-pan-proj-sum-list-prod[of G]

lemma *RG-proj-mult-leftdelta'* :
fixes r s :: 'r::{comm-monoid-add, mult-zero}
shows g \in G \implies RG-proj (r $\delta\delta$ g * (s $\delta\delta$ g')) = r $\delta\delta$ g * RG-proj (s $\delta\delta$ g')
 ⟨proof⟩

lemma *RG-proj-mult-leftdelta* :
fixes r :: 'r::semiring-1
assumes g \in G
shows RG-proj ((r $\delta\delta$ g) * x) = r $\delta\delta$ g * RG-proj x

<proof>

lemma *RG-proj-mult-rightdelta'* :

fixes $r\ s :: 'r::\{\text{comm-monoid-add,mult-zero}\}$

assumes $g' \in G$

shows $RG\text{-proj } (r \ \delta\delta \ g * (s \ \delta\delta \ g')) = RG\text{-proj } (r \ \delta\delta \ g) * (s \ \delta\delta \ g')$

<proof>

lemma *RG-proj-mult-rightdelta* :

fixes $r :: 'r::\text{semiring-1}$

assumes $g \in G$

shows $RG\text{-proj } (x * (r \ \delta\delta \ g)) = (RG\text{-proj } x) * (r \ \delta\delta \ g)$

<proof>

lemma *RG-proj-mult-right* :

$x \in (\text{group-ring} :: ('r::\text{ring-1}, 'g) \text{ aezfun set})$

$\implies RG\text{-proj } (y * x) = RG\text{-proj } y * x$

<proof>

end

3 Modules

3.1 Locales and basic facts

3.1.1 Locales

locale *scalar-mult* =

fixes $smult :: 'r::\text{ring-1} \Rightarrow 'm::\text{ab-group-add} \Rightarrow 'm$ (**infixr** \cdot 70)

locale *R-scalar-mult* = *scalar-mult* $smult + Ring1\ R$

for $R :: 'r::\text{ring-1 set}$

and $smult :: 'r \Rightarrow 'm::\text{ab-group-add} \Rightarrow 'm$ (**infixr** \cdot 70)

lemma (**in** *scalar-mult*) *R-scalar-mult* : *R-scalar-mult UNIV*

<proof>

lemma (**in** *R-scalar-mult*) *Ring1* : *Ring1 R* *<proof>*

locale *RModule* = *R-scalars?*: *R-scalar-mult R smult + VecGroup?*: *Group M*

for $R :: 'r::\text{ring-1 set}$

and $smult :: 'r \Rightarrow 'm::\text{ab-group-add} \Rightarrow 'm$ (**infixr** \cdot 70)

and $M :: 'm \text{ set}$

+ **assumes** *smult-closed* : $\llbracket r \in R; m \in M \rrbracket \implies r \cdot m \in M$

and *smult-distrib-left* [*simp*] : $\llbracket r \in R; m \in M; n \in M \rrbracket$

$\implies r \cdot (m + n) = r \cdot m + r \cdot n$

and *smult-distrib-right* [*simp*] : $\llbracket r \in R; s \in R; m \in M \rrbracket$

$\implies (r + s) \cdot m = r \cdot m + s \cdot m$

and *smult-assoc* [*simp*] : $\llbracket r \in R; s \in R; m \in M \rrbracket$

$\implies r \cdot s \cdot m = (r * s) \cdot m$

and *one-smult* [*simp*] : $m \in M \implies 1 \cdot m = m$

lemmas *RModuleI* = *RModule.intro*[*OF R-scalar-mult.intro*]

locale *Module* = *RModule UNIV smult M*
for *smult* :: '*r*::*ring-1* \Rightarrow '*m*::*ab-group-add* \Rightarrow '*m* (**infixr** \cdot 70)
and *M* :: '*m* *set*

lemmas *ModuleI* = *RModuleI*[*of UNIV, OF full-Ring1, THEN Module.intro*]

3.1.2 Basic facts

lemma *trivial-RModule* :
fixes *smult* :: '*r*::*ring-1* \Rightarrow '*m*::*ab-group-add* \Rightarrow '*m* (**infixr** \cdot 70)
assumes *Ring1 R* $\forall r \in R. smult\ r\ (0::'m::ab-group-add) = 0$
shows *RModule R smult* ($0::'m\ set$)
<proof>

context *RModule*
begin

abbreviation *RSubmodule* :: '*m* *set* \Rightarrow *bool*
where *RSubmodule N* \equiv *RModule R smult N* \wedge *N* \subseteq *M*

lemma *Group* : *Group M*
<proof>

lemma *Subgroup-RSubmodule* : *RSubmodule N* \implies *Subgroup N*
<proof>

lemma *AbGroup* : *AbGroup M*
<proof>

lemmas *zero-closed* = *zero-closed*
lemmas *diff-closed* = *diff-closed*
lemmas *set-plus-closed* = *set-plus-closed*
lemmas *sum-closed* = *AbGroup.sum-closed*[*OF AbGroup*]

lemma *map-smult-closed* :
 $r \in R \implies set\ ms \subseteq M \implies set\ (map\ ((\cdot)\ r)\ ms) \subseteq M$
<proof>

lemma *zero-smult* : $m \in M \implies 0 \cdot m = 0$
<proof>

lemma *smult-zero* : $r \in R \implies r \cdot 0 = 0$
<proof>

lemma *neg-smult* : $r \in R \implies m \in M \implies (-r) \cdot m = - (r \cdot m)$
 ⟨proof⟩

lemma *neg-eq-neg1-smult* : $m \in M \implies (-1) \cdot m = - m$
 ⟨proof⟩

lemma *smult-neg* : $r \in R \implies m \in M \implies r \cdot (- m) = - (r \cdot m)$
 ⟨proof⟩

lemma *smult-distrib-left-diff* :
 $\llbracket r \in R; m \in M; n \in M \rrbracket \implies r \cdot (m - n) = r \cdot m - r \cdot n$
 ⟨proof⟩

lemma *smult-distrib-right-diff* :
 $\llbracket r \in R; s \in R; m \in M \rrbracket \implies (r - s) \cdot m = r \cdot m - s \cdot m$
 ⟨proof⟩

lemma *smult-sum-distrib* :
assumes $r \in R$
shows $\text{finite } A \implies f \text{ ` } A \subseteq M \implies r \cdot (\sum a \in A. f a) = (\sum a \in A. r \cdot f a)$
 ⟨proof⟩

lemma *sum-smult-distrib* :
assumes $m \in M$
shows $\text{finite } A \implies f \text{ ` } A \subseteq R \implies (\sum a \in A. f a) \cdot m = (\sum a \in A. (f a) \cdot m)$
 ⟨proof⟩

lemma *smult-sum-list-distrib* :
 $r \in R \implies \text{set } ms \subseteq M \implies r \cdot (\text{sum-list } ms) = (\sum m \leftarrow ms. r \cdot m)$
 ⟨proof⟩

lemma *sum-list-prod-map-smult-distrib* :
 $m \in M \implies \text{set } (\text{map } (\text{case-prod } f) \text{ } xys) \subseteq R$
 $\implies (\sum (x,y) \leftarrow xys. f x y) \cdot m = (\sum (x,y) \leftarrow xys. f x y \cdot m)$
 ⟨proof⟩

lemma *RSubmoduleI* :
assumes $\text{Subgroup } N \wedge r \in R \implies n \in N \implies r \cdot n \in N$
shows $R\text{Submodule } N$
 ⟨proof⟩

end

lemma (in *R-scalar-mult*) *listset-RModule-Rsmult-closed* :
 $\llbracket \forall M \in \text{set } Ms. R\text{Module } R \text{ smult } M; r \in R; ms \in \text{listset } Ms \rrbracket$
 $\implies [r \cdot m. m \leftarrow ms] \in \text{listset } Ms$
 ⟨proof⟩

context *Module*

begin

abbreviation *Submodule* :: 'm set \Rightarrow bool
 where *Submodule* \equiv *RModule.RSubmodule UNIV smult M*

lemmas *AbGroup* = *AbGroup*
lemmas *SubmoduleI* = *RSubmoduleI*

end

3.1.3 Module and submodule instances

lemma (**in** *R-scalar-mult*) *trivial-RModule* :
 $(\bigwedge r. r \in R \Longrightarrow r \cdot 0 = 0) \Longrightarrow RModule\ R\ smult\ 0$
 $\langle proof \rangle$

context *RModule*
begin

lemma *trivial-RSubmodule* : *RSubmodule 0*
 $\langle proof \rangle$

lemma *RSubmodule-set-plus* :
 assumes *RSubmodule L RSubmodule N*
 shows *RSubmodule (L + N)*
 $\langle proof \rangle$

lemma *RSubmodule-sum-list* :
 $(\forall N \in set\ Ns. RSubmodule\ N) \Longrightarrow RSubmodule\ (\sum N \leftarrow Ns.\ N)$
 $\langle proof \rangle$

lemma *RSubmodule-inner-dirsum* :
 assumes $(\forall N \in set\ Ns. RSubmodule\ N)$
 shows *RSubmodule ($\bigoplus N \leftarrow Ns.\ N$)*
 $\langle proof \rangle$

lemma *RModule-inner-dirsum* :
 $(\forall N \in set\ Ns. RSubmodule\ N) \Longrightarrow RModule\ R\ smult\ (\bigoplus N \leftarrow Ns.\ N)$
 $\langle proof \rangle$

lemma *SModule-restrict-scalars* :
 assumes *Subring1 S*
 shows *RModule S smult M*
 $\langle proof \rangle$

end

3.2 Linear algebra in modules

3.2.1 Linear combinations: *lincomb*

context *scalar-mult*
begin

definition *lincomb* :: 'r list \Rightarrow 'm list \Rightarrow 'm (**infix** \cdot 70)
where $rs \cdot ms = (\sum (r,m) \leftarrow \text{zip } rs \ ms. r \cdot m)$

Note: *zip* will truncate if lengths of coefficient and vector lists differ.

lemma *lincomb-Nil* : $rs = [] \vee ms = [] \implies rs \cdot ms = 0$
<proof>

lemma *lincomb-singles* : $[a] \cdot [m] = a \cdot m$
<proof>

lemma *lincomb-Cons* : $(r \# rs) \cdot (m \# ms) = r \cdot m + rs \cdot ms$
<proof>

lemma *lincomb-append* :
 $\text{length } rs = \text{length } ms \implies (rs@ss) \cdot (ms@ns) = rs \cdot ms + ss \cdot ns$
<proof>

lemma *lincomb-append-left* :
 $(rs @ ss) \cdot ms = rs \cdot ms + ss \cdot \text{drop } (\text{length } rs) \ ms$
<proof>

lemma *lincomb-append-right* :
 $rs \cdot (ms@ns) = rs \cdot ms + (\text{drop } (\text{length } ms) \ rs) \cdot ns$
<proof>

lemma *lincomb-conv-take-right* : $rs \cdot ms = rs \cdot \text{take } (\text{length } rs) \ ms$
<proof>

end

context *RModule*
begin

lemmas *lincomb-Nil* = *lincomb-Nil*

lemmas *lincomb-Cons* = *lincomb-Cons*

lemma *lincomb-closed* : $\text{set } rs \subseteq R \implies \text{set } ms \subseteq M \implies rs \cdot ms \in M$
<proof>

lemma *smult-lincomb* :
 $\llbracket \text{set } rs \subseteq R; s \in R; \text{set } ms \subseteq M \rrbracket \implies s \cdot (rs \cdot ms) = [s*r. r \leftarrow rs] \cdot ms$
<proof>

lemma *neg-lincomb* :

$$\text{set } rs \subseteq R \implies \text{set } ms \subseteq M \implies - (rs \cdot ms) = [-r. r \leftarrow rs] \cdot ms$$

<proof>

lemma *lincomb-sum-left* :

$$\begin{aligned} & \llbracket \text{set } rs \subseteq R; \text{set } ss \subseteq R; \text{set } ms \subseteq M; \text{length } rs \leq \text{length } ss \rrbracket \\ & \implies [r + s. (r,s) \leftarrow \text{zip } rs \ ss] \cdot ms = rs \cdot ms + (\text{take } (\text{length } rs) \ ss) \cdot ms \end{aligned}$$

<proof>

lemma *lincomb-sum* :

$$\begin{aligned} & \text{assumes } \text{set } rs \subseteq R \ \text{set } ss \subseteq R \ \text{set } ms \subseteq M \ \text{length } rs \leq \text{length } ss \\ & \text{shows } rs \cdot ms + ss \cdot ms \\ & \quad = ([a + b. (a,b) \leftarrow \text{zip } rs \ ss] \ @ \ (\text{drop } (\text{length } rs) \ ss)) \cdot ms \end{aligned}$$

<proof>

lemma *lincomb-diff-left* :

$$\begin{aligned} & \llbracket \text{set } rs \subseteq R; \text{set } ss \subseteq R; \text{set } ms \subseteq M; \text{length } rs \leq \text{length } ss \rrbracket \\ & \implies [r - s. (r,s) \leftarrow \text{zip } rs \ ss] \cdot ms = rs \cdot ms - (\text{take } (\text{length } rs) \ ss) \cdot ms \end{aligned}$$

<proof>

lemma *lincomb-replicate-left* :

$$r \in R \implies \text{set } ms \subseteq M \implies (\text{replicate } k \ r) \cdot ms = r \cdot (\sum m \leftarrow (\text{take } k \ ms). \ m)$$

<proof>

lemma *lincomb-replicate0-left* : $\text{set } ms \subseteq M \implies (\text{replicate } k \ 0) \cdot ms = 0$

<proof>

lemma *lincomb-0coeffs* : $\text{set } ms \subseteq M \implies \forall s \in \text{set } rs. \ s = 0 \implies rs \cdot ms = 0$

<proof>

lemma *delta-scalars-lincomb-eq-nth* :

$$\begin{aligned} & \text{set } ms \subseteq M \implies n < \text{length } ms \\ & \implies ((\text{replicate } (\text{length } ms) \ 0)[n := 1]) \cdot ms = ms!n \end{aligned}$$

<proof>

lemma *lincomb-obtain-same-length-Rcoeffs* :

$$\begin{aligned} & \text{set } rs \subseteq R \implies \text{set } ms \subseteq M \\ & \implies \exists ss. \ \text{set } ss \subseteq R \ \wedge \ \text{length } ss = \text{length } ms \\ & \quad \wedge \ \text{take } (\text{length } rs) \ ss = \text{take } (\text{length } ms) \ rs \ \wedge \ rs \cdot ms = ss \cdot ms \end{aligned}$$

<proof>

lemma *lincomb-concat* :

$$\begin{aligned} & \text{list-all2 } (\lambda rs \ ms. \ \text{length } rs = \text{length } ms) \ rss \ mss \\ & \implies (\text{concat } rss) \cdot (\text{concat } mss) = (\sum (rs,ms) \leftarrow \text{zip } rss \ mss. \ rs \cdot ms) \end{aligned}$$

<proof>

lemma *lincomb-snoc0* : $\text{set } ms \subseteq M \implies (as@[0]) \cdot ms = as \cdot ms$

<proof>

lemma *lincomb-strip-while-0coeffs* :
assumes *set ms* $\subseteq M$
shows (*strip-while* ((=) 0) *as*) $\cdot \cdot$ *ms* = *as* $\cdot \cdot$ *ms*
 \langle *proof* \rangle

end

lemmas (**in** *Module*) *lincomb-obtain-same-length-coeffs* = *lincomb-obtain-same-length-Rcoeffs*
lemmas (**in** *Module*) *lincomb-concat* = *lincomb-concat*

3.2.2 Spanning: *RSpan* and *Span*

context *R-scalar-mult*
begin

primrec *RSpan* :: '*m list* \Rightarrow '*m set*
where *RSpan* [] = 0
| *RSpan* (*m#ms*) = { *r* \cdot *m* | *r*. *r* $\in R$ } + *RSpan ms*

lemma *RSpan-single* : *RSpan* [*m*] = { *r* \cdot *m* | *r*. *r* $\in R$ }
 \langle *proof* \rangle

lemma *RSpan-Cons* : *RSpan* (*m#ms*) = *RSpan* [*m*] + *RSpan ms*
 \langle *proof* \rangle

lemma *in-RSpan-obtain-same-length-coeffs* :
n \in *RSpan ms* \Longrightarrow \exists *rs*. *set rs* $\subseteq R$ \wedge *length rs* = *length ms* \wedge *n* = *rs* $\cdot \cdot$ *ms*
 \langle *proof* \rangle

lemma *in-RSpan-Cons-obtain-same-length-coeffs* :
n \in *RSpan* (*m#ms*) \Longrightarrow \exists *r rs*. *set* (*r#rs*) $\subseteq R$ \wedge *length rs* = *length ms*
 \wedge *n* = *r* \cdot *m* + *rs* $\cdot \cdot$ *ms*
 \langle *proof* \rangle

lemma *RSpanD-lincomb* :
RSpan ms = { *rs* $\cdot \cdot$ *ms* | *rs*. *set rs* $\subseteq R$ \wedge *length rs* = *length ms* }
 \langle *proof* \rangle

lemma *RSpan-append* : *RSpan* (*ms* @ *ns*) = *RSpan ms* + *RSpan ns*
 \langle *proof* \rangle

end

context *scalar-mult*
begin

abbreviation *Span* \equiv *R-scalar-mult.RSpan UNIV smult*

lemmas *Span-append* = *R-scalar-mult.RSpan-append*[*OF R-scalar-mult, of smult*]

lemmas *SpanD-lincomb*
 = *R-scalar-mult.RSpanD-lincomb [OF R-scalar-mult, of smult]*

lemmas *in-Span-obtain-same-length-coeffs*
 = *R-scalar-mult.in-RSpan-obtain-same-length-coeffs[*
 OF R-scalar-mult, of - smult
]

end

context *RModule*
begin

lemma *RSpan-contains-spanset-single* : $m \in M \implies m \in RSpan\ [m]$
 ⟨*proof*⟩

lemma *RSpan-single-nonzero* : $m \in M \implies m \neq 0 \implies RSpan\ [m] \neq 0$
 ⟨*proof*⟩

lemma *Group-RSpan-single* :
assumes $m \in M$
shows $Group\ (RSpan\ [m])$
 ⟨*proof*⟩

lemma *Group-RSpan* : $set\ ms \subseteq M \implies Group\ (RSpan\ ms)$
 ⟨*proof*⟩

lemma *RSpanD-lincomb-arb-len-coeffs* :
 $set\ ms \subseteq M \implies RSpan\ ms = \{ rs \cdot ms \mid rs.\ set\ rs \subseteq R \}$
 ⟨*proof*⟩

lemma *RSpanI-lincomb-arb-len-coeffs* :
 $set\ rs \subseteq R \implies set\ ms \subseteq M \implies rs \cdot ms \in RSpan\ ms$
 ⟨*proof*⟩

lemma *RSpan-contains-RSpans-Cons-left* :
 $set\ ms \subseteq M \implies RSpan\ [m] \subseteq RSpan\ (m\#\ms)$
 ⟨*proof*⟩

lemma *RSpan-contains-RSpans-Cons-right* :
 $m \in M \implies RSpan\ ms \subseteq RSpan\ (m\#\ms)$
 ⟨*proof*⟩

lemma *RSpan-contains-RSpans-append-left* :
 $set\ ns \subseteq M \implies RSpan\ ms \subseteq RSpan\ (ms@ns)$
 ⟨*proof*⟩

lemma *RSpan-contains-spanset* : $set\ ms \subseteq M \implies set\ ms \subseteq RSpan\ ms$
 ⟨*proof*⟩

lemma *RSpan-contains-spanset-append-left* :
 $set\ ms \subseteq M \implies set\ ns \subseteq M \implies set\ ms \subseteq RSpan\ (ms@ns)$
 ⟨proof⟩

lemma *RSpan-contains-spanset-append-right* :
 $set\ ms \subseteq M \implies set\ ns \subseteq M \implies set\ ns \subseteq RSpan\ (ms@ns)$
 ⟨proof⟩

lemma *RSpan-zero-closed* : $set\ ms \subseteq M \implies 0 \in RSpan\ ms$
 ⟨proof⟩

lemma *RSpan-single-closed* : $m \in M \implies RSpan\ [m] \subseteq M$
 ⟨proof⟩

lemma *RSpan-closed* : $set\ ms \subseteq M \implies RSpan\ ms \subseteq M$
 ⟨proof⟩

lemma *RSpan-smult-closed* :
assumes $r \in R\ set\ ms \subseteq M\ n \in RSpan\ ms$
shows $r \cdot n \in RSpan\ ms$
 ⟨proof⟩

lemma *RSpan-add-closed* :
assumes $set\ ms \subseteq M\ n \in RSpan\ ms\ n' \in RSpan\ ms$
shows $n + n' \in RSpan\ ms$
 ⟨proof⟩

lemma *RSpan-lincomb-closed* :
 $\llbracket set\ rs \subseteq R; set\ ms \subseteq M; set\ ns \subseteq RSpan\ ms \rrbracket \implies rs \cdot ns \in RSpan\ ms$
 ⟨proof⟩

lemma *RSpanI* : $set\ ms \subseteq M \implies M \subseteq RSpan\ ms \implies M = RSpan\ ms$
 ⟨proof⟩

lemma *RSpan-contains-RSpan-take* :
 $set\ ms \subseteq M \implies RSpan\ (take\ k\ ms) \subseteq RSpan\ ms$
 ⟨proof⟩

lemma *RSubmodule-RSpan-single* :
assumes $m \in M$
shows $RSubmodule\ (RSpan\ [m])$
 ⟨proof⟩

lemma *RSubmodule-RSpan* : $set\ ms \subseteq M \implies RSubmodule\ (RSpan\ ms)$
 ⟨proof⟩

lemma *RSpan-RSpan-closed* :
 $set\ ms \subseteq M \implies set\ ns \subseteq RSpan\ ms \implies RSpan\ ns \subseteq RSpan\ ms$

<proof>

lemma *spanset-reduce-Cons* :

set ms $\subseteq M \implies m \in RSpan\ ms \implies RSpan\ (m\#\ms) = RSpan\ ms$

<proof>

lemma *RSpan-replace-hd* :

assumes $n \in M$ *set ms* $\subseteq M$ $m \in RSpan\ (n\#\ms)$

shows $RSpan\ (m\#\ms) \subseteq RSpan\ (n\#\ms)$

<proof>

end

lemmas (**in** *scalar-mult*)

Span-Cons = *R-scalar-mult.RSpan-Cons*[*OF R-scalar-mult, of smult*]

context *Module*

begin

lemmas *SpanD-lincomb-arb-len-coeffs* = *RSpanD-lincomb-arb-len-coeffs*

lemmas *SpanI* = *RSpanI*

lemmas *SpanI-lincomb-arb-len-coeffs* = *RSpanI-lincomb-arb-len-coeffs*

lemmas *Span-contains-Spans-Cons-right* = *RSpan-contains-RSpans-Cons-right*

lemmas *Span-contains-spanset* = *RSpan-contains-spanset*

lemmas *Span-contains-spanset-append-left* = *RSpan-contains-spanset-append-left*

lemmas *Span-contains-spanset-append-right* = *RSpan-contains-spanset-append-right*

lemmas *Span-closed* = *RSpan-closed*

lemmas *Span-smult-closed* = *RSpan-smult-closed*

lemmas *Span-contains-Span-take* = *RSpan-contains-RSpan-take*

lemmas *Span-replace-hd* = *RSpan-replace-hd*

lemmas *Submodule-Span* = *RSubmodule-RSpan*

end

3.2.3 Finitely generated modules

context *R-scalar-mult*

begin

abbreviation *R-fingen* $M \equiv (\exists ms. \text{set } ms \subseteq M \wedge RSpan\ ms = M)$

Similar to definition of *card* for finite sets, we default *dim* to 0 if no finite spanning set exists. Note that $RSpan\ [] = 0$ implies that $dim-R\ \{0::'b\} = (0::'a)$.

definition *dim-R* :: $'m\ set \Rightarrow nat$

where $dim-R\ M = (\text{if } R\text{-fingen } M \text{ then } ($

$LEAST\ n. \exists ms. \text{length } ms = n \wedge \text{set } ms \subseteq M \wedge RSpan\ ms = M$

$) \text{ else } 0)$

lemma *dim-R-nonzero* :
assumes *dim-R M > 0*
shows $M \neq 0$
<proof>

end

hide-const *real-vector.dim*
hide-const (**open**) *Real-Vector-Spaces.dim*

abbreviation (**in** *scalar-mult*) *fingen* \equiv *R-scalar-mult.R-fingen UNIV smult*
abbreviation (**in** *scalar-mult*) *dim* \equiv *R-scalar-mult.dim-R UNIV smult*

lemmas (**in** *Module*) *dim-nonzero* = *dim-R-nonzero*

3.2.4 *R*-linear independence

context *R-scalar-mult*
begin

primrec *R-lin-independent* :: '*m list* \Rightarrow *bool* **where**
R-lin-independent-Nil: *R-lin-independent []* = *True* |
R-lin-independent-Cons:
R-lin-independent (m#ms) = (*R-lin-independent ms*
 $\wedge (\forall r \text{ rs. } (\text{set } (r\#\text{rs}) \subseteq R \wedge (r\#\text{rs}) \cdot\cdot (m\#\text{ms}) = 0) \longrightarrow r = 0)$)

lemma *R-lin-independent-ConsI* :
assumes *R-lin-independent ms*
 $\bigwedge r \text{ rs. } \text{set } (r\#\text{rs}) \subseteq R \Longrightarrow (r\#\text{rs}) \cdot\cdot (m\#\text{ms}) = 0 \Longrightarrow r = 0$
shows *R-lin-independent (m#ms)*
<proof>

lemma *R-lin-independent-ConsD1* :
R-lin-independent (m#ms) \Longrightarrow R-lin-independent ms
<proof>

lemma *R-lin-independent-ConsD2* :
 $\llbracket \text{R-lin-independent } (m\#\text{ms}); \text{set } (r\#\text{rs}) \subseteq R; (r\#\text{rs}) \cdot\cdot (m\#\text{ms}) = 0 \rrbracket$
 $\Longrightarrow r = 0$
<proof>

end

context *RModule*
begin

lemma *R-lin-independent-imp-same-scalars* :
 $\llbracket \text{length } rs = \text{length } ss; \text{length } rs \leq \text{length } ms; \text{set } rs \subseteq R; \text{set } ss \subseteq R;$

$set\ ms \subseteq M; R\text{-lin-independent}\ ms; rs \cdot ms = ss \cdot ms \implies rs = ss$
 ⟨proof⟩

lemma *R-lin-independent-obtain-unique-scalars* :

[[$set\ ms \subseteq M; R\text{-lin-independent}\ ms; n \in RSpan\ ms$]]
 $\implies (\exists! rs. set\ rs \subseteq R \wedge length\ rs = length\ ms \wedge n = rs \cdot ms)$
 ⟨proof⟩

lemma *R-lin-independentI-all-scalars* :

$set\ ms \subseteq M \implies$
 $(\forall rs. set\ rs \subseteq R \wedge length\ rs = length\ ms \wedge rs \cdot ms = 0 \longrightarrow set\ rs \subseteq 0)$
 $\implies R\text{-lin-independent}\ ms$
 ⟨proof⟩

lemma *R-lin-independentI-concat-all-scalars* :

defines *eq-len*: $eq\text{-len} \equiv \lambda xs\ ys. length\ xs = length\ ys$
assumes $set\ (concat\ mss) \subseteq M$
 $\bigwedge rss. set\ (concat\ rss) \subseteq R \implies list\text{-all2}\ eq\text{-len}\ rss\ mss$
 $\implies (concat\ rss) \cdot (concat\ mss) = 0 \implies (\forall rs \in set\ rss. set\ rs \subseteq 0)$
shows $R\text{-lin-independent}\ (concat\ mss)$
 ⟨proof⟩

lemma *R-lin-independentD-all-scalars* :

[[$set\ rs \subseteq R; set\ ms \subseteq M; length\ rs \leq length\ ms; R\text{-lin-independent}\ ms;$
 $rs \cdot ms = 0$]] $\implies set\ rs \subseteq 0$
 ⟨proof⟩

lemma *R-lin-independentD-all-scalars-nth* :

assumes $set\ rs \subseteq R\ set\ ms \subseteq M\ R\text{-lin-independent}\ ms\ rs \cdot ms = 0$
 $k < \min (length\ rs) (length\ ms)$
shows $rs!k = 0$
 ⟨proof⟩

lemma *R-lin-dependent-dependence-relation* :

$set\ ms \subseteq M \implies \neg R\text{-lin-independent}\ ms$
 $\implies \exists rs. set\ rs \subseteq R \wedge set\ rs \neq 0 \wedge length\ rs = length\ ms \wedge rs \cdot ms = 0$
 ⟨proof⟩

lemma *R-lin-independent-imp-distinct* :

$set\ ms \subseteq M \implies R\text{-lin-independent}\ ms \implies distinct\ ms$
 ⟨proof⟩

lemma *R-lin-independent-imp-independent-take* :

$set\ ms \subseteq M \implies R\text{-lin-independent}\ ms \implies R\text{-lin-independent}\ (take\ n\ ms)$
 ⟨proof⟩

lemma *R-lin-independent-Cons-imp-independent-RSpans* :

assumes $m \in M\ R\text{-lin-independent}\ (m\#\ms)$
shows $add\text{-independentS}\ [RSpan\ [m], RSpan\ ms]$

<proof>

lemma *hd0-imp-R-lin-dependent* : $\neg R\text{-lin-independent } (0\#ms)$
<proof>

lemma *R-lin-independent-imp-hd-n0* : $R\text{-lin-independent } (m\#ms) \implies m \neq 0$
<proof>

lemma *R-lin-independent-imp-hd-independent-from-RSpan* :
 assumes $m \in M$ *set* $ms \subseteq M$ $R\text{-lin-independent } (m\#ms)$
 shows $m \notin R\text{Span } ms$
<proof>

lemma *R-lin-independent-reduce* :
 assumes $n \in M$
 shows $set\ ms \subseteq M \implies R\text{-lin-independent } (ms @ n \# ns)$
 $\implies R\text{-lin-independent } (ms @ ns)$
<proof>

lemma *R-lin-independent-vs-lincomb0* :
 assumes $set\ (ms@n\#ns) \subseteq M$ $R\text{-lin-independent } (ms @ n \# ns)$
 $set\ (rs@s\#ss) \subseteq R$ $length\ rs = length\ ms$
 $(rs@s\#ss) \cdot (ms@n\#ns) = 0$
 shows $s = 0$
<proof>

lemma *R-lin-independent-append-imp-independent-RSpans* :
 $set\ ms \subseteq M \implies R\text{-lin-independent } (ms@ns)$
 $\implies add\text{-independentS } [R\text{Span } ms, R\text{Span } ns]$
<proof>

end

3.2.5 Linear independence over UNIV

context *scalar-mult*
begin

abbreviation *lin-independent ms*
 $\equiv R\text{-scalar-mult.R-lin-independent UNIV smult } ms$

lemmas *lin-independent-ConsI*
 $= R\text{-scalar-mult.R-lin-independent-ConsI } [OF\ R\text{-scalar-mult, of smult}]$

lemmas *lin-independent-ConsD1*
 $= R\text{-scalar-mult.R-lin-independent-ConsD1 } [OF\ R\text{-scalar-mult, of smult}]$

end

context *Module*

begin

lemmas *lin-independent-imp-independent-take* = *R-lin-independent-imp-independent-take*
lemmas *lin-independent-reduce* = *R-lin-independent-reduce*
lemmas *lin-independent-vs-lincomb0* = *R-lin-independent-vs-lincomb0*
lemmas *lin-dependent-dependence-relation* = *R-lin-dependent-dependence-relation*
lemmas *lin-independent-imp-distinct* = *R-lin-independent-imp-distinct*

lemmas *lin-independent-imp-hd-independent-from-Span*
= *R-lin-independent-imp-hd-independent-from-RSpan*
lemmas *lin-independent-append-imp-independent-Spans*
= *R-lin-independent-append-imp-independent-RSpans*

end

3.2.6 Rank

context *R-scalar-mult*

begin

definition *R-finrank* :: 'm set \Rightarrow bool

where *R-finrank* *M* = $(\exists n. \forall ms. \text{set } ms \subseteq M$
 $\wedge \text{R-lin-independent } ms \longrightarrow \text{length } ms \leq n)$

lemma *R-finrankI* :

$(\bigwedge ms. \text{set } ms \subseteq M \Longrightarrow \text{R-lin-independent } ms \Longrightarrow \text{length } ms \leq n)$
 $\Longrightarrow \text{R-finrank } M$
<proof>

lemma *R-finrankD* :

R-finrank *M* $\Longrightarrow \exists n. \forall ms. \text{set } ms \subseteq M \wedge \text{R-lin-independent } ms$
 $\longrightarrow \text{length } ms \leq n$
<proof>

lemma *submodule-R-finrank* : *R-finrank* *M* $\Longrightarrow N \subseteq M \Longrightarrow \text{R-finrank } N$

<proof>

end

context *scalar-mult*

begin

abbreviation *finrank* :: 'm set \Rightarrow bool

where *finrank* $\equiv \text{R-scalar-mult.R-finrank UNIV smult}$

lemmas *finrankI* = *R-scalar-mult.R-finrankI* [*OF R-scalar-mult, of - smult*]

lemmas *finrankD* = *R-scalar-mult.R-finrankD* [*OF R-scalar-mult, of smult*]

lemmas *submodule-finrank*
= *R-scalar-mult.submodule-R-finrank* [*OF R-scalar-mult, of smult*]

end

3.3 Module homomorphisms

3.3.1 Locales

```
locale RModuleHom = Domain?: RModule R smult M
+ Codomain?: scalar-mult smult'
+ GroupHom?: GroupHom M T
  for R      :: 'r::ring-1 set
  and smult  :: 'r ⇒ 'm::ab-group-add ⇒ 'm (infixr · 70)
  and M      :: 'm set
  and smult' :: 'r ⇒ 'n::ab-group-add ⇒ 'n (infixr ★ 70)
  and T      :: 'm ⇒ 'n
+ assumes R-map:  $\bigwedge r m. r \in R \implies m \in M \implies T (r \cdot m) = r \star T m$ 
```

```
abbreviation (in RModuleHom) lincomb' :: 'r list ⇒ 'n list ⇒ 'n (infix ★ 70)
  where lincomb' ≡ Codomain.lincomb
```

```
lemma (in RModule) RModuleHomI :
  assumes GroupHom M T
     $\bigwedge r m. r \in R \implies m \in M \implies T (r \cdot m) = smult' r (T m)$ 
  shows RModuleHom R smult M smult' T
  ⟨proof⟩
```

```
locale RModuleEnd = RModuleHom R smult M smult T
  for R      :: 'r::ring-1 set
  and smult  :: 'r ⇒ 'm::ab-group-add ⇒ 'm (infixr · 70)
  and M      :: 'm set
  and T      :: 'm ⇒ 'm
+ assumes endomorph:  $ImG \subseteq M$ 
```

```
locale ModuleHom = RModuleHom UNIV smult M smult' T
  for smult  :: 'r::ring-1 ⇒ 'm::ab-group-add ⇒ 'm (infixr · 70)
  and M      :: 'm set
  and smult' :: 'r ⇒ 'n::ab-group-add ⇒ 'n (infixr ★ 70)
  and T      :: 'm ⇒ 'n
```

```
lemmas (in ModuleHom) hom = hom
```

```
lemmas (in Module) ModuleHomI = RModuleHomI[THEN ModuleHom.intro]
```

```
locale ModuleEnd = ModuleHom smult M smult T
  for smult  :: 'r::ring-1 ⇒ 'm::ab-group-add ⇒ 'm (infixr · 70)
  and M      :: 'm set and T :: 'm ⇒ 'm
+ assumes endomorph:  $ImG \subseteq M$ 
```

```
locale RModuleIso = RModuleHom R smult M smult' T
  for R      :: 'r::ring-1 set
```

and $smult :: 'r \Rightarrow 'm::ab\text{-group-add} \Rightarrow 'm$ (**infixr** \cdot 70)
and $M :: 'm$ set
and $smult' :: 'r \Rightarrow 'n::ab\text{-group-add} \Rightarrow 'n$ (**infixr** \star 70)
and $T :: 'm \Rightarrow 'n$
+ **fixes** $N :: 'n$ set
assumes *bijjective: bij-betw T M N*

lemma (**in** *RModule*) *RModuleIsoI* :
assumes *GroupIso M T N*
 $\bigwedge r m. r \in R \implies m \in M \implies T (r \cdot m) = smult' r (T m)$
shows *RModuleIso R smult M smult' T N*
<proof>

3.3.2 Basic facts

lemma (**in** *RModule*) *trivial-RModuleHom* :
 $\forall r \in R. smult' r 0 = 0 \implies RModuleHom R smult M smult' 0$
<proof>

lemma (**in** *RModule*) *RModHom-idhom* : *RModuleHom R smult M smult (id↓M)*
<proof>

context *RModuleHom*
begin

lemmas *additive* = *hom*
lemmas *supp* = *supp*
lemmas *im-zero* = *im-zero*
lemmas *im-diff* = *im-diff*
lemmas *Ker-Im-iff* = *Ker-Im-iff*
lemmas *Ker0-imp-inj-on* = *Ker0-imp-inj-on*

lemma *GroupHom* : *GroupHom M T* *<proof>*

lemma *codomain-smult-zero* : $r \in R \implies r \star 0 = 0$
<proof>

lemma *RSubmodule-Ker* : *Domain.RSubmodule Ker*
<proof>

lemma *RModule-Im* : *RModule R smult' ImG*
<proof>

lemma *im-submodule* :
assumes *RSubmodule N*
shows *RModule.RSubmodule R smult' ImG (T ' N)*
<proof>

lemma *RModHom-composite-left* :

assumes $T \text{ ' } M \subseteq N \text{ RModuleHom } R \text{ smult' } N \text{ smult'' } S$
shows $\text{RModuleHom } R \text{ smult } M \text{ smult'' } (S \circ T)$
 <proof>

lemma *RModuleHom-restrict0-submodule* :
assumes $\text{RSubmodule } N$
shows $\text{RModuleHom } R \text{ smult } N \text{ smult' } (T \downarrow N)$
 <proof>

lemma *distrib-lincomb* :
 $\text{set } rs \subseteq R \implies \text{set } ms \subseteq M \implies T (rs \cdot ms) = rs \cdot \star \text{ map } T \text{ ms}$
 <proof>

lemma *same-image-on-RSpanset-imp-same-hom* :
assumes $\text{RModuleHom } R \text{ smult } M \text{ smult' } S \text{ set } ms \subseteq M$
 $M = \text{Domain.R-scalars.RSpan } ms \forall m \in \text{set } ms. S \text{ m} = T \text{ m}$
shows $S = T$
 <proof>

end

lemma *RSubmodule-eigenspace* :
fixes $\text{smult} :: 'r::\text{ring-1} \Rightarrow 'm::\text{ab-group-add} \Rightarrow 'm \text{ (infixr } \cdot 70)$
assumes $\text{RModHom}: \text{RModuleHom } R \text{ smult } M \text{ smult } T$
and $r: r \in R \wedge s \text{ m}. s \in R \implies m \in M \implies s \cdot r \cdot m = r \cdot s \cdot m$
defines $E: E \equiv \{m \in M. T \text{ m} = r \cdot m\}$
shows $\text{RModule.RSubmodule } R \text{ smult } M E$
 <proof>

3.3.3 Basic facts about endomorphisms

lemma (in *RModule*) *Rmap-endomorph-is-RModuleEnd* :
assumes $\text{grpend}: \text{GroupEnd } M \text{ T}$
and $\text{Rmap} : \wedge r \text{ m}. r \in R \implies m \in M \implies T (r \cdot m) = r \cdot (T \text{ m})$
shows $\text{RModuleEnd } R \text{ smult } M \text{ T}$
 <proof>

lemma (in *RModuleEnd*) *GroupEnd* : $\text{GroupEnd } M \text{ T}$
 <proof>

lemmas (in *RModuleEnd*) *proj-decomp* = $\text{GroupEnd.proj-decomp}[OF \text{ GroupEnd}]$

lemma (in *ModuleEnd*) *RModuleEnd* : $\text{RModuleEnd } UNIV \text{ smult } M \text{ T}$
 <proof>

lemmas (in *ModuleEnd*) *GroupEnd* = $\text{RModuleEnd.GroupEnd}[OF \text{ RModuleEnd}]$

lemma *RModuleEnd-over-UNIV-is-ModuleEnd* :
 $\text{RModuleEnd } UNIV \text{ rsmult } M \text{ T} \implies \text{ModuleEnd } \text{rsmult } M \text{ T}$

<proof>

3.3.4 Basic facts about isomorphisms

context *RModuleIso*

begin

abbreviation $invT \equiv (the\text{-}inv\text{-}into\ M\ T) \downarrow N$

lemma *GroupIso* : *GroupIso* *M* *T* *N*

<proof>

lemmas $ImG = GroupIso.ImG$ [OF *GroupIso*]

lemmas $GroupHom\text{-}inv = GroupIso.inv$ [OF *GroupIso*]

lemmas $invT\text{-}into = GroupIso.invT\text{-}into$ [OF *GroupIso*]

lemmas $T\text{-}invT = GroupIso.T\text{-}invT$ [OF *GroupIso*]

lemmas $invT\text{-}eq = GroupIso.invT\text{-}eq$ [OF *GroupIso*]

lemma *RModuleN* : *RModule* *R* *smult'* *N* *<proof>*

lemma $inv : RModuleIso\ R\ smult'\ N\ smult\ invT\ M$

<proof>

end

3.4 Inner direct sums of RModules

lemma (**in** *RModule*) *RModule-inner-dirsum-el-decomp-Rsmult* :

assumes $\forall N \in set\ Ns.\ RSubmodule\ N\ add\text{-}independentS\ Ns\ r \in R$

$x \in (\bigoplus N \leftarrow Ns.\ N)$

shows $(\bigoplus Ns \leftarrow (r \cdot x)) = [r \cdot m.\ m \leftarrow (\bigoplus Ns \leftarrow x)]$

<proof>

lemma (**in** *RModule*) *RModuleEnd-inner-dirsum-el-decomp-nth* :

assumes $\forall N \in set\ Ns.\ RSubmodule\ N\ add\text{-}independentS\ Ns\ n < length\ Ns$

shows $RModuleEnd\ R\ smult\ (\bigoplus N \leftarrow Ns.\ N)\ (\bigoplus Ns \downarrow n)$

<proof>

4 Vector Spaces

4.1 Locales and basic facts

Here we don't care about being able to switch scalars.

locale *fscalar-mult* = *scalar-mult* *smult*

for $smult :: 'f::field \Rightarrow 'v::ab\text{-}group\text{-}add \Rightarrow 'v$ (**infixr** \cdot 70)

abbreviation (**in** *fscalar-mult*) *findim* \equiv *fingen*

locale *VectorSpace* = *Module* *smult* *V*

```

for smult :: 'f::field  $\Rightarrow$  'v::ab-group-add  $\Rightarrow$  'v (infixr  $\cdot$  70)
and V      :: 'v set

lemmas VectorSpaceI = ModuleI[THEN VectorSpace.intro]

sublocale VectorSpace < fscalar-mult <proof>

locale FinDimVectorSpace = VectorSpace
+ assumes findim: findim V

lemma (in VectorSpace) FinDimVectorSpaceI :
  findim V  $\Longrightarrow$  FinDimVectorSpace ( $\cdot$ ) V
  <proof>

context VectorSpace
begin

abbreviation Subspace :: 'v set  $\Rightarrow$  bool where Subspace  $\equiv$  Submodule

lemma SubspaceD1 : Subspace U  $\Longrightarrow$  VectorSpace smult U
  <proof>

lemmas AbGroup           = AbGroup
lemmas add-closed        = add-closed
lemmas smult-closed      = smult-closed
lemmas one-smult        = one-smult
lemmas smult-assoc       = smult-assoc
lemmas smult-distrib-left = smult-distrib-left
lemmas smult-distrib-right = smult-distrib-right
lemmas zero-closed      = zero-closed
lemmas zero-smult       = zero-smult
lemmas smult-zero       = smult-zero
lemmas smult-lincomb     = smult-lincomb
lemmas smult-distrib-left-diff = smult-distrib-left-diff
lemmas smult-sum-distrib = smult-sum-distrib
lemmas sum-smult-distrib = sum-smult-distrib
lemmas lincomb-sum       = lincomb-sum
lemmas lincomb-closed    = lincomb-closed
lemmas lincomb-concat    = lincomb-concat
lemmas lincomb-replicate0-left = lincomb-replicate0-left
lemmas delta-scalars-lincomb-eq-nth = delta-scalars-lincomb-eq-nth
lemmas SpanI             = SpanI
lemmas Span-closed       = Span-closed
lemmas SpanD-lincomb-arb-len-coeffs = SpanD-lincomb-arb-len-coeffs
lemmas SpanI-lincomb-arb-len-coeffs = SpanI-lincomb-arb-len-coeffs
lemmas in-Span-obtain-same-length-coeffs = in-Span-obtain-same-length-coeffs
lemmas SubspaceI        = SubmoduleI
lemmas subspace-finrank = submodule-finrank

```

lemma *cancel-scalar*: $\llbracket a \neq 0; u \in V; v \in V; a \cdot u = a \cdot v \rrbracket \implies u = v$
 $\langle proof \rangle$

end

4.2 Linear algebra in vector spaces

4.2.1 Linear independence and spanning

context *VectorSpace*

begin

lemmas *Subspace-Span* = *Submodule-Span*
lemmas *lin-independent-Nil* = *R-lin-independent-Nil*
lemmas *lin-independentI-concat-all-scalars* = *R-lin-independentI-concat-all-scalars*
lemmas *lin-independentD-all-scalars* = *R-lin-independentD-all-scalars*
lemmas *lin-independent-obtain-unique-scalars* = *R-lin-independent-obtain-unique-scalars*

lemma *lincomb-Cons-0-imp-in-Span* :
 $\llbracket v \in V; \text{set } vs \subseteq V; a \neq 0; (a \# as) \cdot (v \# vs) = 0 \rrbracket \implies v \in \text{Span } vs$
 $\langle proof \rangle$

lemma *lin-independent-Cons-conditions* :
 $\llbracket v \in V; \text{set } vs \subseteq V; v \notin \text{Span } vs; \text{lin-independent } vs \rrbracket$
 $\implies \text{lin-independent } (v \# vs)$
 $\langle proof \rangle$

lemma *coeff-n0-imp-in-Span-others* :
assumes $v \in V \text{ set } us \subseteq V \text{ set } vs \subseteq V b \neq 0 \text{ length } as = \text{length } us$
 $w = (as @ b \# bs) \cdot (us @ v \# vs)$
shows $v \in \text{Span } (w \# us @ vs)$
 $\langle proof \rangle$

lemma *lin-independent-replace1-by-lincomb* :
assumes $\text{set } us \subseteq V v \in V \text{ set } vs \subseteq V \text{lin-independent } (us @ v \# vs)$
 $\text{length } as = \text{length } us b \neq 0$
shows $\text{lin-independent } ((as @ b \# bs) \cdot (us @ v \# vs)) \# us @ vs$
 $\langle proof \rangle$

lemma *build-lin-independent-seq* :
assumes $us \text{-} V: \text{set } us \subseteq V$
and $\text{indep-us: lin-independent } us$
shows $\exists ws. \text{set } ws \subseteq V \wedge \text{lin-independent } (ws @ us) \wedge (\text{Span } (ws @ us) = V$
 $\vee \text{length } ws = n)$
 $\langle proof \rangle$

end

4.2.2 Basis for a vector space: *basis-for*

abbreviation (in *fscalar-mult*) *basis-for* :: 'v set \Rightarrow 'v list \Rightarrow bool

where *basis-for* $V\ vs \equiv (set\ vs \subseteq V \wedge V = Span\ vs \wedge lin\text{-independent}\ vs)$

context *VectorSpace*

begin

lemma *spanset-contains-basis* :

set vs $\subseteq V \implies \exists us. set\ us \subseteq set\ vs \wedge basis\text{-for}\ (Span\ vs)\ us$

<proof>

lemma *basis-for-Span-ex* : *set vs $\subseteq V \implies \exists us. basis\text{-for}\ (Span\ vs)\ us$*

<proof>

lemma *replace-basis-one-step* :

assumes *closed*: *set vs $\subseteq V$ set us $\subseteq V$ and indep*: *lin-independent (us@vs)*

and *new-w*: *w $\in Span\ (us@vs) - Span\ us$*

shows $\exists xs\ y\ ys. vs = xs\ @\ y\ \#\ ys$

$\wedge basis\text{-for}\ (Span\ (us@vs))\ (w\ \#\ us\ @\ xs\ @\ ys)$

<proof>

lemma *replace-basis* :

assumes *closed*: *set vs $\subseteq V$ and indep-vs*: *lin-independent vs*

shows $\llbracket length\ us \leq length\ vs; set\ us \subseteq Span\ vs; lin\text{-independent}\ us \rrbracket$

$\implies \exists pvs. length\ pvs = length\ vs \wedge set\ pvs = set\ vs$

$\wedge basis\text{-for}\ (Span\ vs)\ (take\ (length\ vs)\ (us\ @\ pvs))$

<proof>

lemma *replace-basis-completely* :

$\llbracket set\ vs \subseteq V; lin\text{-independent}\ vs; length\ us = length\ vs;$

$set\ us \subseteq Span\ vs; lin\text{-independent}\ us \rrbracket \implies basis\text{-for}\ (Span\ vs)\ us$

<proof>

lemma *basis-for-obtain-unique-scalars* :

basis-for V vs $\implies v \in V \implies \exists! as. length\ as = length\ vs \wedge v = as \cdot\ vs$

<proof>

lemma *add-unique-scalars* :

assumes *vs*: *basis-for V vs* **and** *v*: *v $\in V$ and v'*: *v' $\in V$*

defines *as*: *as $\equiv (THE\ ds. length\ ds = length\ vs \wedge v = ds \cdot\ vs)$*

and *bs*: *bs $\equiv (THE\ ds. length\ ds = length\ vs \wedge v' = ds \cdot\ vs)$*

and *cs*: *cs $\equiv (THE\ ds. length\ ds = length\ vs \wedge v+v' = ds \cdot\ vs)$*

shows *cs = [a+b. (a,b) \leftarrow zip as bs]*

<proof>

lemma *smult-unique-scalars* :

fixes *a*: 'f

assumes *vs*: *basis-for V vs* **and** *v*: *v $\in V$*

defines *as*: *as $\equiv (THE\ cs. length\ cs = length\ vs \wedge v = cs \cdot\ vs)$*

and $bs: bs \equiv (THE\ cs.\ length\ cs = length\ vs \wedge a \cdot v = cs \cdot\cdot\ vs)$
shows $bs = map\ ((*)\ a)\ as$
 $\langle proof \rangle$

lemma *max-lin-independent-set-in-Span* :
assumes $set\ vs \subseteq V\ set\ us \subseteq Span\ vs\ lin\text{-}independent\ us$
shows $length\ us \leq length\ vs$
 $\langle proof \rangle$

lemma *finrank-Span* : $set\ vs \subseteq V \implies finrank\ (Span\ vs)$
 $\langle proof \rangle$

end

4.3 Finite dimensional spaces

context *VectorSpace*
begin

lemma *dim-eq-size-basis* : $basis\text{-}for\ V\ vs \implies length\ vs = dim\ V$
 $\langle proof \rangle$

lemma *finrank-imp-findim* :
assumes $finrank\ V$
shows $findim\ V$
 $\langle proof \rangle$

lemma *subspace-Span-is-findim* :
 $\llbracket set\ vs \subseteq V; Subspace\ W; W \subseteq Span\ vs \rrbracket \implies findim\ W$
 $\langle proof \rangle$

end

context *FinDimVectorSpace*
begin

lemma *Subspace-is-findim* : $Subspace\ U \implies findim\ U$
 $\langle proof \rangle$

lemma *basis-ex* : $\exists\ vs.\ basis\text{-}for\ V\ vs$
 $\langle proof \rangle$

lemma *lin-independent-length-le-dim* :
 $set\ us \subseteq V \implies lin\text{-}independent\ us \implies length\ us \leq dim\ V$
 $\langle proof \rangle$

lemma *too-long-lin-dependent* :
 $set\ us \subseteq V \implies length\ us > dim\ V \implies \neg\ lin\text{-}independent\ us$
 $\langle proof \rangle$

lemma *extend-lin-independent-to-basis* :
assumes *set us* $\subseteq V$ *lin-independent us*
shows \exists *vs. basis-for V (vs @ us)*
 \langle *proof* \rangle

lemma *extend-Subspace-basis* :
 $U \subseteq V \implies$ *basis-for U us* $\implies \exists$ *vs. basis-for V (vs@us)*
 \langle *proof* \rangle

lemma *Subspace-dim-le* :
assumes *Subspace U*
shows $\dim U \leq \dim V$
 \langle *proof* \rangle

lemma *Subspace-eqdim-imp-equal* :
assumes *Subspace U* $\dim U = \dim V$
shows $U = V$
 \langle *proof* \rangle

lemma *Subspace-dim-lt* : *Subspace U* $\implies U \neq V \implies \dim U < \dim V$
 \langle *proof* \rangle

lemma *semisimple* :
assumes *Subspace U*
shows $\exists W. \text{Subspace } W \wedge (V = W \oplus U)$
 \langle *proof* \rangle

end

4.4 Vector space homomorphisms

4.4.1 Locales

locale *VectorSpaceHom = ModuleHom smult V smult' T*
for *smult* $:: 'f::\text{field} \Rightarrow 'v::\text{ab-group-add} \Rightarrow 'v$ (**infixr** \cdot 70)
and *V* $:: 'v$ *set*
and *smult'* $:: 'f \Rightarrow 'w::\text{ab-group-add} \Rightarrow 'w$ (**infixr** \star 70)
and *T* $:: 'v \Rightarrow 'w$

sublocale *VectorSpaceHom* $<$ *VectorSpace* \langle *proof* \rangle

lemmas (**in** *VectorSpace*)
VectorSpaceHomI = ModuleHomI[THEN VectorSpaceHom.intro]

lemma (**in** *VectorSpace*) *VectorSpaceHomI-fromaxioms* :
assumes $\bigwedge g g'. g \in V \implies g' \in V \implies T (g + g') = T g + T g'$
 $\text{supp } T \subseteq V$
 $\bigwedge r m. r \in UNIV \implies m \in V \implies T (r \cdot m) = \text{smult}' r (T m)$
shows *VectorSpaceHom smult V smult' T*

<proof>

locale *VectorSpaceEnd* = *VectorSpaceHom smult V smult T*
 for *smult* :: 'f::field \Rightarrow 'v::ab-group-add \Rightarrow 'v (**infixr** · 70)
 and *V* :: 'v set
 and *T* :: 'v \Rightarrow 'v
+ **assumes** *endomorph*: $ImG \subseteq V$

abbreviation (**in** *VectorSpace*) *VEnd* \equiv *VectorSpaceEnd smult V*

lemma *VectorSpaceEndI* :
 fixes *smult* :: 'f::field \Rightarrow 'v::ab-group-add \Rightarrow 'v
 assumes *VectorSpaceHom smult V smult T T ' V \subseteq V*
 shows *VectorSpaceEnd smult V T*
 <proof>

lemma (**in** *VectorSpaceEnd*) *VectorSpaceHom*: *VectorSpaceHom smult V smult T*
 <proof>

lemma (**in** *VectorSpaceEnd*) *ModuleEnd* : *ModuleEnd smult V T*
 <proof>

locale *VectorSpaceIso* = *VectorSpaceHom smult V smult' T*
 for *smult* :: 'f::field \Rightarrow 'v::ab-group-add \Rightarrow 'v (**infixr** · 70)
 and *V* :: 'v set
 and *smult'* :: 'f \Rightarrow 'w::ab-group-add \Rightarrow 'w (**infixr** ★ 70)
 and *T* :: 'v \Rightarrow 'w
+ **fixes** *W* :: 'w set
 assumes *bijjective*: *bij-betw T V W*

abbreviation (**in** *VectorSpace*) *isomorphic* ::
 ('f \Rightarrow 'w::ab-group-add \Rightarrow 'w) \Rightarrow 'w set \Rightarrow bool
 where *isomorphic smult' W* \equiv ($\exists T. \text{VectorSpaceIso smult V smult' T W}$)

4.4.2 Basic facts

lemma (**in** *VectorSpace*) *trivial-VectorSpaceHom* :
 ($\bigwedge a. \text{smult}' a 0 = 0$) \implies *VectorSpaceHom smult V smult' 0*
 <proof>

lemma (**in** *VectorSpace*) *VectorSpaceHom-idhom* :
 VectorSpaceHom smult V smult (id \downarrow V)
 <proof>

context *VectorSpaceHom*
begin

lemmas *hom* = *hom*
lemmas *supp* = *supp*

```

lemmas f-map           = R-map
lemmas im-zero        = im-zero
lemmas im-sum-list-prod = im-sum-list-prod
lemmas additive       = additive
lemmas GroupHom       = GroupHom
lemmas distrib-lincomb = distrib-lincomb

lemmas same-image-on-spanset-imp-same-hom
  = same-image-on-RSpanset-imp-same-hom[
    OF ModuleHom.axioms(1), OF VectorSpaceHom.axioms(1)
  ]

```

```

lemma VectorSpace-Im : VectorSpace smult' ImG
  ⟨proof⟩

```

```

lemma VectorSpaceHom-scalar-mul :
  VectorSpaceHom smult V smult' (λv. a ★ T v)
  ⟨proof⟩

```

```

lemma VectorSpaceHom-composite-left :
  assumes ImG ⊆ W VectorSpaceHom smult' W smult'' S
  shows VectorSpaceHom smult V smult'' (S ◦ T)
  ⟨proof⟩

```

```

lemma findim-domain-findim-image :
  assumes findim V
  shows fscalar-mult.findim smult' ImG
  ⟨proof⟩

```

end

```

lemma (in VectorSpace) basis-im-defines-hom :
  fixes smult' :: 'f ⇒ 'w::ab-group-add ⇒ 'w (infixr ★ 70)
  and lincomb' :: 'f list ⇒ 'w list ⇒ 'w (infixr •★ 70)
  defines lincomb' : lincomb' ≡ scalar-mult.lincomb smult'
  assumes VSpW : VectorSpace smult' W
  and basisV : basis-for V vs
  and basisV-im : set ws ⊆ W length ws = length vs
  shows ∃! T. VectorSpaceHom smult V smult' T ∧ map T vs = ws
  ⟨proof⟩

```

4.4.3 Hom-sets

```

definition VectorSpaceHomSet ::
  ('f::field ⇒ 'v::ab-group-add ⇒ 'v) ⇒ 'v set ⇒ ('f ⇒ 'w::ab-group-add ⇒ 'w)
  ⇒ 'w set ⇒ ('v ⇒ 'w) set
  where VectorSpaceHomSet fsmult V fsmult' W
    ≡ {T. VectorSpaceHom fsmult V fsmult' T} ∩ {T. T ' V ⊆ W}

```

abbreviation (in *VectorSpace*) *VectorSpaceEndSet* $\equiv \{S. VEnd\ S\}$

lemma *VectorSpaceHomSetI* :

VectorSpaceHom fsmult V fsmult' T $\implies T \text{ ' } V \subseteq W$
 $\implies T \in \text{VectorSpaceHomSet fsmult V fsmult' } W$
 ⟨proof⟩

lemma *VectorSpaceHomSetD-VectorSpaceHom* :

T $\in \text{VectorSpaceHomSet fsmult V fsmult' } N$
 $\implies \text{VectorSpaceHom fsmult V fsmult' } T$
 ⟨proof⟩

lemma *VectorSpaceHomSetD-Im* :

T $\in \text{VectorSpaceHomSet fsmult V fsmult' } W \implies T \text{ ' } V \subseteq W$
 ⟨proof⟩

context *VectorSpace*

begin

lemma *VectorSpaceHomSet-is-fmaps-in-GroupHomSet* :

fixes *smult' :: 'f \Rightarrow 'w::ab-group-add \Rightarrow 'w* (**infixr** \star 70)
shows *VectorSpaceHomSet smult V smult' W*
 $= (\text{GroupHomSet } V\ W) \cap \{T. \forall a. \forall v \in V. T (a \cdot v) = a \star (T\ v)\}$
 ⟨proof⟩

lemma *Group-VectorSpaceHomSet* :

fixes *smult' :: 'f \Rightarrow 'w::ab-group-add \Rightarrow 'w* (**infixr** \star 70)
assumes *VectorSpace smult' W*
shows *Group (VectorSpaceHomSet smult V smult' W)*
 ⟨proof⟩

lemma *VectorSpace-VectorSpaceHomSet* :

fixes *smult' :: 'f \Rightarrow 'w::ab-group-add \Rightarrow 'w* (**infixr** \star 70)
and *hom-smult :: 'f \Rightarrow ('v \Rightarrow 'w) \Rightarrow ('v \Rightarrow 'w)* (**infixr** \star 70)
defines *hom-smult: hom-smult $\equiv \lambda a\ T\ v. a \star T\ v$*
assumes *VSpW: VectorSpace smult' W*
shows *VectorSpace hom-smult (VectorSpaceHomSet smult V smult' W)*
 ⟨proof⟩

end

4.4.4 Basic facts about endomorphisms

lemma *ModuleEnd-over-field-is-VectorSpaceEnd* :

fixes *smult :: 'f::field \Rightarrow 'v::ab-group-add \Rightarrow 'v*
assumes *ModuleEnd smult V T*
shows *VectorSpaceEnd smult V T*
 ⟨proof⟩

context *VectorSpace*
begin

lemmas *VectorSpaceEnd-inner-dirsum-el-decomp-nth* =
RModuleEnd-inner-dirsum-el-decomp-nth[
 THEN RModuleEnd-over-UNIV-is-ModuleEnd,
 THEN ModuleEnd-over-field-is-VectorSpaceEnd
]

abbreviation *end-smult* :: '*f* ⇒ ('*v* ⇒ '*v*) ⇒ ('*v* ⇒ '*v*) (**infixr** .. 70)
where *a* .. *T* ≡ (λ*v*. *a* · *T* *v*)

abbreviation *end-lincomb*
:: '*f* *list* ⇒ (('v ⇒ '*v*) *list*) ⇒ ('*v* ⇒ '*v*) (**infixr** ... 70)
where *end-lincomb* ≡ *scalar-mult.lincomb end-smult*

lemma *end-smult0*: *a* .. 0 = 0
⟨*proof*⟩

lemma *end-0smult*: *range T* ⊆ *V* ⇒ 0 .. *T* = 0
⟨*proof*⟩

lemma *end-smult-distrib-left* :
assumes *range S* ⊆ *V* *range T* ⊆ *V*
shows *a* .. (*S* + *T*) = *a* .. *S* + *a* .. *T*
⟨*proof*⟩

lemma *end-smult-distrib-right* :
assumes *range T* ⊆ *V*
shows (*a*+*b*) .. *T* = *a* .. *T* + *b* .. *T*
⟨*proof*⟩

lemma *end-smult-assoc* :
assumes *range T* ⊆ *V*
shows *a* .. *b* .. *T* = (*a* * *b*) .. *T*
⟨*proof*⟩

lemma *end-smult-comp-comm-left* : (*a* .. *T*) ∘ *S* = *a* .. (*T* ∘ *S*)
⟨*proof*⟩

lemma *end-idhom* : *VEnd* (*id* ↓ *V*)
⟨*proof*⟩

lemma *VectorSpaceEndSet-is-VectorSpaceHomSet* :
VectorSpaceHomSet smult V smult V = {*T*. *VEnd T*}
⟨*proof*⟩

lemma *VectorSpace-VectorSpaceEndSet* : *VectorSpace end-smult VectorSpaceEnd-Set*

$\langle proof \rangle$
end
context *VectorSpaceEnd*
begin
lemmas *f-map* = *R-map*
lemmas *supp* = *supp*
lemmas *GroupEnd* = *ModuleEnd.GroupEnd[OF ModuleEnd]*
lemmas *idhom-left* = *idhom-left*
lemmas *range* = *GroupEnd.range[OF GroupEnd]*
lemmas *Ker0-imp-inj-on* = *Ker0-imp-inj-on*
lemmas *inj-on-imp-Ker0* = *inj-on-imp-Ker0*
lemmas *nonzero-Ker-el-imp-n-inj* = *nonzero-Ker-el-imp-n-inj*
lemmas *VectorSpaceHom-composite-left*
= *VectorSpaceHom-composite-left[OF endomorph]*

lemma *in-VEndSet* : $T \in \text{VectorSpaceEndSet}$
 $\langle proof \rangle$

lemma *end-smult-comp-comm-right* :
 $\text{range } S \subseteq V \implies T \circ (a \cdot S) = a \cdot (T \circ S)$
 $\langle proof \rangle$

lemma *VEnd-end-smult-VEnd* : $V\text{End } (a \cdot T)$
 $\langle proof \rangle$

lemma *VEnd-composite-left* :
assumes $V\text{End } S$
shows $V\text{End } (S \circ T)$
 $\langle proof \rangle$

lemma *VEnd-composite-right* : $V\text{End } S \implies V\text{End } (T \circ S)$
 $\langle proof \rangle$

end

lemma (**in** *VectorSpace*) *inj-comp-end* :
assumes $V\text{End } S \text{ inj-on } S \ V \ V\text{End } T \text{ inj-on } T \ V$
shows $\text{inj-on } (S \circ T) \ V$
 $\langle proof \rangle$

lemma (**in** *VectorSpace*) *n-inj-comp-end* :
 $\llbracket V\text{End } S; V\text{End } T; \neg \text{inj-on } (S \circ T) \ V \rrbracket \implies \neg \text{inj-on } S \ V \vee \neg \text{inj-on } T \ V$
 $\langle proof \rangle$

4.4.5 Polynomials of endomorphisms

context *VectorSpaceEnd*

begin

primrec *endpow* :: *nat* \Rightarrow (*v* \Rightarrow *v*)
where *endpow0*: *endpow* 0 = *id* \downarrow *V*
| *endpowSuc*: *endpow* (*Suc* *n*) = *T* \circ (*endpow* *n*)

definition *polymap* :: *'f poly* \Rightarrow (*v* \Rightarrow *v*)
where *polymap* *p* \equiv (*coeffs* *p*) \cdots (*map* *endpow* [0..*Suc* (*degree* *p*)])

lemma *VEnd-endpow* : *VEnd* (*endpow* *n*)
 \langle *proof* \rangle

lemma *endpow-list-apply-closed* :
v \in *V* \Longrightarrow *set* (*map* (λ *S*. *S* *v*) (*map* *endpow* [0..*k*])) \subseteq *V*
 \langle *proof* \rangle

lemma *map-endpow-Suc* :
map *endpow* [0..*Suc* *n*] = (*id* \downarrow *V*) $\#$ *map* (\circ) *T* (*map* *endpow* [0..*n*])
 \langle *proof* \rangle

lemma *T-endpow-list-apply-commute* :
map *T* (*map* (λ *S*. *S* *v*) (*map* *endpow* [0..*n*]))
= *map* (λ *S*. *S* *v*) (*map* (\circ) *T*) (*map* *endpow* [0..*n*])
 \langle *proof* \rangle

lemma *polymap0* : *polymap* 0 = 0
 \langle *proof* \rangle

lemma *VEnd-polymap* : *VEnd* (*polymap* *p*)
 \langle *proof* \rangle

lemma *polymap-pCons* : *polymap* (*pCons* *a* *p*) = *a* \cdot (*id* \downarrow *V*) + (*T* \circ (*polymap* *p*))
 \langle *proof* \rangle

lemma *polymap-plus* : *polymap* (*p* + *q*) = *polymap* *p* + *polymap* *q*
 \langle *proof* \rangle

lemma *polymap-polysmult* : *polymap* (*Polynomial.smult* *a* *p*) = *a* \cdot *polymap* *p*
 \langle *proof* \rangle

lemma *polymap-times* : *polymap* (*p* * *q*) = (*polymap* *p*) \circ (*polymap* *q*)
 \langle *proof* \rangle

lemma *polymap-apply* :
assumes *v* \in *V*
shows *polymap* *p* *v* = (*coeffs* *p*)
 \cdots (*map* (λ *S*. *S* *v*) (*map* *endpow* [0..*Suc* (*degree* *p*)]))

<proof>

lemma *polymap-apply-linear* : $v \in V \implies \text{polymap } [-c, 1:] v = T v - c \cdot v$
<proof>

lemma *polymap-const-inj* :
 assumes *degree p = 0 p ≠ 0*
 shows *inj-on (polymap p) V*
<proof>

lemma *n-inj-polymap-times* :
 $\neg \text{inj-on } (\text{polymap } (p * q)) V$
 $\implies \neg \text{inj-on } (\text{polymap } p) V \vee \neg \text{inj-on } (\text{polymap } q) V$
<proof>

In the following lemma, $[-c, 1::'a:]$ is the linear polynomial $x - c$.

lemma *n-inj-polymap-findlinear* :
 assumes *alg-closed: $\bigwedge p::'f \text{ poly. degree } p > 0 \implies \exists c. \text{poly } p c = 0$*
 shows $p \neq 0 \implies \neg \text{inj-on } (\text{polymap } p) V$
 $\implies \exists c. \neg \text{inj-on } (\text{polymap } [-c, 1:]) V$
<proof>

end

4.4.6 Existence of eigenvectors of endomorphisms of finite-dimensional vector spaces

lemma (in *FinDimVectorSpace*) *endomorph-has-eigenvector* :
 assumes *alg-closed: $\bigwedge p::'a \text{ poly. degree } p > 0 \implies \exists c. \text{poly } p c = 0$*
 and *dim : dim V > 0*
 and *endo : VectorSpaceEnd smult V T*
 shows $\exists c u. u \in V \wedge u \neq 0 \wedge T u = c \cdot u$
<proof>

5 Modules Over a Group Ring

5.1 Almost-everywhere-zero functions as scalars

locale *aezfun-scalar-mult = scalar-mult smult*
 for *smult* ::
 (*'r::ring-1, 'g::group-add*) *aezfun* \Rightarrow *'v::ab-group-add* \Rightarrow *'v* (**infixr** \cdot 70)
begin

definition *fsmult* :: *'r* \Rightarrow *'v* \Rightarrow *'v* (**infixr** $\#$ 70) **where** $a \# v \equiv (a \delta \delta 0) \cdot v$

abbreviation *flincomb* :: *'r list* \Rightarrow *'v list* \Rightarrow *'v* (**infixr** $\cdot \#$ 70)

where $as \cdot \# vs \equiv \text{scalar-mult.lincomb fsmult as vs}$

abbreviation *f-lin-independent* :: *'v list* \Rightarrow *bool*

where *f-lin-independent* $\equiv \text{scalar-mult.lin-independent fsmult}$

abbreviation *fSpan* :: *'v list* \Rightarrow *'v set* **where** *fSpan* $\equiv \text{scalar-mult.Span fsmult}$

definition $Gmult :: 'g \Rightarrow 'v \Rightarrow 'v$ (**infixr** $*$ 70) **where** $g * v \equiv (1 \ \delta\delta \ g) \cdot v$

lemmas $R\text{-scalar-mult} = R\text{-scalar-mult}$

lemma $fsmultD : a \# \cdot v = (a \ \delta\delta \ 0) \cdot v$
 $\langle proof \rangle$

lemma $GmultD : g * v = (1 \ \delta\delta \ g) \cdot v$
 $\langle proof \rangle$

definition $negGorbit\text{-list} :: 'g \text{ list} \Rightarrow ('a \Rightarrow 'v) \Rightarrow 'a \text{ list} \Rightarrow 'v \text{ list list}$
where $negGorbit\text{-list} \ gs \ T \ as \equiv \text{map} (\lambda g. \text{map} (Gmult \ (-g) \circ T) \ as) \ gs$

lemma $negGorbit\text{-Cons} :$
 $negGorbit\text{-list} (g\#gs) \ T \ as$
 $= (\text{map} (Gmult \ (-g) \circ T) \ as) \# \ negGorbit\text{-list} \ gs \ T \ as$
 $\langle proof \rangle$

lemma $length\text{-negGorbit-list} : length (negGorbit\text{-list} \ gs \ T \ as) = length \ gs$
 $\langle proof \rangle$

lemma $length\text{-negGorbit-list-sublist} :$
 $fs \in \text{set} (negGorbit\text{-list} \ gs \ T \ as) \implies length \ fs = length \ as$
 $\langle proof \rangle$

lemma $length\text{-concat-negGorbit-list} :$
 $length (\text{concat} (negGorbit\text{-list} \ gs \ T \ as)) = (length \ gs) * (length \ as)$
 $\langle proof \rangle$

lemma $negGorbit\text{-list-nth} :$
 $\bigwedge i. i < length \ gs \implies (negGorbit\text{-list} \ gs \ T \ as)!i = \text{map} (Gmult \ (-gs!i) \circ T) \ as$
 $\langle proof \rangle$

end

5.2 Locale and basic facts

locale $FGModule = ActingGroup? : Group \ G$
 $+ \ FGMod? : RModule \ ActingGroup.group\text{-ring} \ smult \ V$
for $G :: 'g :: group\text{-add} \ \text{set}$
and $smult :: ('f :: field, 'g) \ aezfun \Rightarrow 'v :: ab\text{-group-add} \Rightarrow 'v$ (**infixr** \cdot 70)
and $V :: 'v \ \text{set}$

sublocale $FGModule < aezfun\text{-scalar-mult} \langle proof \rangle$

lemma (**in** $Group$) $trivial\text{-FGModule} :$
fixes $smult :: ('f :: field, 'g) \ aezfun \Rightarrow 'v :: ab\text{-group-add} \Rightarrow 'v$
assumes $smult\text{-zero} : \forall a \in group\text{-ring}. smult \ a \ (0 :: 'v) = 0$
shows $FGModule \ G \ smult \ (0 :: 'v \ \text{set})$

<proof>

context *FGModule*
begin

abbreviation *FG* :: (*f, 'g*) *aezfun set* **where** *FG* \equiv *ActingGroup.group-ring*

abbreviation *FGSubmodule* \equiv *RSubmodule*

abbreviation *FG-proj* \equiv *ActingGroup.RG-proj*

lemma *GroupG*: *Group G* *<proof>*

lemmas *zero-closed* = *zero-closed*

lemmas *neg-closed* = *neg-closed*

lemmas *diff-closed* = *diff-closed*

lemmas *zero-smult* = *zero-smult*

lemmas *smult-zero* = *smult-zero*

lemmas *AbGroup* = *AbGroup*

lemmas *sum-closed* = *AbGroup.sum-closed*[*OF AbGroup*]

lemmas *FGSubmoduleI* = *RSubmoduleI*

lemmas *FG-proj-mult-leftdelta* = *ActingGroup.RG-proj-mult-leftdelta*

lemmas *FG-proj-mult-right* = *ActingGroup.RG-proj-mult-right*

lemmas *FG-el-decomp* = *ActingGroup.RG-el-decomp-aezdeltafun*

lemma *FG-n0*: *FG \neq 0* *<proof>*

lemma *FG-proj-in-FG* : *FG-proj x \in FG*
<proof>

lemma *FG-fddg-closed* : *g \in G \implies a $\delta\delta$ g \in FG*
<proof>

lemma *FG-fdd0-closed* : *a $\delta\delta$ 0 \in FG*
<proof>

lemma *Gmult-closed* : *g \in G \implies v \in V \implies g \ast v \in V*
<proof>

lemma *map-Gmult-closed* :
g \in G \implies set vs \subseteq V \implies set (map ((\ast) g) vs) \subseteq V
<proof>

lemma *Gmult0* :
assumes *v \in V*
shows *0 \ast v = v*
<proof>

lemma *Gmult-assoc* :
assumes *g \in G h \in G v \in V*
shows *g \ast h \ast v = (g + h) \ast v*

<proof>

lemma *Gmult-distrib-left* :

$\llbracket g \in G; v \in V; v' \in V \rrbracket \implies g * \cdot (v + v') = g * \cdot v + g * \cdot v'$

<proof>

lemma *neg-Gmult* : $g \in G \implies v \in V \implies g * \cdot (-v) = -(g * \cdot v)$

<proof>

lemma *Gmult-neg-left* : $g \in G \implies v \in V \implies (-g) * \cdot g * \cdot v = v$

<proof>

lemma *fddg-smult-decomp* : $g \in G \implies v \in V \implies (f \delta \delta g) \cdot v = f \# \cdot g * \cdot v$

<proof>

lemma *sum-list-aezdeltafun-smult-distrib* :

assumes $v \in V$ *set* $(\text{map } \text{snd } \text{fgs}) \subseteq G$

shows $(\sum (f,g) \leftarrow \text{fgs}. f \delta \delta g) \cdot v = (\sum (f,g) \leftarrow \text{fgs}. f \# \cdot g * \cdot v)$

<proof>

abbreviation *GSubspace* \equiv *RSubmodule*

abbreviation *GSpan* \equiv *RSpan*

abbreviation *G-fingen* \equiv *R-fingen*

lemma *GSubspaceI* : *FGModule* G *smult* $U \implies U \subseteq V \implies$ *GSubspace* U

<proof>

lemma *GSubspace-is-FGModule* :

assumes *GSubspace* U

shows *FGModule* G *smult* U

<proof>

lemma *restriction-to-subgroup-is-module* :

fixes $H :: 'g$ *set*

assumes *subgrp*: *Group.Subgroup* G H

shows *FGModule* H *smult* V

<proof>

lemma *negGorbit-list-V* :

assumes $\text{set } \text{gs} \subseteq G$ $T \text{ ' } (\text{set } \text{as}) \subseteq V$

shows $\text{set } (\text{concat } (\text{negGorbit-list } \text{gs } T \text{ as})) \subseteq V$

<proof>

lemma *negGorbit-list-Cons0* :

$T \text{ ' } (\text{set } \text{as}) \subseteq V$

$\implies \text{negGorbit-list } (0 \# \text{gs}) T \text{ as} = (\text{map } T \text{ as}) \# (\text{negGorbit-list } \text{gs } T \text{ as})$

<proof>

end

5.3 Modules over a group ring as a vector spaces

context *FGModule*

begin

lemma *fVectorSpace* : *VectorSpace fsmult V*

<proof>

abbreviation *fSubspace* \equiv *VectorSpace.Subspace fsmult V*

abbreviation *fbasis-for* \equiv *fscalar-mult.basis-for fsmult*

abbreviation *fdim* \equiv *scalar-mult.dim fsmult V*

lemma *VectorSpace-fSubspace* : *fSubspace W \implies VectorSpace fsmult W*

<proof>

lemma *fsmult-closed* : *v \in V \implies a \cdot v \in V*

<proof>

lemmas *one-fsmult* [simp] = *VectorSpace.one-smult* [OF *fVectorSpace*]

lemmas *fsmult-assoc* [simp] = *VectorSpace.smult-assoc* [OF *fVectorSpace*]

lemmas *fsmult-zero* [simp] = *VectorSpace.smult-zero* [OF *fVectorSpace*]

lemmas *fsmult-distrib-left* [simp] = *VectorSpace.smult-distrib-left*
[OF *fVectorSpace*]

lemmas *flincomb-closed* = *VectorSpace.lincomb-closed* [OF *fVectorSpace*]

lemmas *fsmult-sum-distrib* = *VectorSpace.smult-sum-distrib* [OF *fVectorSpace*]

lemmas *sum-fsmult-distrib* = *VectorSpace.sum-smult-distrib* [OF *fVectorSpace*]

lemmas *flincomb-concat* = *VectorSpace.lincomb-concat* [OF *fVectorSpace*]

lemmas *fSpan-closed* = *VectorSpace.Span-closed* [OF *fVectorSpace*]

lemmas *flin-independentD-all-scalars*

= *VectorSpace.lin-independentD-all-scalars* [OF *fVectorSpace*]

lemmas *in-fSpan-obtain-same-length-coeffs*

= *VectorSpace.in-Span-obtain-same-length-coeffs* [OF *fVectorSpace*]

lemma *fsmult-smult-comm* : *r \in FG \implies v \in V \implies a \cdot r \cdot v = r \cdot a \cdot v*

<proof>

lemma *fsmult-Gmult-comm* : *g \in G \implies v \in V \implies a \cdot g \cdot v = g \cdot a \cdot v*

<proof>

lemma *Gmult-flincomb-comm* :

assumes *g \in G set vs \subseteq V*

shows *g \cdot as \cdot vs = as \cdot (map (Gmult g) vs)*

<proof>

lemma *GSubspace-is-Subspace* :

GSubspace U \implies VectorSpace.Subspace fsmult V U

<proof>

end

5.4 Homomorphisms of modules over a group ring

5.4.1 Locales

```

locale FGModuleHom = ActingGroup?: Group G
+ RModHom?: RModuleHom ActingGroup.group-ring smult V smult' T
  for G      :: 'g::group-add set
  and smult :: ('f::field, 'g) aezfun  $\Rightarrow$  'v::ab-group-add  $\Rightarrow$  'v (infixr · 70)
  and V      :: 'v set
  and smult' :: ('f, 'g) aezfun  $\Rightarrow$  'w::ab-group-add  $\Rightarrow$  'w (infixr ★ 70)
  and T      :: 'v  $\Rightarrow$  'w

```

```

sublocale FGModuleHom < FGModule ⟨proof⟩

```

```

lemma (in FGModule) FGModuleHomI-fromaxioms :
  assumes  $\bigwedge v v'. v \in V \Longrightarrow v' \in V \Longrightarrow T (v + v') = T v + T v'$ 
     $\text{supp } T \subseteq V \bigwedge r m. r \in FG \Longrightarrow m \in V \Longrightarrow T (r \cdot m) = \text{smult}' r (T m)$ 
  shows FGModuleHom G smult V smult' T
  ⟨proof⟩

```

```

locale FGModuleEnd = FGModuleHom G smult V smult T
  for G      :: 'g::group-add set
  and FG     :: ('f::field, 'g) aezfun set
  and smult :: ('f, 'g) aezfun  $\Rightarrow$  'v::ab-group-add  $\Rightarrow$  'v (infixr · 70)
  and V      :: 'v set
  and T      :: 'v  $\Rightarrow$  'v
+ assumes endomorph:  $\text{Im } G \subseteq V$ 

```

```

locale FGModuleIso = FGModuleHom G smult V smult' T
  for G      :: 'g::group-add set
  and smult :: ('f::field, 'g) aezfun  $\Rightarrow$  'v::ab-group-add  $\Rightarrow$  'v (infixr · 70)
  and V      :: 'v set
  and smult' :: ('f, 'g) aezfun  $\Rightarrow$  'w::ab-group-add  $\Rightarrow$  'w (infixr ★ 70)
  and T      :: 'v  $\Rightarrow$  'w
+ fixes W      :: 'w set
  assumes bijjective: bij-betw T V W

```

```

abbreviation (in FGModule) isomorphic ::
  (('f, 'g) aezfun  $\Rightarrow$  'w::ab-group-add  $\Rightarrow$  'w)  $\Rightarrow$  'w set  $\Rightarrow$  bool
  where isomorphic smult' W  $\equiv$  ( $\exists T. \text{FGModuleIso } G \text{ smult } V \text{ smult}' T W$ )

```

5.4.2 Basic facts

```

context FGModule
begin

```

```

lemma trivial-FGModuleHom :
  assumes  $\bigwedge r. r \in FG \Longrightarrow \text{smult}' r 0 = 0$ 
  shows FGModuleHom G smult V smult' 0
  ⟨proof⟩

```

lemma *FGModuleHom-idhom* : *FGModuleHom* *G smult V smult* (*id*↓*V*)

⟨*proof*⟩

lemma *VecHom-GMap-is-FGModuleHom* :

fixes *smult'* :: (*f*, *g*) *aezfun* ⇒ *w*::*ab-group-add* ⇒ *w* (**infixr** ★ 70)

and *fsmult'* :: *f* ⇒ *w* ⇒ *w* (**infixr** ‡★ 70)

and *Gmult'* :: *g* ⇒ *w* ⇒ *w* (**infixr** ** 70)

defines *fsmult'*: *fsmult'* ≡ *aezfun-scalar-mult.fsmult smult'*

and *Gmult'* : *Gmult'* ≡ *aezfun-scalar-mult.Gmult smult'*

assumes *hom* : *VectorSpaceHom fsmult V fsmult' T*

and *Im-W* : *FGModule G smult' W T ' V* ⊆ *W*

and *G-map* : $\bigwedge g v. g \in G \implies v \in V \implies T (g * \cdot v) = g ** (T v)$

shows *FGModuleHom G smult V smult' T*

⟨*proof*⟩

lemma *VecHom-GMap-on-fbasis-is-FGModuleHom* :

fixes *smult'* :: (*f*, *g*) *aezfun* ⇒ *w*::*ab-group-add* ⇒ *w* (**infixr** ★ 70)

and *fsmult'* :: *f* ⇒ *w* ⇒ *w* (**infixr** ‡★ 70)

and *Gmult'* :: *g* ⇒ *w* ⇒ *w* (**infixr** ** 70)

and *flincomb'* :: *f list* ⇒ *w list* ⇒ *w* (**infixr** ·‡★ 70)

defines *fsmult'* : *fsmult'* ≡ *aezfun-scalar-mult.fsmult smult'*

and *Gmult'* : *Gmult'* ≡ *aezfun-scalar-mult.Gmult smult'*

and *flincomb'* : *flincomb'* ≡ *aezfun-scalar-mult.flincomb smult'*

assumes *fbasis* : *fbasis-for V vs*

and *hom* : *VectorSpaceHom fsmult V fsmult' T*

and *Im-W* : *FGModule G smult' W T ' V* ⊆ *W*

and *G-map* : $\bigwedge g v. g \in G \implies v \in \text{set } vs \implies T (g * \cdot v) = g ** (T v)$

shows *FGModuleHom G smult V smult' T*

⟨*proof*⟩

end

context *FGModuleHom*

begin

abbreviation *fsmult'* :: *f* ⇒ *w* ⇒ *w* (**infixr** ‡★ 70)

where *fsmult'* ≡ *aezfun-scalar-mult.fsmult smult'*

abbreviation *Gmult'* :: *g* ⇒ *w* ⇒ *w* (**infixr** ** 70)

where *Gmult'* ≡ *aezfun-scalar-mult.Gmult smult'*

lemmas *supp* = *supp*

lemmas *additive* = *additive*

lemmas *FG-map* = *R-map*

lemmas *FG-fdd0-closed* = *FG-fdd0-closed*

lemmas *fsmult-smult-domain-comm* = *fsmult-smult-comm*

lemmas *GSubspace-Ker* = *RSubmodule-Ker*

lemmas *Ker-Im-iff* = *Ker-Im-iff*

lemmas *Ker0-imp-inj-on* = *Ker0-imp-inj-on*

lemmas *eq-im-imp-diff-in-Ker* = *eq-im-imp-diff-in-Ker*
lemmas *im-submodule* = *im-submodule*
lemmas *fsmultD'* = *aezfun-scalar-mult.fsmultD[of smult']*
lemmas *GmultD'* = *aezfun-scalar-mult.GmultD[of smult']*

lemma *f-map* : $v \in V \implies T (a \# \cdot v) = a \# \star T v$
 <proof>

lemma *G-map* : $g \in G \implies v \in V \implies T (g * \cdot v) = g * \star T v$
 <proof>

lemma *VectorSpaceHom* : *VectorSpaceHom fsmult V fsmult' T*
 <proof>

lemmas *distrib-flincomb* = *VectorSpaceHom.distrib-lincomb[OF VectorSpaceHom]*

lemma *FGModule-Im* : *FGModule G smult' ImG*
 <proof>

lemma *FGModHom-composite-left* :
assumes *FGModuleHom G smult' W smult'' S T ' V \subseteq W*
shows *FGModuleHom G smult V smult'' (S \circ T)*
 <proof>

lemma *restriction-to-subgroup-is-hom* :
fixes *H* :: 'g set
assumes *subgrp: Group.Subgroup G H*
shows *FGModuleHom H smult V smult' T*
 <proof>

lemma *FGModuleHom-restrict0-GSubspace* :
assumes *GSubspace U*
shows *FGModuleHom G smult U smult' (T \downarrow U)*
 <proof>

lemma *FGModuleHom-fscalar-mul* :
FGModuleHom G smult V smult' ($\lambda v. a \# \star T v$)
 <proof>

end

lemma *GSubspace-eigenspace* :
fixes *e* :: 'f::field
and *E* :: 'v::ab-group-add set
and *smult* :: ('f::field, 'g::group-add) *aezfun* \implies 'v \implies 'v (**infixr** \cdot 70)
assumes *FGModHom: FGModuleHom G smult V smult T*
defines *E* : $E \equiv \{v \in V. T v = aezfun-scalar-mult.fsmult smult e v\}$
shows *FGModule.GSubspace G smult V E*
 <proof>

5.4.3 Basic facts about endomorphisms

lemma *RModuleEnd-over-group-ring-is-FGModuleEnd* :
fixes $G :: 'g::\text{group-add set}$
and $\text{smult} :: ('f::\text{field}, 'g) \text{aezfun} \Rightarrow 'v::\text{ab-group-add} \Rightarrow 'v$
assumes $G : \text{Group } G$ **and** $\text{endo} : \text{RModuleEnd } (\text{Group.group-ring } G) \text{ smult } V T$
shows $\text{FGModuleEnd } G \text{ smult } V T$
<proof>

lemma (**in** *FGModule*) *VecEnd-GMap-is-FGModuleEnd* :
assumes $\text{endo} : \text{VectorSpaceEnd } \text{fsmult } V T$
and $G\text{-map} : \bigwedge g v. g \in G \Longrightarrow v \in V \Longrightarrow T (g * \cdot v) = g * \cdot (T v)$
shows $\text{FGModuleEnd } G \text{ smult } V T$
<proof>

lemma (**in** *FGModule*) *GEnd-inner-dirsum-el-decomp-nth* :
 $\llbracket \forall U \in \text{set } Us. G\text{Subspace } U; \text{add-independent } S \text{ } Us; n < \text{length } Us \rrbracket$
 $\Longrightarrow \text{FGModuleEnd } G \text{ smult } (\bigoplus U \leftarrow Us. U) (\bigoplus Us \downarrow n)$
<proof>

context *FGModuleEnd*
begin

lemma *RModuleEnd* : $\text{RModuleEnd } \text{ActingGroup.group-ring } \text{smult } V T$
<proof>

lemma *VectorSpaceEnd* : $\text{VectorSpaceEnd } \text{fsmult } V T$
<proof>

lemmas *proj-decomp* = $\text{RModuleEnd.proj-decomp}[OF \text{RModuleEnd}]$
lemmas *GSubspace-Ker* = $G\text{Subspace-Ker}$
lemmas *FGModuleHom-restrict0-GSubspace* = $\text{FGModuleHom-restrict0-GSubspace}$

end

5.4.4 Basic facts about isomorphisms

context *FGModuleIso*
begin

lemmas *VectorSpaceHom* = VectorSpaceHom

abbreviation $\text{inv}T \equiv (\text{the-inv-into } V T) \downarrow W$

lemma *RModuleIso* : $\text{RModuleIso } FG \text{ smult } V \text{ smult}' T W$
<proof>

lemmas $\text{Im}G = \text{RModuleIso.Im}G[OF \text{RModuleIso}]$

lemma *FGModuleIso-restrict0-GSubspace* :

assumes $GSubspace\ U$
shows $FGModuleIso\ G\ smult\ U\ smult'\ (T\ \downarrow\ U)\ (T\ \leftarrow\ U)$
 $\langle proof \rangle$

lemma $inv : FGModuleIso\ G\ smult'\ W\ smult\ invT\ V$
 $\langle proof \rangle$

lemma $FGModIso-composite-left :$
assumes $FGModuleIso\ G\ smult'\ W\ smult''\ S\ X$
shows $FGModuleIso\ G\ smult\ V\ smult''\ (S\ \circ\ T)\ X$
 $\langle proof \rangle$

lemma $isomorphic-sym : FGModule.isomorphic\ G\ smult'\ W\ smult\ V$
 $\langle proof \rangle$

lemma $isomorphic-trans :$
 $FGModule.isomorphic\ G\ smult'\ W\ smult''\ X$
 $\implies FGModule.isomorphic\ G\ smult\ V\ smult''\ X$
 $\langle proof \rangle$

lemma $isomorphic-to-zero-left : V = 0 \implies W = 0$
 $\langle proof \rangle$

lemma $isomorphic-to-zero-right : W = 0 \implies V = 0$
 $\langle proof \rangle$

lemma $isomorphic-to-irr-right' :$
assumes $\bigwedge U. FGModule.GSubspace\ G\ smult'\ W\ U \implies U = 0 \vee U = W$
shows $\bigwedge U. GSubspace\ U \implies U = 0 \vee U = V$
 $\langle proof \rangle$

end

context $FGModule$
begin

lemma $isomorphic-sym :$
 $isomorphic\ smult'\ W \implies FGModule.isomorphic\ G\ smult'\ W\ smult\ V$
 $\langle proof \rangle$

lemma $isomorphic-trans :$
 $isomorphic\ smult'\ W \implies FGModule.isomorphic\ G\ smult'\ W\ smult''\ X$
 $\implies isomorphic\ smult''\ X$
 $\langle proof \rangle$

lemma $isomorphic-to-zero-left : V = 0 \implies isomorphic\ smult'\ W \implies W = 0$
 $\langle proof \rangle$

lemma $isomorphic-to-zero-right : isomorphic\ smult'\ 0 \implies V = 0$

<proof>

lemma *FGModIso-idhom* : *FGModuleIso G smult V smult (id↓V) V*
<proof>

lemma *isomorphic-refl* : *isomorphic smult V <proof>*

end

5.4.5 Hom-sets

definition *FGModuleHomSet* ::

'g::group-add set ⇒ (('f::field,'g) aezfun ⇒ 'v::ab-group-add ⇒ 'v) ⇒ 'v set
⇒ (('f,'g) aezfun ⇒ 'w::ab-group-add ⇒ 'w) ⇒ 'w set
⇒ ('v ⇒ 'w) set

where *FGModuleHomSet G fgsmult V fgsmult' W*
≡ {T. FGModuleHom G fgsmult V fgsmult' T} ∩ {T. T ' V ⊆ W}

lemma *FGModuleHomSetI* :

FGModuleHom G fgsmult V fgsmult' T ⇒ T ' V ⊆ W
⇒ T ∈ FGModuleHomSet G fgsmult V fgsmult' W
<proof>

lemma *FGModuleHomSetD-FGModuleHom* :

T ∈ FGModuleHomSet G fgsmult V fgsmult' W
⇒ FGModuleHom G fgsmult V fgsmult' T
<proof>

lemma *FGModuleHomSetD-Im* :

T ∈ FGModuleHomSet G fgsmult V fgsmult' W ⇒ T ' V ⊆ W
<proof>

context *FGModule*

begin

lemma *FGModuleHomSet-is-Gmaps-in-VectorSpaceHomSet* :

fixes *smult' :: ('f, 'g) aezfun ⇒ 'w::ab-group-add ⇒ 'w (infixr ★ 70)*

and *fsmult' :: 'f ⇒ 'w ⇒ 'w (infixr ‡★ 70)*

and *Gmult' :: 'g ⇒ 'w ⇒ 'w (infixr ** 70)*

defines *fsmult' : fsmult' ≡ aezfun-scalar-mult.fsmult smult'*

and *Gmult' : Gmult' ≡ aezfun-scalar-mult.Gmult smult'*

assumes *FGModW : FGModule G smult' W*

shows *FGModuleHomSet G smult V smult' W*

= (VectorSpaceHomSet fsmult V fsmult' W)
*∩ {T. ∀ g∈G. ∀ v∈V. T (g * v) = g ** (T v)}*

<proof>

lemma *Group-FGModuleHomSet* :

fixes *smult' :: ('f, 'g) aezfun ⇒ 'w::ab-group-add ⇒ 'w (infixr ★ 70)*

and $fsmult' :: 'f \Rightarrow 'w \Rightarrow 'w$ (**infixr** $\#^*$ 70)
and $Gmult' :: 'g \Rightarrow 'w \Rightarrow 'w$ (**infixr** $**$ 70)
defines $fsmult' : fsmult' \equiv aezfun\text{-}scalar\text{-}mult.fsmult\ smult'$
and $Gmult' : Gmult' \equiv aezfun\text{-}scalar\text{-}mult.Gmult\ smult'$
assumes $FGModW : FGModule\ G\ smult'\ W$
shows $Group\ (FGModuleHomSet\ G\ smult'\ V\ smult'\ W)$
 $\langle proof \rangle$

lemma *Subspace-FGModuleHomSet* :
fixes $smult' :: ('f, 'g)\ aezfun \Rightarrow 'w::ab\text{-}group\text{-}add \Rightarrow 'w$ (**infixr** \star 70)
and $fsmult' :: 'f \Rightarrow 'w \Rightarrow 'w$ (**infixr** $\#^*$ 70)
and $Gmult' :: 'g \Rightarrow 'w \Rightarrow 'w$ (**infixr** $**$ 70)
and $hom\text{-}fsmult :: 'f \Rightarrow ('v \Rightarrow 'w) \Rightarrow ('v \Rightarrow 'w)$ (**infixr** $\#^*$ 70)
defines $fsmult' : fsmult' \equiv aezfun\text{-}scalar\text{-}mult.fsmult\ smult'$
and $Gmult' : Gmult' \equiv aezfun\text{-}scalar\text{-}mult.Gmult\ smult'$
defines $hom\text{-}fsmult : hom\text{-}fsmult \equiv \lambda a\ T\ v.\ a\ \#^*\ T\ v$
assumes $FGModW : FGModule\ G\ smult'\ W$
shows $VectorSpace.Subspace\ hom\text{-}fsmult$
 $(VectorSpaceHomSet\ fsmult\ V\ fsmult'\ W)$
 $(FGModuleHomSet\ G\ smult'\ V\ smult'\ W)$
 $\langle proof \rangle$

lemma *VectorSpace-FGModuleHomSet* :
fixes $smult' :: ('f, 'g)\ aezfun \Rightarrow 'w::ab\text{-}group\text{-}add \Rightarrow 'w$ (**infixr** \star 70)
and $fsmult' :: 'f \Rightarrow 'w \Rightarrow 'w$ (**infixr** $\#^*$ 70)
and $hom\text{-}fsmult :: 'f \Rightarrow ('v \Rightarrow 'w) \Rightarrow ('v \Rightarrow 'w)$ (**infixr** $\#^*$ 70)
defines $fsmult' \equiv aezfun\text{-}scalar\text{-}mult.fsmult\ smult'$
defines $hom\text{-}fsmult \equiv \lambda a\ T\ v.\ a\ \#^*\ T\ v$
assumes $FGModule\ G\ smult'\ W$
shows $VectorSpace\ hom\text{-}fsmult\ (FGModuleHomSet\ G\ smult'\ V\ smult'\ W)$
 $\langle proof \rangle$

end

5.5 Induced modules

5.5.1 Additive function spaces

definition *addfunset* ::
 $'a::monoid\text{-}add\ set \Rightarrow 'm::monoid\text{-}add\ set \Rightarrow ('a \Rightarrow 'm)\ set$
where $addfunset\ A\ M \equiv \{f.\ supp\ f \subseteq A \wedge range\ f \subseteq M$
 $\wedge (\forall x \in A.\ \forall y \in A.\ f\ (x+y) = f\ x + f\ y)\}$

lemma *addfunsetI* :
 $\llbracket supp\ f \subseteq A; range\ f \subseteq M; \forall x \in A.\ \forall y \in A.\ f\ (x+y) = f\ x + f\ y \rrbracket$
 $\implies f \in addfunset\ A\ M$
 $\langle proof \rangle$

lemma *addfunsetD-supp* : $f \in addfunset\ A\ M \implies supp\ f \subseteq A$
 $\langle proof \rangle$

lemma *addfunsetD-range* : $f \in \text{addfunset } A \ M \implies \text{range } f \subseteq M$
 ⟨proof⟩

lemma *addfunsetD-range'* : $f \in \text{addfunset } A \ M \implies f \ x \in M$
 ⟨proof⟩

lemma *addfunsetD-add* :
 $\llbracket f \in \text{addfunset } A \ M; x \in A; y \in A \rrbracket \implies f \ (x+y) = f \ x + f \ y$
 ⟨proof⟩

lemma *addfunset0* : $\text{addfunset } A \ (0::'m::\text{monoid-add set}) = 0$
 ⟨proof⟩

lemma *Group-addfunset* :
 fixes $M::'g::\text{ab-group-add set}$
 assumes *Group* M
 shows *Group* $(\text{addfunset } R \ M)$
 ⟨proof⟩

5.5.2 Spaces of functions which transform under scalar multiplication by almost-everywhere-zero functions

context *aezfun-scalar-mult*
begin

definition *smultfunset* :: $'g \ \text{set} \Rightarrow ('r, 'g) \ \text{aezfun set} \Rightarrow (('r, 'g) \ \text{aezfun} \Rightarrow 'v) \ \text{set}$
 where $\text{smultfunset } G \ FH \equiv \{f. (\forall a::'r. \forall g \in G. \forall x \in FH. f \ (a \ \delta\delta \ g \ * \ x) = (a \ \delta\delta \ g) \cdot (f \ x))\}$

lemma *smultfunsetD* :
 $\llbracket f \in \text{smultfunset } G \ FH; g \in G; x \in FH \rrbracket \implies f \ (a \ \delta\delta \ g \ * \ x) = (a \ \delta\delta \ g) \cdot (f \ x)$
 ⟨proof⟩

lemma *smultfunsetI* :
 $\forall a::'r. \forall g \in G. \forall x \in FH. f \ (a \ \delta\delta \ g \ * \ x) = (a \ \delta\delta \ g) \cdot (f \ x)$
 $\implies f \in \text{smultfunset } G \ FH$
 ⟨proof⟩

end

5.5.3 General induced spaces of functions on a group ring

context *aezfun-scalar-mult*
begin

definition *indspace* ::
 $'g \ \text{set} \Rightarrow ('r, 'g) \ \text{aezfun set} \Rightarrow 'v \ \text{set} \Rightarrow (('r, 'g) \ \text{aezfun} \Rightarrow 'v) \ \text{set}$
 where $\text{indspace } G \ FH \ V = \text{addfunset } FH \ V \cap \text{smultfunset } G \ FH$

lemma *indspaceD* :

$f \in \text{indspace } G \text{ } FH \text{ } V \implies f \in \text{addfunset } FH \text{ } V \cap \text{multfunset } G \text{ } FH$
<proof>

lemma *indspaceD-supp* : $f \in \text{indspace } G \text{ } FH \text{ } V \implies \text{supp } f \subseteq FH$

<proof>

lemma *indspaceD-supp'* : $f \in \text{indspace } G \text{ } FH \text{ } V \implies x \notin FH \implies f x = 0$

<proof>

lemma *indspaceD-range* : $f \in \text{indspace } G \text{ } FH \text{ } V \implies \text{range } f \subseteq V$

<proof>

lemma *indspaceD-range'* : $f \in \text{indspace } G \text{ } FH \text{ } V \implies f x \in V$

<proof>

lemma *indspaceD-add* :

$\llbracket f \in \text{indspace } G \text{ } FH \text{ } V; x \in FH; y \in FH \rrbracket \implies f (x+y) = f x + f y$
<proof>

lemma *indspaceD-transform* :

$\llbracket f \in \text{indspace } G \text{ } FH \text{ } V; g \in G; x \in FH \rrbracket \implies f (a \delta \delta g * x) = (a \delta \delta g) \cdot (f x)$
<proof>

lemma *indspaceI* :

$f \in \text{addfunset } FH \text{ } V \implies f \in \text{multfunset } G \text{ } FH \implies f \in \text{indspace } G \text{ } FH \text{ } V$
<proof>

lemma *indspaceI'* :

$\llbracket \text{supp } f \subseteq FH; \text{range } f \subseteq V; \forall x \in FH. \forall y \in FH. f (x+y) = f x + f y;$
 $\forall a::'r. \forall g \in G. \forall x \in FH. f (a \delta \delta g * x) = (a \delta \delta g) \cdot (f x) \rrbracket$
 $\implies f \in \text{indspace } G \text{ } FH \text{ } V$

<proof>

lemma *mono-indspace* : *mono (indspace G FH)*

<proof>

end

context *FGModule*

begin

lemma *zero-transforms* : $0 \in \text{multfunset } G \text{ } FH$

<proof>

lemma *indspace0* : $\text{indspace } G \text{ } FH \text{ } 0 = 0$

<proof>

lemma *Group-indspace* :

assumes *Ring1 FH*
shows *Group (indspace G FH V)*
 ⟨*proof*⟩

end

5.5.4 The right regular action

context *Ring1*
begin

definition *rightreg-scalar-mult* ::
'r::ring-1 \Rightarrow (*'r* \Rightarrow *'m::ab-group-add*) \Rightarrow (*'r* \Rightarrow *'m*) (**infixr** \bowtie 70)
where *rightreg-scalar-mult* *r f* = ($\lambda x.$ if $x \in R$ then $f(x*r)$ else 0)

lemma *rightreg-scalar-multD1* : $x \in R \Longrightarrow (r \bowtie f) x = f(x*r)$
 ⟨*proof*⟩

lemma *rightreg-scalar-multD2* : $x \notin R \Longrightarrow (r \bowtie f) x = 0$
 ⟨*proof*⟩

lemma *rrsmult-supp* : $\text{supp } (r \bowtie f) \subseteq R$
 ⟨*proof*⟩

lemma *rrsmult-range* : $\text{range } (r \bowtie f) \subseteq \{0\} \cup \text{range } f$
 ⟨*proof*⟩

lemma *rrsmult-distrib-left* : $r \bowtie (f + g) = r \bowtie f + r \bowtie g$
 ⟨*proof*⟩

lemma *rrsmult-distrib-right* :
assumes $\bigwedge x y. x \in R \Longrightarrow y \in R \Longrightarrow f(x+y) = f x + f y$ $r \in R$ $s \in R$
shows $(r + s) \bowtie f = r \bowtie f + s \bowtie f$
 ⟨*proof*⟩

lemma *RModule-addfunset* :
fixes *M::'g::ab-group-add set*
assumes *Group M*
shows *RModule R rightreg-scalar-mult (addfunset R M)*
 ⟨*proof*⟩

end

5.5.5 Locale and basic facts

In the following locale, G is a subgroup of H , V is a module over the group ring for G , and the induced space $\text{ind}V$ will be shown to be a module over the group ring for H under the right regular scalar multiplication *rrsmult*.

locale *InducedFHModule = Supgroup?: Group H*

```

+ BaseFGMod? : FGModule G smult V
+ induced-smult?: aezfun-scalar-mult rrsmult
  for H      :: 'g::group-add set
  and G      :: 'g set
  and FG     :: ('f::field, 'g) aezfun set
  and smult  :: ('f, 'g) aezfun  $\Rightarrow$  'v::ab-group-add  $\Rightarrow$  'v (infixl · 70)
  and V      :: 'v set
  and rrsmult :: ('f,'g) aezfun  $\Rightarrow$  (('f,'g) aezfun  $\Rightarrow$  'v)  $\Rightarrow$  (('f,'g) aezfun  $\Rightarrow$  'v)
                                                    (infixl  $\bowtie$  70)

+ fixes FH    :: ('f, 'g) aezfun set
  and indV    :: (('f, 'g) aezfun  $\Rightarrow$  'v) set
  defines FH  : FH  $\equiv$  Supgroup.group-ring
  and indV   : indV  $\equiv$  BaseFGMod.indspace G FH V
  assumes rrsmult : rrsmult = Ring1.rightreg-scalar-mult FH
  and Subgroup: Supgroup.Subgroup G
begin

abbreviation indfsmult ::
  'f  $\Rightarrow$  (('f, 'g) aezfun  $\Rightarrow$  'v)  $\Rightarrow$  (('f, 'g) aezfun  $\Rightarrow$  'v) (infixl  $\bowtie$  70)
  where indfsmult  $\equiv$  induced-smult.fsmult
abbreviation indflincomb ::
  'f list  $\Rightarrow$  (('f, 'g) aezfun  $\Rightarrow$  'v) list  $\Rightarrow$  (('f, 'g) aezfun  $\Rightarrow$  'v) (infixl ·  $\bowtie$  70)
  where indflincomb  $\equiv$  induced-smult.flincomb
abbreviation Hmult ::
  'g  $\Rightarrow$  (('f, 'g) aezfun  $\Rightarrow$  'v)  $\Rightarrow$  (('f, 'g) aezfun  $\Rightarrow$  'v) (infixl *  $\bowtie$  70)
  where Hmult  $\equiv$  induced-smult.Gmult

lemma Ring1-FH : Ring1 FH  $\langle$ proof $\rangle$ 

lemma FG-subring-FH : Ring1.Subring1 FH BaseFGMod.FG
   $\langle$ proof $\rangle$ 

lemma rrsmultD1 :  $x \in FH \Longrightarrow (r \bowtie f) x = f (x*r)$ 
   $\langle$ proof $\rangle$ 

lemma rrsmultD2 :  $x \notin FH \Longrightarrow (r \bowtie f) x = 0$ 
   $\langle$ proof $\rangle$ 

lemma rrsmult-supp :  $\text{supp } (r \bowtie f) \subseteq FH$ 
   $\langle$ proof $\rangle$ 

lemma rrsmult-range :  $\text{range } (r \bowtie f) \subseteq \{0\} \cup \text{range } f$ 
   $\langle$ proof $\rangle$ 

lemma FHModule-addfunset : FGModule H rrsmult (addfunset FH V)
   $\langle$ proof $\rangle$ 

lemma FHSubmodule-indspace :
  FGModule.FGSubmodule H rrsmult (addfunset FH V) indV

```

$\langle proof \rangle$

lemma *FHModule-indspace* : *FGModule H rrsmult indV*

$\langle proof \rangle$

lemmas *fVectorSpace-indspace* = *FGModule.fVectorSpace*[*OF FHModule-indspace*]

lemmas *restriction-is-FGModule*

= *FGModule.restriction-to-subgroup-is-module*[*OF FHModule-indspace*]

definition *induced-vector* :: '*v* \Rightarrow ((*f*, '*g*) *aezfun* \Rightarrow '*v*)

where *induced-vector* *v* \equiv (*if* *v* \in *V*

then (λy . *if* *y* \in *FH* then (*FG-proj* *y*) \cdot *v* else 0) else 0)

lemma *induced-vector-apply1* :

v \in *V* \Rightarrow *x* \in *FH* \Rightarrow *induced-vector* *v* *x* = (*FG-proj* *x*) \cdot *v*

$\langle proof \rangle$

lemma *induced-vector-apply2* : *v* \in *V* \Rightarrow *x* \notin *FH* \Rightarrow *induced-vector* *v* *x* = 0

$\langle proof \rangle$

lemma *induced-vector-indV* :

assumes *v*: *v* \in *V*

shows *induced-vector* *v* \in *indV*

$\langle proof \rangle$

lemma *induced-vector-additive* :

v \in *V* \Rightarrow *v'* \in *V*

\Rightarrow *induced-vector* (*v*+*v'*) = *induced-vector* *v* + *induced-vector* *v'*

$\langle proof \rangle$

lemma *hom-induced-vector* : *FGModuleHom G smult V rrsmult induced-vector*

$\langle proof \rangle$

lemma *indspace-sum-list-fddh*:

\llbracket *fhs* \neq []; *set* (*map snd fhs*) \subseteq *H*; *f* \in *indV* \rrbracket

\Rightarrow *f* (\sum (*a,h*) \leftarrow *fhs*. *a* $\delta\delta$ *h*) = (\sum (*a,h*) \leftarrow *fhs*. *f* (*a* $\delta\delta$ *h*))

$\langle proof \rangle$

lemma *induced-fsmult-conv-fsmult-1ddh* :

f \in *indV* \Rightarrow *h* \in *H* \Rightarrow (*r* \bowtie *f*) (*1* $\delta\delta$ *h*) = *r* \ddagger . (*f* (*1* $\delta\delta$ *h*))

$\langle proof \rangle$

lemma *indspace-el-eq-on-1ddh-imp-eq-on-rddh* :

assumes *HmodG* \subseteq *H* *H* = (\bigcup *h* \in *HmodG*. *G* + {*h*}) *f* \in *indV* *f'* \in *indV*

\forall *h* \in *HmodG*. *f* (*1* $\delta\delta$ *h*) = *f'* (*1* $\delta\delta$ *h*) *h* \in *H*

shows *f* (*r* $\delta\delta$ *h*) = *f'* (*r* $\delta\delta$ *h*)

$\langle proof \rangle$

lemma *indspace-el-eq* :

assumes $H\text{mod}G \subseteq H$ $H = (\bigcup h \in H\text{mod}G. G + \{h\})$ $f \in \text{ind}V$ $f' \in \text{ind}V$
 $\forall h \in H\text{mod}G. f (1 \ \delta \delta \ h) = f' (1 \ \delta \delta \ h)$
shows $f = f'$
 $\langle \text{proof} \rangle$

lemma *indspace-el-eq'* :
assumes $\text{set } hs \subseteq H$ $H = (\bigcup h \in \text{set } hs. G + \{h\})$ $f \in \text{ind}V$ $f' \in \text{ind}V$
 $\forall i < \text{length } hs. f (1 \ \delta \delta \ (hs!i)) = f' (1 \ \delta \delta \ (hs!i))$
shows $f = f'$
 $\langle \text{proof} \rangle$

end

6 Representations of Finite Groups

6.1 Locale and basic facts

Define a group representation to be a module over the group ring that is finite-dimensional as a vector space. The only restriction on the characteristic of the field is that it does not divide the order of the group. Also, we don't explicitly assume that the group is finite; instead, the *good-char* assumption implies that the cardinality of G is not zero, which implies G is finite. (See lemma *good-card-imp-finite*.)

locale *FinGroupRepresentation* = *FGModule* G *smult* V
for G :: $'g::\text{group-add set}$
and *smult* :: $('f::\text{field}, 'g) \text{ aezfun} \Rightarrow 'v::\text{ab-group-add} \Rightarrow 'v$ (**infixl** · 70)
and V :: $'v \text{ set}$
+
assumes *good-char*: $\text{of-nat } (\text{card } G) \neq (0::'f)$
and *findim* : $\text{fscalar-mult.findim } \text{fsmult } V$

lemma (**in** *Group*) *trivial-FinGroupRep* :
fixes *smult* :: $('f::\text{field}, 'g) \text{ aezfun} \Rightarrow 'v::\text{ab-group-add} \Rightarrow 'v$
assumes *good-char* : $\text{of-nat } (\text{card } G) \neq (0::'f)$
and *smult-zero* : $\forall a \in \text{group-ring}. \text{smult } a (0::'v) = 0$
shows *FinGroupRepresentation* G *smult* $(0::'v \text{ set})$
 $\langle \text{proof} \rangle$

context *FinGroupRepresentation*
begin

abbreviation *ordG* :: $'f$ **where** $\text{ord}G \equiv \text{of-nat } (\text{card } G)$
abbreviation *GRepHom* $\equiv \text{FGModuleHom } G$ *smult* V
abbreviation *GRepIso* $\equiv \text{FGModuleIso } G$ *smult* V
abbreviation *GRepEnd* $\equiv \text{FGModuleEnd } G$ *smult* V

lemmas *zero-closed* = *zero-closed*
lemmas *Group* = *Group*

lemmas *GSubmodule-GSpan-single* = *RSubmodule-RSpan-single*
lemmas *GSpan-single-nonzero* = *RSpan-single-nonzero*

lemma *finiteG*: *finite G*
 ⟨*proof*⟩

lemma *FinDimVectorSpace*: *FinDimVectorSpace fsmult V*
 ⟨*proof*⟩

lemma *GSubspace-is-FinGroupRep* :
assumes *GSubspace U*
shows *FinGroupRepresentation G smult U*
 ⟨*proof*⟩

lemma *isomorphic-imp-GRep* :
assumes *isomorphic smult' W*
shows *FinGroupRepresentation G smult' W*
 ⟨*proof*⟩

end

6.2 Irreducible representations

locale *IrrFinGroupRepresentation* = *FinGroupRepresentation*
 + **assumes** *irr*: *GSubspace U* $\implies U = 0 \vee U = V$
begin

lemmas *AbGroup* = *AbGroup*

lemma *zero-isomorphic-to-FG-zero* :
assumes *V = 0*
shows *isomorphic (*) (0::('b,'a) aezfun set)*
 ⟨*proof*⟩

lemma *eq-GSpan-single* : *v* ∈ *V* $\implies v \neq 0 \implies V = GSpan [v]$
 ⟨*proof*⟩

end

lemma (in *Group*) *trivial-IrrFinGroupRepI* :
fixes *smult* :: ('f::field, 'g) aezfun $\implies 'v::ab-group-add \implies 'v$
assumes *of-nat (card G) $\neq (0::'f)$*
and $\forall a \in \text{group-ring. } smult\ a\ (0::'v) = 0$
shows *IrrFinGroupRepresentation G smult (0::'v set)*
 ⟨*proof*⟩

lemma (in *Group*) *trivial-IrrFinGroupRepresentation-in-FG* :
of-nat (card G) $\neq (0::'f::field)$
 $\implies IrrFinGroupRepresentation\ G\ (*)\ (0::('f,'g)\ aezfun\ set)$

<proof>

context *FinGroupRepresentation*
begin

lemma *IrrFinGroupRep-trivialGSubspace* :
 IrrFinGroupRepresentation G smult (0::'v set)
<proof>

lemma *IrrI* :
 assumes $\bigwedge U. \text{FGModule.GSubspace } G \text{ smult } V \ U \implies U = 0 \vee U = V$
 shows *IrrFinGroupRepresentation G smult V*
<proof>

lemma *notIrr* :
 $\neg \text{IrrFinGroupRepresentation } G \text{ smult } V$
 $\implies \exists U. \text{GSubspace } U \wedge U \neq 0 \wedge U \neq V$
<proof>

end

6.3 Maschke's theorem

6.3.1 Averaged projection onto a G-subspace

context *FinGroupRepresentation*
begin

lemma *avg-proj-eq-id-on-right* :
 assumes *VectorSpace fsmult W add-independentS [W, V] v ∈ V*
 defines $P : P \equiv (\bigoplus [W, V] \downarrow 1)$
 defines $CP : CP \equiv (\lambda g. \text{Gmult } (- g) \circ P \circ \text{Gmult } g)$
 defines $T : T \equiv \text{fsmult } (1/\text{ord}G) \circ (\sum_{g \in G}. CP \ g)$
 shows $T \ v = v$
<proof>

lemma *avg-proj-onto-right* :
 assumes *VectorSpace fsmult W GSubspace U add-independentS [W, U]*
 $V = W \oplus U$
 defines $P : P \equiv (\bigoplus [W, U] \downarrow 1)$
 defines $CP : CP \equiv (\lambda g. \text{Gmult } (- g) \circ P \circ \text{Gmult } g)$
 defines $T : T \equiv \text{fsmult } (1/\text{ord}G) \circ (\sum_{g \in G}. CP \ g)$
 shows $T \ ` V = U$
<proof>

lemma *FGModuleEnd-avg-proj-right* :
 assumes *fSubspace W GSubspace U add-independentS [W, U] V = W ⊕ U*
 defines $P : P \equiv (\bigoplus [W, U] \downarrow 1)$
 defines $CP : CP \equiv (\lambda g. \text{Gmult } (- g) \circ P \circ \text{Gmult } g)$
 defines $T : T \equiv (\text{fsmult } (1/\text{ord}G) \circ (\sum_{g \in G}. CP \ g)) \downarrow V$

shows $FGModuleEnd\ G\ smult\ V\ T$
 $\langle proof \rangle$

lemma *avg-proj-is-proj-right* :

assumes $VectorSpace\ fsmult\ W\ GSubspace\ U\ add-independentS\ [W,U]$
 $V = W \oplus U\ v \in V$

defines $P : P \equiv (\bigoplus [W,U] \downarrow 1)$

defines $CP : CP \equiv (\lambda g. Gmult\ (-\ g) \circ P \circ Gmult\ g)$

defines $T : T \equiv fsmult\ (1/ordG) \circ (\sum g \in G. CP\ g)$

shows $T\ (T\ v) = T\ v$

$\langle proof \rangle$

end

6.3.2 The theorem

context $FinGroupRepresentation$

begin

theorem *Maschke* :

assumes $GSubspace\ U$

shows $\exists W. GSubspace\ W \wedge V = W \oplus U$

$\langle proof \rangle$

corollary *Maschke-proper* :

assumes $GSubspace\ U\ U \neq 0\ U \neq V$

shows $\exists W. GSubspace\ W \wedge W \neq 0 \wedge W \neq V \wedge V = W \oplus U$

$\langle proof \rangle$

end

6.3.3 Consequence: complete reducibility

lemma (in $FinGroupRepresentation$) *notIrr-decompose* :

$\neg IrrFinGroupRepresentation\ G\ smult\ V$

$\implies \exists U\ W. GSubspace\ U \wedge U \neq 0 \wedge U \neq V \wedge GSubspace\ W \wedge W \neq 0$
 $\wedge W \neq V \wedge V = U \oplus W$

$\langle proof \rangle$

In the following decomposition lemma, we do not need to explicitly include the condition that all U in set Us are subsets of V . (See lemma *Ab-Group-subset-inner-dirsum*.)

lemma *FinGroupRepresentation-reducible'* :

fixes $n::nat$

shows $\bigwedge V. FinGroupRepresentation\ G\ fgsmult\ V$

$\wedge n = FGModule.fdim\ fgsmult\ V$

$\implies (\exists Us. Ball\ (set\ Us)\ (IrrFinGroupRepresentation\ G\ fgsmult))$

$\wedge (0 \notin set\ Us) \wedge V = (\bigoplus U \leftarrow Us. U)$

$\langle proof \rangle$

theorem (in *FinGroupRepresentation*) *reducible* :
 $\exists Us. (\forall U \in \text{set } Us. \text{IrrFinGroupRepresentation } G \text{ smult } U) \wedge (0 \notin \text{set } Us)$
 $\wedge V = (\bigoplus U \leftarrow Us. U)$
 ⟨proof⟩

6.3.4 Consequence: decomposition relative to a homomorphism

lemma (in *FinGroupRepresentation*) *GRepHom-decomp* :
fixes $T :: 'v \Rightarrow 'w::\text{ab-group-add}$
defines $\text{Ker}T : \text{Ker}T \equiv (\text{ker } T \cap V)$
assumes $\text{hom} : \text{GRepHom } \text{smult}' T$ **and** $\text{nonzero} : V \neq 0$
shows $\exists U. \text{GSubspace } U \wedge V = U \oplus \text{Ker}T$
 $\wedge \text{FGModule.isomorphic } G \text{ smult } U \text{ smult}' (T \text{ ' } V)$
 ⟨proof⟩

6.4 Schur's lemma

lemma (in *IrrFinGroupRepresentation*) *Schur-Ker* :
 $\text{GRepHom } \text{smult}' T \implies T \text{ ' } V \neq 0 \implies \text{inj-on } T V$
 ⟨proof⟩

lemma (in *FinGroupRepresentation*) *Schur-Im* :
assumes $\text{IrrFinGroupRepresentation } G \text{ smult}' W \text{ GRepHom } \text{smult}' T$
 $T \text{ ' } V \subseteq W$
 $T \text{ ' } V \neq 0$
shows $T \text{ ' } V = W$
 ⟨proof⟩

theorem (in *IrrFinGroupRepresentation*) *Schur1* :
assumes $\text{IrrFinGroupRepresentation } G \text{ smult}' W$
 $\text{GRepHom } \text{smult}' T \text{ ' } V \subseteq W \text{ ' } V \neq 0$
shows $\text{GRepIso } \text{smult}' T W$
 ⟨proof⟩

theorem (in *IrrFinGroupRepresentation*) *Schur2* :
assumes $\text{GRep} \quad : \text{GRepEnd } T$
and $\text{fdim} \quad : \text{fdim} > 0$
and $\text{alg-closed} : \bigwedge p::'b \text{ poly. degree } p > 0 \implies \exists c. \text{poly } p \ c = 0$
shows $\exists c. \forall v \in V. T v = c \# \cdot v$
 ⟨proof⟩

6.5 The group ring as a representation space

6.5.1 The group ring is a representation space

lemma (in *Group*) *FGModule-FG* :
defines $\text{FG} : \text{FG} \equiv \text{group-ring} :: ('f::\text{field}, 'g) \text{ aezfun set}$
shows $\text{FGModule } G (*) \text{ FG}$
 ⟨proof⟩

theorem (in *Group*) *FinGroupRepresentation-FG* :
defines FG : $FG \equiv \text{group-ring} :: ('f::\text{field}, 'g) \text{aezfun set}$
assumes *good-char*: $\text{of-nat} (\text{card } G) \neq (0::'f)$
shows *FinGroupRepresentation* $G (*) FG$
 $\langle \text{proof} \rangle$

lemma (in *FinGroupRepresentation*) *FinGroupRepresentation-FG* :
FinGroupRepresentation $G (*) FG$
 $\langle \text{proof} \rangle$

lemma (in *Group*) *FG-reducible* :
assumes $\text{of-nat} (\text{card } G) \neq (0::'f::\text{field})$
shows $\exists Us::('f, 'g) \text{aezfun set list.}$
 $(\forall U \in \text{set } Us. \text{IrrFinGroupRepresentation } G (*) U) \wedge 0 \notin \text{set } Us$
 $\wedge \text{group-ring} = (\bigoplus U \leftarrow Us. U)$
 $\langle \text{proof} \rangle$

6.5.2 Irreducible representations are constituents of the group ring

lemma (in *FGModuleIso*) *isomorphic-to-irr-right* :
assumes *IrrFinGroupRepresentation* $G \text{smult}' W$
shows *IrrFinGroupRepresentation* $G \text{smult } V$
 $\langle \text{proof} \rangle$

lemma (in *FinGroupRepresentation*) *IrrGSubspace-iso-constituent* :
assumes *nonzero* : $V \neq 0$
and *subsp* : $W \subseteq V \ W \neq 0 \ \text{IrrFinGroupRepresentation } G \text{smult } W$
and *V-decomp*: $\forall U \in \text{set } Us. \text{IrrFinGroupRepresentation } G \text{smult } U$
 $0 \notin \text{set } Us \ V = (\bigoplus U \leftarrow Us. U)$
shows $\exists U \in \text{set } Us. \text{FGModule.isomorphic } G \text{smult } W \text{smult } U$
 $\langle \text{proof} \rangle$

theorem (in *IrrFinGroupRepresentation*) *iso-FG-constituent* :
assumes *nonzero* : $V \neq 0$
and *FG-decomp*: $\forall U \in \text{set } Us. \text{IrrFinGroupRepresentation } G (*) U$
 $0 \notin \text{set } Us \ FG = (\bigoplus U \leftarrow Us. U)$
shows $\exists U \in \text{set } Us. \text{isomorphic} (*) U$
 $\langle \text{proof} \rangle$

6.6 Isomorphism classes of irreducible representations

We have already demonstrated that the relation *FGModule.isomorphic* is reflexive (lemma *FGModule.isomorphic-refl*), symmetric (lemma *FGModule.isomorphic-sym*), and transitive (lemma *FGModule.isomorphic-trans*). In this section, we provide a finite set of representatives for the resulting isomorphism classes of irreducible representations.

context *Group*
begin

primrec *remisodups* :: ('f::field,'g) aezfun set list \Rightarrow ('f,'g) aezfun set list **where**
remisodups [] = []
| *remisodups* (U # Us) = (if
 $(\exists W \in \text{set } Us. \text{FGModule.isomorphic } G (*) U (*) W)$
then *remisodups* Us else U # *remisodups* Us)

lemma *set-remisodups* : set (*remisodups* Us) \subseteq set Us
<proof>

lemma *isodistinct-remisodups* :
 $\llbracket \forall U \in \text{set } Us. \text{FGModule } G (*) U; V \in \text{set } (\text{remisodups } Us);$
 $W \in \text{set } (\text{remisodups } Us); V \neq W \rrbracket$
 $\implies \neg (\text{FGModule.isomorphic } G (*) V (*) W)$
<proof>

definition *FG-constituents* \equiv SOME Us.
 $(\forall U \in \text{set } Us. \text{IrrFinGroupRepresentation } G (*) U)$
 $\wedge 0 \notin \text{set } Us \wedge \text{group-ring} = (\bigoplus U \leftarrow Us. U)$

lemma *FG-constituents-irr* :
of-nat (card G) \neq (0::'f::field)
 $\implies \forall U \in \text{set } (\text{FG-constituents}::('f,'g) \text{ aezfun set list}).$
 $\text{IrrFinGroupRepresentation } G (*) U$
<proof>

lemma *FG-constituents-n0*:
of-nat (card G) \neq (0::'f::field)
 $\implies 0 \notin \text{set } (\text{FG-constituents}::('f,'g) \text{ aezfun set list})$
<proof>

lemma *FG-constituents-constituents* :
of-nat (card G) \neq (0::'f::field)
 $\implies (\text{group-ring}::('f,'g) \text{ aezfun set}) = (\bigoplus U \leftarrow \text{FG-constituents. } U)$
<proof>

definition *GIrrRep-repset* \equiv 0 \cup set (*remisodups* *FG-constituents*)

lemma *finite-GIrrRep-repset* : finite *GIrrRep-repset*
<proof>

lemma *all-irr-GIrrRep-repset* :
assumes of-nat (card G) \neq (0::'f::field)
shows $\forall U \in (\text{GIrrRep-repset}::('f,'g) \text{ aezfun set set}).$
 $\text{IrrFinGroupRepresentation } G (*) U$
<proof>

lemma *isodistinct-GIrrRep-repset* :
defines $GIRRS \equiv GIrrRep\text{-}repset :: ('f::field, 'g) \text{ aezfun set set}$
assumes $of\text{-}nat (card\ G) \neq (0::'f) \ V \in GIRRS \ W \in GIRRS \ V \neq W$
shows $\neg (FGModule.isomorphic\ G\ (*)\ V\ (*)\ W)$
<proof>

end

lemma (**in** *FGModule*) *iso-in-list-imp-iso-in-remisodups* :
 $\exists U \in set\ Us. isomorphic\ (*)\ U$
 $\implies \exists U \in set\ (ActingGroup.remisodups\ Us). isomorphic\ (*)\ U$
<proof>

lemma (**in** *IrrFinGroupRepresentation*) *iso-to-GIrrRep-rep* :
 $\exists U \in ActingGroup.GIrrRep\text{-}repset. isomorphic\ (*)\ U$
<proof>

theorem (**in** *Group*) *iso-class-reps* :
defines $GIRRS \equiv GIrrRep\text{-}repset :: ('f::field, 'g) \text{ aezfun set set}$
assumes $of\text{-}nat (card\ G) \neq (0::'f)$
shows *finite* $GIRRS$
 $\forall U \in GIRRS. IrrFinGroupRepresentation\ G\ (*)\ U$
 $\bigwedge U\ W. \llbracket U \in GIRRS; W \in GIRRS; U \neq W \rrbracket$
 $\implies \neg (FGModule.isomorphic\ G\ (*)\ U\ (*)\ W)$
 $\bigwedge fgsmult\ V. IrrFinGroupRepresentation\ G\ fgsmult\ V$
 $\implies \exists U \in GIRRS. FGModule.isomorphic\ G\ fgsmult\ V\ (*)\ U$
<proof>

6.7 Induced representations

6.7.1 Locale and basic facts

locale *InducedFinGroupRepresentation = Supgroup?: Group H*
+ *BaseRep?: FinGroupRepresentation G smult V*
+ *induced-smult?: aezfun-scalar-mult rrsmult*
for $H \quad :: 'g::group\text{-}add\ set$
and $G \quad :: 'g\ set$
and $smult \quad :: ('f::field, 'g) \text{ aezfun} \Rightarrow 'v::ab\text{-}group\text{-}add \Rightarrow 'v\ (\mathbf{infixl}\ \cdot\ 70)$
and $V \quad :: 'v\ set$
and $rrsmult \quad :: ('f, 'g) \text{ aezfun} \Rightarrow (('f, 'g) \text{ aezfun} \Rightarrow 'v)$
 $\implies (('f, 'g) \text{ aezfun} \Rightarrow 'v)\ (\mathbf{infixl}\ \alpha\ 70)$
+ **fixes** $FH \quad :: ('f, 'g) \text{ aezfun set}$
and $indV \quad :: (('f, 'g) \text{ aezfun} \Rightarrow 'v) \text{ set}$
defines $FH \quad : FH \equiv Supgroup.group\text{-}ring$
and $indV \quad : indV \equiv BaseRep.indspace\ G\ FH\ V$
assumes $rrsmult \quad : rrsmult = Ring1.rightreg\text{-}scalar\text{-}mult\ FH$
and $good\text{-}ordSupgrp: of\text{-}nat (card\ H) \neq (0::'f)$
and $Subgroup \quad : Supgroup.Subgroup\ G$

sublocale *InducedFinGroupRepresentation < InducedFHModule*

<proof>

context *InducedFinGroupRepresentation*
begin

abbreviation *ordH* :: 'f **where** *ordH* \equiv *of-nat* (card *H*)
abbreviation *is-Vfbasis* \equiv *fbasis-for* *V*
abbreviation *GRepHomSet* \equiv *FGModuleHomSet* *G* *smult* *V*
abbreviation *HRepHom* \equiv *FGModuleHom* *H* *rrsmult* *indV*
abbreviation *HRepHomSet* \equiv *FGModuleHomSet* *H* *rrsmult* *indV*

lemma *finiteSupgroup*: *finite* *H*
<proof>

lemma *FinGroupSupgroup* : *FinGroup* *H*
<proof>

lemmas *fVectorSpace* = *fVectorSpace*
lemmas *FinDimVectorSpace* = *FinDimVectorSpace*
lemmas *ex-rcoset-replist-hd0*
= *FinGroup.ex-rcoset-replist-hd0*[*OF FinGroupSupgroup Subgroup*]

end

6.7.2 A basis for the induced space

context *InducedFinGroupRepresentation*
begin

abbreviation *negHorbit-list* \equiv *induced-smult.negGorbit-list*

lemmas *ex-rcoset-replist*
= *FinGroup.ex-rcoset-replist*[*OF FinGroupSupgroup Subgroup*]
lemmas *length-negHorbit-list* = *induced-smult.length-negGorbit-list*
lemmas *length-negHorbit-list-sublist* = *induced-smult.length-negGorbit-list-sublist*
lemmas *negHorbit-list-indV* = *FGModule.negGorbit-list-V*[*OF FHModule-indspace*]

lemma *fincomb-Horbit-induced-veclist-reduce* :

fixes *vs* :: 'v list
and *hs* :: 'g list
defines *hfuss* : *hfuss* \equiv *negHorbit-list* *hs* *induced-vector* *vs*
assumes *vs* : *set* *vs* \subseteq *V* **and** *i*: *i* < *length* *hs*
and *scalars* : *list-all2* (λ *rs ms. length* *rs* = *length* *ms*) *css* *hfuss*
and *rcoset-reps* : *Supgroup.is-rcoset-replist* *G* *hs*
shows ((*concat* *css*) \cdot \boxtimes (*concat* *hfuss*)) (1 δ δ (*hs*!*i*)) = (*css*!*i*) \cdot $\#$ *vs*
<proof>

lemma *indspace-fspanning-set* :

```

fixes vs      :: 'v list
and   hs      :: 'g list
defines hfvs   : hfvs ≡ negHorbit-list hs induced-vector vs
assumes base-spset : set vs ⊆ V V = BaseRep.fSpan vs
and   rcoset-reps : Supgroup.is-rcoset-replist G hs
shows indV = induced-smult.fSpan (concat hfvs)
⟨proof⟩

lemma indspace-basis :
fixes vs      :: 'v list
and   hs      :: 'g list
defines hfvs   : hfvs ≡ negHorbit-list hs induced-vector vs
assumes base-spset : BaseRep.fbasis-for V vs
and   rcoset-reps : Supgroup.is-rcoset-replist G hs
shows fscalar-mult.basis-for induced-smult.fsmult indV (concat hfvs)
⟨proof⟩

lemma induced-vector-decomp :
fixes vs      :: 'v list
and   hs      :: 'g list
and   cs      :: 'f list
defines hfvs   : hfvs ≡ negHorbit-list (0#hs) induced-vector vs
and   extrazeros : extrazeros ≡ replicate ((length hs)*(length vs)) 0
assumes base-spset : BaseRep.fbasis-for V vs
and   rcoset-reps : Supgroup.is-rcoset-replist G (0#hs)
and   cs          : length cs = length vs
and   v           : v = cs ·#· vs
shows induced-vector v = (cs@extrazeros) ·⊗⊗ (concat hfvs)
⟨proof⟩

end

```

6.7.3 The induced space is a representation space

```

context InducedFinGroupRepresentation
begin

```

```

lemma indspace-findim :
  fscalar-mult.findim induced-smult.fsmult indV
⟨proof⟩

```

```

theorem FinGroupRepresentation-indspace :
  FinGroupRepresentation H rrsmult indV
⟨proof⟩

```

```

end

```

6.8 Frobenius reciprocity

6.8.1 Locale and basic facts

There are a number of defined objects and lemmas concerning those objects leading up to the theorem of Frobenius reciprocity, so we create a locale to contain it all.

```

locale FrobeniusReciprocity
= GRep?: InducedFinGroupRepresentation H G smult V rrsmult
+ HRep?: FinGroupRepresentation H smult' W
  for H      :: 'g::group-add set
  and G      :: 'g set
  and smult  :: ('f::field, 'g) aezfun  $\Rightarrow$  'v::ab-group-add  $\Rightarrow$  'v (infixl  $\cdot$  70)
  and V      :: 'v set
  and rrsmult :: ('f,'g) aezfun  $\Rightarrow$  (('f,'g) aezfun  $\Rightarrow$  'v)
                     $\Rightarrow$  (('f,'g) aezfun  $\Rightarrow$  'v) (infixl  $\bowtie$  70)
  and smult' :: ('f, 'g) aezfun  $\Rightarrow$  'w::ab-group-add  $\Rightarrow$  'w (infixr  $\star$  70)
  and W      :: 'w set
begin

abbreviation fsmult'  :: 'f  $\Rightarrow$  'w  $\Rightarrow$  'w          (infixr  $\sharp\star$  70)
  where fsmult'  $\equiv$  HRep.fsmult
abbreviation flincomb' :: 'f list  $\Rightarrow$  'w list  $\Rightarrow$  'w (infixr  $\cdot\sharp\star$  70)
  where flincomb'  $\equiv$  HRep.flincomb
abbreviation Hmult'   :: 'g  $\Rightarrow$  'w  $\Rightarrow$  'w          (infixr  $\star\star$  70)
  where Hmult'  $\equiv$  HRep.Gmult

definition Tsmult1 ::
  'f  $\Rightarrow$  (((('f, 'g) aezfun  $\Rightarrow$  'v) $\Rightarrow$ 'w)  $\Rightarrow$  (((('f, 'g) aezfun  $\Rightarrow$  'v) $\Rightarrow$ 'w) (infixr  $\star\bowtie$  70)
  where Tsmult1  $\equiv$   $\lambda a T. \lambda f. a \sharp\star (T f)$ 

definition Tsmult2 :: 'f  $\Rightarrow$  ('v $\Rightarrow$ 'w)  $\Rightarrow$  ('v $\Rightarrow$ 'w) (infixr  $\star$  70)
  where Tsmult2  $\equiv$   $\lambda a T. \lambda v. a \sharp\star (T v)$ 

lemma FHModuleW : FGModule H ( $\star$ ) W  $\langle$ proof $\rangle$ 

lemma FGModuleW: FGModule G smult' W
 $\langle$ proof $\rangle$ 

In order to build an inverse for the required isomorphism of Hom-sets, we
will need a basis for the induced  $H$ -space.

definition Vfbasis :: 'v list where Vfbasis  $\equiv$  (SOME vs. is-Vfbasis vs)

lemma Vfbasis : is-Vfbasis Vfbasis
 $\langle$ proof $\rangle$ 

lemma Vfbasis-V : set Vfbasis  $\subseteq$  V
 $\langle$ proof $\rangle$ 

```

definition *nonzero-H-rcoset-reps* :: 'g list
where *nonzero-H-rcoset-reps* \equiv (SOME hs. Group.is-rcoset-replist H G (0#hs))

definition *H-rcoset-reps* :: 'g list **where** *H-rcoset-reps* \equiv 0 # nonzero-H-rcoset-reps

lemma *H-rcoset-reps* : Group.is-rcoset-replist H G *H-rcoset-reps*
⟨proof⟩

lemma *H-rcoset-reps-H* : set *H-rcoset-reps* \subseteq H
⟨proof⟩

lemma *nonzero-H-rcoset-reps-H* : set *nonzero-H-rcoset-reps* \subseteq H
⟨proof⟩

abbreviation *negHorbit-homVfbasis* :: ('v \Rightarrow 'w) \Rightarrow 'w list list
where *negHorbit-homVfbasis* T \equiv HRep.negGorbit-list *H-rcoset-reps* T *Vfbasis*

lemma *negHorbit-Hom-indVfbasis-W* :
T ' V \subseteq W \implies set (concat (negHorbit-homVfbasis T)) \subseteq W
⟨proof⟩

lemma *negHorbit-HomSet-indVfbasis-W* :
T \in GRepHomSet smult' W \implies set (concat (negHorbit-homVfbasis T)) \subseteq W
⟨proof⟩

definition *indVfbasis* :: (('f, 'g) aezfun \Rightarrow 'v) list list
where *indVfbasis* \equiv GRep.negHorbit-list *H-rcoset-reps* induced-vector *Vfbasis*

lemma *indVfbasis* :
fscalar-mult.basis-for induced-smult.fsmult indV (concat *indVfbasis*)
⟨proof⟩

lemma *indVfbasis-indV* : hfvs \in set *indVfbasis* \implies set hfvs \subseteq indV
⟨proof⟩

end

6.8.2 The required isomorphism of Hom-sets

context *FrobeniusReciprocity*
begin

The following function will demonstrate the required isomorphism of Hom-sets (as vector spaces).

definition φ :: (('f, 'g) aezfun \Rightarrow 'v) \Rightarrow 'w) \Rightarrow ('v \Rightarrow 'w)
where $\varphi \equiv$ restrict0 (λ T. T \circ GRep.induced-vector) (HRepHomSet smult' W)

lemma φ -im : φ ' HRepHomSet (\star) W \subseteq GRepHomSet (\star) W
⟨proof⟩

end

6.8.3 The inverse map of Hom-sets

In this section we build an inverse for the required isomorphism, φ .

context *FrobeniusReciprocity*

begin

definition ψ -condition :: ('v \Rightarrow 'w) \Rightarrow (((('f, 'g) aezfun \Rightarrow 'v) \Rightarrow 'w) \Rightarrow bool

where ψ -condition T S

$$\equiv \text{VectorSpaceHom induced-smult.fsmult indV fsmult}' S \\ \wedge \text{map (map S) indVfbasis} = \text{negHorbit-homVfbasis T}$$

lemma *inverse-im-exists'* :

assumes $T \in \text{GRepHomSet } (\star) W$

shows $\exists! S. \text{VectorSpaceHom induced-smult.fsmult indV fsmult}' S$

$$\wedge \text{map S (concat indVfbasis)} = \text{concat (negHorbit-homVfbasis T)}$$

<proof>

lemma *inverse-im-exists* :

assumes $T \in \text{GRepHomSet } (\star) W$

shows $\exists! S. \psi$ -condition T S

<proof>

definition $\psi :: ('v \Rightarrow 'w) \Rightarrow (((('f, 'g) \text{aezfun} \Rightarrow 'v) \Rightarrow 'w)$

where $\psi \equiv \text{restrict0 } (\lambda T. \text{THE } S. \psi\text{-condition T S}) (\text{GRepHomSet } (\star) W)$

lemma $\psi D : T \in \text{GRepHomSet } (\star) W \Longrightarrow \psi$ -condition T (ψ T)

<proof>

lemma ψD -VectorSpaceHom :

$T \in \text{GRepHomSet } (\star) W$

$$\Longrightarrow \text{VectorSpaceHom induced-smult.fsmult indV fsmult}' (\psi T)$$

<proof>

lemma ψD -im :

$T \in \text{GRepHomSet } (\star) W \Longrightarrow \text{map (map } (\psi T)) \text{ indVfbasis}$

$$= \text{aezfun-scalar-mult.negGorbit-list } (\star) \text{H-rcoset-reps T Vfbasis}$$

<proof>

lemma ψD -im-single :

assumes $T \in \text{GRepHomSet } (\star) W$ $h \in \text{set H-rcoset-reps}$ $v \in \text{set Vfbasis}$

shows $\psi T ((- h) * \boxtimes (\text{induced-vector } v)) = (-h) ** (T v)$

<proof>

lemma ψT -W :

assumes $T \in \text{GRepHomSet } (\star) W$

shows $\psi T \text{' indV} \subseteq W$

<proof>

lemma $\psi T\text{-Hmap-on-indVfbasis}$:

assumes $T \in \text{GRepHomSet } (\star) W$

shows $\bigwedge x f. x \in H \implies f \in \text{set } (\text{concat } \text{indVfbasis})$
 $\implies \psi T (x \text{ ** } f) = x \text{ ** } (\psi T f)$

<proof>

lemma $\psi T\text{-hom}$:

assumes $T \in \text{GRepHomSet } (\star) W$

shows $\text{HRepHom } (\star) (\psi T)$

<proof>

lemma $\psi\text{-im} : \psi \text{ ' } \text{GRepHomSet } (\star) W \subseteq \text{HRepHomSet } (\star) W$

<proof>

end

6.8.4 Demonstration of bijectivity

Now we demonstrate that φ is bijective via the inverse ψ .

context *FrobeniusReciprocity*

begin

lemma $\varphi\psi$:

assumes $T \in \text{GRepHomSet } \text{smult}' W$

shows $(\varphi \circ \psi) T = T$

<proof>

lemma $\varphi\text{-inverse-im} : \varphi \text{ ' } \text{HRepHomSet } (\star) W \supseteq \text{GRepHomSet } (\star) W$

<proof>

lemma $\text{bij-}\varphi : \text{bij-betw } \varphi (\text{HRepHomSet } (\star) W) (\text{GRepHomSet } (\star) W)$

<proof>

end

6.8.5 The theorem

Finally we demonstrate that φ is an isomorphism of vector spaces between the two hom-sets, leading to Frobenius reciprocity.

context *FrobeniusReciprocity*

begin

lemma *VectorSpaceIso-}\varphi* :

VectorSpaceIso Tsmult1 (HRepHomSet } (\star) W) Tsmult2 \varphi
(GRepHomSet } (\star) W)

<proof>

theorem *FrobeniusReciprocity* :
 VectorSpace.isomorphic Tsmult1 (HRepHomSet smult' W) Tsmult2
 (GRepHomSet smult' W)
 ⟨*proof*⟩

end

end

7 Bibliography

- [1] S. Axler. *Linear Algebra Done Right*. Undergraduate Texts in Mathematics. Springer, New York, third edition, 2015.
- [2] D. S. Dummit and R. M. Foote. *Abstract Algebra*. John Wiley & Sons, New York, second edition, 1999.
- [3] G. James and M. Liebeck. *Representations and Characters of Groups*. Cambridge Mathematical Textbooks. Cambridge University Press, Cambridge, UK, 1993.