

Representations of Finite Groups

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

May 14, 2024

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>	7
1.3.4 <i>sum-list</i>	7
1.3.5 <i>listset</i>	8
1.4 Functions	10
1.4.1 Miscellaneous facts	10
1.4.2 Support of a function	10
1.4.3 Convolution	11
1.5 Almost-everywhere-zero functions	13

1.5.1	Definition and basic properties	13
1.5.2	Delta (impulse) functions	14
1.5.3	Convolution of almost-everywhere-zero functions	16
1.5.4	Type definition, instantiations, and instances	21
1.5.5	Transfer facts	26
1.5.6	Almost-everywhere-zero functions with constrained support	27
1.6	Polynomials	31
1.7	Algebra of sets	32
1.7.1	General facts	32
1.7.2	Additive independence of sets	33
1.7.3	Inner direct sums	35
2	Groups	36
2.1	Locales and basic facts	36
2.1.1	Locale <i>Group</i> and finite variant <i>FinGroup</i>	36
2.1.2	Abelian variant locale <i>AbGroup</i>	38
2.2	Right cosets	39
2.3	Group homomorphisms	44
2.3.1	Preliminaries	44
2.3.2	Locales	44
2.3.3	Basic facts	45
2.3.4	Basic facts about endomorphisms	48
2.3.5	Basic facts about isomorphisms	49
2.3.6	Hom-sets	50
2.4	Facts about collections of groups	51
2.5	Inner direct sums of Abelian groups	53
2.5.1	General facts	53
2.5.2	Element decomposition and projection	57
2.6	Rings	60
2.6.1	Preliminaries	60
2.6.2	Locale and basic facts	60
2.7	The group ring	61
2.7.1	Definition and basic facts	61
2.7.2	Projecting almost-everywhere-zero functions onto a group ring	63
3	Modules	65
3.1	Locales and basic facts	65
3.1.1	Locales	65
3.1.2	Basic facts	66
3.1.3	Module and submodule instances	69
3.2	Linear algebra in modules	70
3.2.1	Linear combinations: <i>lincomb</i>	70

3.2.2	Spanning: $R\text{Span}$ and Span	75
3.2.3	Finitely generated modules	81
3.2.4	R -linear independence	82
3.2.5	Linear independence over $UNIV$	89
3.2.6	Rank	90
3.3	Module homomorphisms	91
3.3.1	Locales	91
3.3.2	Basic facts	92
3.3.3	Basic facts about endomorphisms	95
3.3.4	Basic facts about isomorphisms	96
3.4	Inner direct sums of RModules	97
4	Vector Spaces	98
4.1	Locales and basic facts	98
4.2	Linear algebra in vector spaces	99
4.2.1	Linear independence and spanning	99
4.2.2	Basis for a vector space: <i>basis-for</i>	101
4.3	Finite dimensional spaces	106
4.4	Vector space homomorphisms	109
4.4.1	Locales	109
4.4.2	Basic facts	110
4.4.3	Hom-sets	113
4.4.4	Basic facts about endomorphisms	116
4.4.5	Polynomials of endomorphisms	119
4.4.6	Existence of eigenvectors of endomorphisms of finite-dimensional vector spaces	126
5	Modules Over a Group Ring	127
5.1	Almost-everywhere-zero functions as scalars	127
5.2	Locale and basic facts	128
5.3	Modules over a group ring as a vector spaces	132
5.4	Homomorphisms of modules over a group ring	133
5.4.1	Locales	133
5.4.2	Basic facts	134
5.4.3	Basic facts about endomorphisms	139
5.4.4	Basic facts about isomorphisms	140
5.4.5	Hom-sets	142
5.5	Induced modules	146
5.5.1	Additive function spaces	146
5.5.2	Spaces of functions which transform under scalar multiplication by almost-everywhere-zero functions	147
5.5.3	General induced spaces of functions on a group ring	147
5.5.4	The right regular action	149
5.5.5	Locale and basic facts	151

6	Representations of Finite Groups	157
6.1	Locale and basic facts	157
6.2	Irreducible representations	159
6.3	Maschke's theorem	160
6.3.1	Averaged projection onto a G -subspace	160
6.3.2	The theorem	164
6.3.3	Consequence: complete reducibility	165
6.3.4	Consequence: decomposition relative to a homomorphism	168
6.4	Schur's lemma	169
6.5	The group ring as a representation space	170
6.5.1	The group ring is a representation space	170
6.5.2	Irreducible representations are constituents of the group ring	172
6.6	Isomorphism classes of irreducible representations	174
6.7	Induced representations	178
6.7.1	Locale and basic facts	178
6.7.2	A basis for the induced space	179
6.7.3	The induced space is a representation space	185
6.8	Frobenius reciprocity	185
6.8.1	Locale and basic facts	185
6.8.2	The required isomorphism of Hom-sets	187
6.8.3	The inverse map of Hom-sets	188
6.8.4	Demonstration of bijectivity	194
6.8.5	The theorem	196
7	Bibliography	197

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
 using *assms*
 by *fast*

1.2 Sets

lemma *empty-set-diff-single* : $A - \{x\} = \{\} \implies A = \{\} \vee A = \{x\}$
 by *auto*

lemma *seteqI* : $(\bigwedge a. a \in A \implies a \in B) \implies (\bigwedge b. b \in B \implies b \in A) \implies A = B$
 using *subset-antisym subsetI* **by** *fast*

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

lemma *good-card-imp-finite* : $of_nat (card A) \neq (0::'a::semiring-1) \implies finite A$
 using *card-ge-0-finite[of A]* **by** *fastforce*

1.3 Lists

1.3.1 zip

lemma *zip-truncate-left* : $zip\ xs\ ys = zip\ (take\ (length\ ys)\ xs)\ ys$
 by (*induct xs ys rule:list-induct2'*) *auto*

lemma *zip-truncate-right* : $zip\ xs\ ys = zip\ xs\ (take\ (length\ xs)\ ys)$
 by (*induct xs ys rule:list-induct2'*) *auto*

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

lemma *zip-append-left* :
 $zip\ (xs@ys)\ zs = zip\ xs\ zs @ zip\ ys\ (drop\ (length\ xs)\ zs)$
 using *zip-append1 zip-truncate-right[of xs zs]* **by** *simp*

lemma *zip-append-right* :

$zip\ xs\ (ys@zs) = zip\ xs\ ys\ @\ zip\ (drop\ (length\ ys)\ xs)\ zs$
using $zip-append2\ zip-truncate-left[of\ xs\ ys]$ **by** $simp$

lemma $length-concat-map-split-zip$:
 $length\ [f\ x\ y.\ (x,y)\leftarrow zip\ xs\ ys] = min\ (length\ xs)\ (length\ ys)$
by $(induct\ xs\ ys\ rule:\ list-induct2')$ $auto$

lemma $concat-map-split-eq-map-split-zip$:
 $[f\ x\ y.\ (x,y)\leftarrow zip\ xs\ ys] = map\ (case-prod\ f)\ (zip\ xs\ ys)$
by $(induct\ xs\ ys\ rule:\ list-induct2')$ $auto$

lemma $set-zip-map2$:
 $(a,z) \in set\ (zip\ xs\ (map\ f\ ys)) \implies \exists b.\ (a,b) \in set\ (zip\ xs\ ys) \wedge z = f\ b$
by $(induct\ xs\ ys\ rule:\ list-induct2')$ $auto$

1.3.2 concat

lemma $concat-eq$:
 $list-all2\ (\lambda xs\ ys.\ length\ xs = length\ ys)\ xss\ yss \implies concat\ xss = concat\ yss$
 $\implies xss = yss$
by $(induct\ xss\ yss\ rule:\ list-all2-induct)$ $auto$

lemma $match-concat$:
fixes $bss :: 'b\ list\ list$
defines $eq-len \equiv \lambda xs\ ys.\ length\ xs = length\ ys$
shows $\forall as :: 'a\ list.\ length\ as = length\ (concat\ bss)$
 $\longrightarrow (\exists css :: 'a\ list\ list.\ as = concat\ css \wedge list-all2\ eq-len\ css\ bss)$

proof $(induct\ bss)$
from $eq-len$
show $\forall as.\ length\ as = length\ (concat\ [])$
 $\longrightarrow (\exists css.\ as = concat\ css \wedge list-all2\ eq-len\ css\ [])$
by $simp$

next
fix $fs :: 'b\ list$ **and** $fss :: 'b\ list\ list$
assume $prevcase:\ \forall as.\ length\ as = length\ (concat\ fss)$
 $\longrightarrow (\exists css.\ as = concat\ css \wedge list-all2\ eq-len\ css\ fss)$
have $\bigwedge as.\ length\ as = length\ (concat\ (fs\ \# fss))$
 $\implies (\exists css.\ as = concat\ css \wedge list-all2\ eq-len\ css\ (fs\ \# fss))$

proof
fix $as :: 'a\ list$
assume $as:\ length\ as = length\ (concat\ (fs\ \# fss))$
define $xs\ ys$ **where** $xs = take\ (length\ fs)\ as$ **and** $ys = drop\ (length\ fs)\ as$
define gss **where** $gss = (SOME\ css.\ ys = concat\ css \wedge list-all2\ eq-len\ css\ fss)$
define hss **where** $hss = xs\ \#\ gss$
with $xs-def\ ys-def\ as\ gss-def\ eq-len\ prevcase$
show $as = concat\ hss \wedge list-all2\ eq-len\ hss\ (fs\ \# fss)$
using $someI-ex[of\ \lambda css.\ ys = concat\ css \wedge list-all2\ eq-len\ css\ fss]$ **by** $auto$
qed
thus $\forall as.\ length\ as = length\ (concat\ (fs\ \# fss))$

$\longrightarrow (\exists \text{css. } \text{as} = \text{concat } \text{css} \wedge \text{list-all2 } \text{eq-len } \text{css} (\text{fs} \# \text{fss}))$
 by fast
 qed

1.3.3 strip-while

lemma *strip-while-0-nnil* :
 $\text{as} \neq [] \implies \text{set } \text{as} \neq 0 \implies \text{strip-while } ((=) 0) \text{ as} \neq []$
 by (induct as rule: rev-nonempty-induct) auto

1.3.4 sum-list

lemma *const-sum-list* :
 $\forall x \in \text{set } \text{xs. } f x = a \implies \text{sum-list } (\text{map } f \text{ xs}) = a * (\text{length } \text{xs})$
 by (induct xs) auto

lemma *sum-list-prod-cong* :
 $\forall (x,y) \in \text{set } \text{xys. } f x y = g x y$
 $\implies (\sum (x,y) \leftarrow \text{xys. } f x y) = (\sum (x,y) \leftarrow \text{xys. } g x y)$
using *arg-cong*[of map (case-prod f) xys map (case-prod g) xys sum-list] **by**
fastforce

lemma *sum-list-prod-map2* :
 $(\sum (a,y) \leftarrow \text{zip } (\text{map } f \text{ bs}). g a y) = (\sum (a,b) \leftarrow \text{zip } \text{as } \text{bs. } g a (f b))$
by (induct as bs rule: list-induct2') auto

lemma *sum-list-fun-apply* : $(\sum x \leftarrow \text{xs. } f x) y = (\sum x \leftarrow \text{xs. } f x y)$
by (induct xs) auto

lemma *sum-list-prod-fun-apply* : $(\sum (x,y) \leftarrow \text{xys. } f x y) z = (\sum (x,y) \leftarrow \text{xys. } f x y z)$
by (induct xys) auto

lemma (in *comm-monoid-add*) *sum-list-plus* :
 $\text{length } \text{xs} = \text{length } \text{ys}$
 $\implies \text{sum-list } \text{xs} + \text{sum-list } \text{ys} = \text{sum-list } [a+b. (a,b) \leftarrow \text{zip } \text{xs } \text{ys}]$
proof (induct xs ys rule: list-induct2)
case *Cons* **thus** ?*case* **by** (simp add: algebra-simps)
qed *simp*

lemma *sum-list-const-mult-prod* :
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'r :: \text{semiring-0}$
shows $r * (\sum (x,y) \leftarrow \text{xys. } f x y) = (\sum (x,y) \leftarrow \text{xys. } r * (f x y))$
using *sum-list-const-mult*[of r case-prod f] *prod.case-distrib*[of $\lambda x. r * x$ f]
by *simp*

lemma *sum-list-mult-const-prod* :
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'r :: \text{semiring-0}$
shows $(\sum (x,y) \leftarrow \text{xys. } f x y) * r = (\sum (x,y) \leftarrow \text{xys. } (f x y) * r)$
using *sum-list-mult-const*[of case-prod f r] *prod.case-distrib*[of $\lambda x. x * r$ f]
by *simp*

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 (*induct xs arbitrary: n*)
case *Cons* **thus** ?*case* **by** (*cases n*) *auto*
qed *simp*

lemma *sum-list-replicate0* : $sum-list\ (replicate\ n\ 0) = 0$
by (*induct n*) *auto*

1.3.5 listset

lemma *listset-ConsI* : $x \in X \implies xs \in listset\ Xs \implies x\#\!xs \in listset\ (X\#\!Xs)$
unfolding *listset-def set-Cons-def* **by** *simp*

lemma *listset-ConsD* : $x\#\!xs \in listset\ (A\ \#\!As) \implies x \in A \wedge xs \in listset\ As$
unfolding *listset-def set-Cons-def* **by** *auto*

lemma *listset-Cons-conv* :
 $xs \in listset\ (A\ \#\!As) \implies (\exists\ ys.\ y \in A \wedge ys \in listset\ As \wedge xs = y\#\!ys)$
unfolding *listset-def set-Cons-def* **by** *auto*

lemma *listset-length* : $xs \in listset\ Xs \implies length\ xs = length\ Xs$
using *listset-ConsD*
unfolding *listset-def set-Cons-def*
by (*induct xs Xs rule: list-induct2'*) *auto*

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 (*induct As arbitrary: x*)
case *Nil* **hence** $x = (\sum a \leftarrow [].\ a)$ **by** *simp*
moreover **have** $[] \in listset\ []$ **by** *simp*
ultimately **show** ?*case* **by** *fast*
next
case (*Cons A As*)
from *this* **obtain** $a\ as$
where $a-as: a \in A\ as \in listset\ As\ x = (\sum b \leftarrow (a\#\!as).\ b)$
using *set-plus-def[of A]*
by *fastforce*
have $listset\ (A\#\!As) = set-Cons\ A\ (listset\ As)$ **by** *simp*
with $a-as(1,2)$ **have** $a\#\!as \in listset\ (A\#\!As)$ **unfolding** *set-Cons-def* **by** *fast*
with $a-as(3)$ **show** $\exists\ bs \in listset\ (A\#\!As).\ x = (\sum b \leftarrow bs.\ b)$ **by** *fast*
qed

lemma *set-sum-list-element-Cons* :
assumes $x \in (\sum X \leftarrow (A\#\!As).\ X)$
shows $\exists\ a\ as.\ a \in A \wedge as \in listset\ As \wedge x = a + (\sum b \leftarrow as.\ b)$
proof –

from *assms* **obtain** *xs* **where** *xs*: $xs \in \text{listset } (A\#As) \ x = (\sum b \leftarrow xs. b)$
using *set-sum-list-element* **by** *fast*
from *xs(1)* **obtain** *a as* **where** $a \in A \ as \in \text{listset } As \ xs = a \# as$
using *listset-Cons-conv* **by** *fast*
with *xs(2)* **show** *?thesis* **by** *auto*
qed

lemma *sum-list-listset* : $as \in \text{listset } As \implies \text{sum-list } as \in (\sum A \leftarrow As. A)$
proof–
have $\text{length } as = \text{length } As \implies as \in \text{listset } As \implies \text{sum-list } as \in (\sum A \leftarrow As. A)$
proof (*induct as As rule: list-induct2*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons a as A As*) **thus** *?case*
using *listset-ConsD[of a]* *set-plus-def* **by** *auto*
qed
thus $as \in \text{listset } As \implies \text{sum-list } as \in (\sum A \leftarrow As. A)$ **using** *listset-length* **by** *fast*
qed

lemma *listsetI-nth* :
 $\text{length } xs = \text{length } Xs \implies \forall n < \text{length } xs. xs!n \in Xs!n \implies xs \in \text{listset } Xs$
proof (*induct xs Xs rule: list-induct2*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons x xs X Xs*) **thus** $x\#xs \in \text{listset } (X\#Xs)$
using *listset-ConsI[of x X xs Xs]* **by** *fastforce*
qed

lemma *listsetD-nth* : $xs \in \text{listset } Xs \implies \forall n < \text{length } xs. xs!n \in Xs!n$
proof–
have $\text{length } xs = \text{length } Xs \implies xs \in \text{listset } Xs \implies \forall n < \text{length } xs. xs!n \in Xs!n$
proof (*induct xs Xs rule: list-induct2*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons x xs X Xs*)
from *Cons(3)* **have** $x\#xs: x \in X \ xs \in \text{listset } Xs$
using *listset-ConsD[of x]* **by** *auto*
with *Cons(2)* **have** $1: (x\#xs)!0 \in (X\#Xs)!0 \ \forall n < \text{length } xs. xs!n \in Xs!n$
by *auto*
have $\bigwedge n. n < \text{length } (x\#xs) \implies (x\#xs)!n \in (X\#Xs)!n$
proof–
fix *n* **assume** $n < \text{length } (x\#xs)$
with 1 **show** $(x\#xs)!n \in (X\#Xs)!n$ **by** (*cases n*) *auto*
qed
thus $\forall n < \text{length } (x\#xs). (x\#xs)!n \in (X\#Xs)!n$ **by** *fast*
qed
thus $xs \in \text{listset } Xs \implies \forall n < \text{length } xs. xs!n \in Xs!n$ **using** *listset-length* **by** *fast*
qed

lemma *set-listset-el-subset* :
 $xs \in \text{listset } Xs \implies \forall X \in \text{set } Xs. X \subseteq A \implies \text{set } xs \subseteq A$
proof –
have $\llbracket \text{length } xs = \text{length } Xs; xs \in \text{listset } Xs; \forall X \in \text{set } Xs. X \subseteq A \rrbracket$
 $\implies \text{set } xs \subseteq A$
proof (*induct xs Xs rule: list-induct2*)
case *Cons* **thus** ?*case* **using** *listset-ConsD* **by** *force*
qed *simp*
thus $xs \in \text{listset } Xs \implies \forall X \in \text{set } Xs. X \subseteq A \implies \text{set } xs \subseteq A$
using *listset-length* **by** *fast*
qed

1.4 Functions

1.4.1 Miscellaneous facts

lemma *sum-fun-apply* : $\text{finite } A \implies (\sum a \in A. f a) x = (\sum a \in A. f a x)$
by (*induct set: finite*) *auto*

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 (*induct A rule: finite-induct*)
case (*insert a A*)
show $\forall x \in \text{insert } a A. \text{sum } (f x) (\text{insert } a A) = g x$
proof
fix *x* **assume** $x: x \in \text{insert } a A$
show $\text{sum } (f x) (\text{insert } a A) = g x$
proof (*cases x = a*)
case *True*
moreover with *insert(2,4)* **have** $\forall y \in A. f x y = 0$ **by** *simp*
ultimately show ?*thesis* **using** *insert(1,2,4)* **by** *simp*
next
case *False* **with** *x insert* **show** ?*thesis* **by** *simp*
qed
qed
qed *simp*

lemma *distrib-comp-sum-right* : $(T + T') \circ S = (T \circ S) + (T' \circ S)$
by *auto*

1.4.2 Support of a function

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

lemma *suppI*: $f x \neq 0 \implies x \in \text{supp } f$
using *supp-def* **by** *fast*

lemma *suppI-contr*: $x \notin \text{supp } f \implies f x = 0$
using *suppI* **by** *fast*

lemma *suppD*: $x \in \text{supp } f \implies f x \neq 0$
using *supp-def* **by** *fast*

lemma *suppD-contr*: $f x = 0 \implies x \notin \text{supp } f$
using *suppD* **by** *fast*

lemma *zerofun-imp-empty-supp* : $\text{supp } 0 = \{\}$
unfolding *supp-def* **by** *simp*

lemma *supp-zerofun-subset-any* : $\text{supp } 0 \subseteq A$
using *zerofun-imp-empty-supp* **by** *fast*

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

lemma *supp-neg-eq-supp* :
fixes $f :: 'a \Rightarrow 'b::\text{group-add}$
shows $\text{supp } (- f) = \text{supp } f$
unfolding *supp-def*
by *auto*

lemma *supp-diff-subset-union-supp* :
fixes $f g :: 'a \Rightarrow 'b::\text{group-add}$
shows $\text{supp } (f - g) \subseteq \text{supp } f \cup \text{supp } g$
unfolding *supp-def*
by *auto*

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 –
have $\bigwedge a. a \notin A \implies a \notin \text{supp } (f \downarrow A)$ **using** *suppD-contr*[*of f ↓ A*] **by** *simp*
thus *?thesis* **by** *fast*
qed

lemma *bij-betw-restrict0* : $\text{bij-betw } f A B \implies \text{bij-betw } (f \downarrow A) A B$
using *bij-betw-imp-inj-on* *bij-betw-imp-surj-on*
unfolding *bij-betw-def* *inj-on-def*
by *auto*

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 $f = 0 \vee g = 0 \implies \text{convolution } f g = 0$

unfolding *convolution-def*

by *auto*

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

fix $x::'a$

define $c1 c2 i S1 S2$

where $c1 y = (f (x - y)) * g y$

and $c2 y = (f y) * g (-y + x)$

and $i y = -y + x$

and $S1 = \{y. x - y \in \text{supp } f \wedge y \in \text{supp } g\}$

and $S2 = \{y. y \in \text{supp } f \wedge -y + x \in \text{supp } g\}$

for y

have *inj-on* $i S2$ **unfolding** *inj-on-def* **using** *i-def* **by** *simp*

hence $(\sum y \in (i ' S2). c1 y) = (\sum y \in S2. (c1 \circ i) y)$

using *sum.reindex* **by** *fast*

moreover **have** $S1 \text{-} i S2: S1 = i ' S2$

proof (*rule seteqI*)

fix y **assume** $y \text{-} S1: y \in S1$

define z **where** $z = x - y$

hence $y \text{-} eq: -z + x = y$ **by** (*auto simp add: algebra-simps*)

hence $-z + x \in \text{supp } g$ **using** $y \text{-} S1$ *S1-def* **by** *fast*

moreover **have** $z \in \text{supp } f$ **using** $z \text{-} def$ $y \text{-} S1$ *S1-def* **by** *fast*

ultimately **have** $z \in S2$ **using** *S2-def* **by** *fast*

moreover **have** $y = i z$ **using** *i-def* [*abs-def*] $y \text{-} eq$ **by** *fast*

ultimately **show** $y \in i ' S2$ **by** *fast*

next

fix y **assume** $y \in i ' S2$

from *this* **obtain** z **where** $z \text{-} S2: z \in S2$ **and** $y \text{-} eq: y = -z + x$

using *i-def* **by** *fast*

from $y \text{-} eq$ **have** $x - y = z$ **by** (*auto simp add: algebra-simps*)

hence $x - y \in \text{supp } f \wedge y \in \text{supp } g$ **using** $y \text{-} eq$ $z \text{-} S2$ *S2-def* **by** *fastforce*

thus $y \in S1$ **using** *S1-def* **by** *fast*

qed

ultimately **have** $(\sum y \in S1. c1 y) = (\sum y \in S2. (c1 \circ i) y)$ **by** *fast*

with *i-def c1-def c2-def* **have** $(\sum_{y \in S1}. c1\ y) = (\sum_{y \in S2}. c2\ y)$
using *diff-add-eq-diff-diff-swap*[of $x - x$] **by** *simp*
thus *convolution f g x*
 $= (\sum y | y \in \text{supp } f \wedge -y + x \in \text{supp } g. (f\ y) * g\ (-y + x))$
unfolding *S1-def c1-def S2-def c2-def convolution-def* **by** *fast*
qed

lemma *supp-convolution-subset-sum-supp* :
fixes $f\ g :: 'a::\text{group-add} \Rightarrow 'b::\{\text{comm-monoid-add,times}\}$
shows $\text{supp } (\text{convolution } f\ g) \subseteq \text{supp } f + \text{supp } g$
proof –
define *SS* **where** $SS\ x = \{y. x - y \in \text{supp } f \wedge y \in \text{supp } g\}$ **for** x
have $\text{convolution } f\ g = (\lambda x. \text{sum } (\lambda y. (f\ (x - y)) * g\ y) (SS\ x))$
unfolding *SS-def convolution-def* **by** *fast*
moreover **have** $\bigwedge x. x \notin \text{supp } f + \text{supp } g \implies SS\ x = \{\}$
proof –
have $\bigwedge x. SS\ x \neq \{\} \implies x \in \text{supp } f + \text{supp } g$
proof –
fix $x::'a$ **assume** $SS\ x \neq \{\}$
from *this* **obtain** y **where** $x - y \in \text{supp } f$ **and** $y \in \text{supp } g$
using *SS-def* **by** *fast*
from *this* **obtain** z **where** $z \in \text{supp } f$ **and** $z - y = x$ **by** *fast*
from *z-eq* **have** $x = z + y$ **using** *diff-eq-eq* **by** *fast*
with *z-F y-G* **show** $x \in \text{supp } f + \text{supp } g$ **by** *fast*
qed
thus $\bigwedge x. x \notin \text{supp } f + \text{supp } g \implies SS\ x = \{\}$ **by** *fast*
qed
ultimately **have** $\bigwedge x. x \notin \text{supp } f + \text{supp } g$
 $\implies \text{convolution } f\ g\ x = \text{sum } (\lambda y. (f\ (x - y)) * g\ y) \{\}$
by *simp*
hence $\bigwedge x. x \notin \text{supp } f + \text{supp } g \implies \text{convolution } f\ g\ x = 0$
using *sum.empty* **by** *simp*
thus *?thesis* **unfolding** *supp-def* **by** *fast*
qed

1.5 Almost-everywhere-zero functions

1.5.1 Definition and basic properties

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

lemma *aezfun-setD*: $f \in \text{aezfun-set} \implies \text{finite } (\text{supp } f)$
unfolding *aezfun-set-def* **by** *fast*

lemma *aezfun-setI*: $\text{finite } (\text{supp } f) \implies f \in \text{aezfun-set}$
unfolding *aezfun-set-def* **by** *fast*

lemma *zerofun-is-aezfun* : $0 \in \text{aezfun-set}$
unfolding *supp-def aezfun-set-def* **by** *auto*

lemma *sum-of-aezfun-is-aezfun* :
fixes $f\ g :: 'a \Rightarrow 'b :: \text{monoid-add}$
shows $f \in \text{aezfun-set} \Longrightarrow g \in \text{aezfun-set} \Longrightarrow f + g \in \text{aezfun-set}$
using *supp-sum-subset-union-supp*[of $f\ g$] *finite-subset*[of $- \text{supp } f \cup \text{supp } g$]
unfolding *aezfun-set-def*
by *fastforce*

lemma *neg-of-aezfun-is-aezfun* :
fixes $f :: 'a \Rightarrow 'b :: \text{group-add}$
shows $f \in \text{aezfun-set} \Longrightarrow -f \in \text{aezfun-set}$
using *supp-neg-eq-supp*[of f]
unfolding *aezfun-set-def*
by *simp*

lemma *diff-of-aezfun-is-aezfun* :
fixes $f\ g :: 'a \Rightarrow 'b :: \text{group-add}$
shows $f \in \text{aezfun-set} \Longrightarrow g \in \text{aezfun-set} \Longrightarrow f - g \in \text{aezfun-set}$
using *supp-diff-subset-union-supp*[of $f\ g$] *finite-subset*[of $- \text{supp } f \cup \text{supp } g$]
unfolding *aezfun-set-def*
by *fastforce*

lemma *restrict-and-extend0-aezfun-is-aezfun* :
assumes $f \in \text{aezfun-set}$
shows $f \downarrow A \in \text{aezfun-set}$
proof (*rule aezfun-setI*)
have $\bigwedge a. a \notin \text{supp } f \cap A \Longrightarrow a \notin \text{supp } (f \downarrow A)$
proof –
fix a **assume** $a \notin \text{supp } f \cap A$
thus $a \notin \text{supp } (f \downarrow A)$ **using** *suppI-contr*[of a] *suppD-contr*[of $f \downarrow A\ a$]
by (*cases a ∈ A*) *auto*
qed
with *assms* **show** *finite* (*supp* ($f \downarrow A$))
using *aezfun-setD* *finite-subset*[of *supp* ($f \downarrow A$)] **by** *auto*
qed

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

lemma *deltafun-apply-eq* : $(b \delta a) a = b$
unfolding *deltafun-def* **by** *simp*

lemma *deltafun-apply-neg* : $x \neq a \Longrightarrow (b \delta a) x = 0$
unfolding *deltafun-def* **by** *simp*

lemma *deltafun0* : $0 \delta a = 0$
unfolding *deltafun-def* **by** *auto*

lemma *deltafun-plus* :
fixes $b\ c :: 'b::\text{monoid-add}$
shows $(b+c) \delta a = (b \delta a) + (c \delta a)$
unfolding *deltafun-def*
by *auto*

lemma *supp-delta0fun* : $\text{supp } (0 \delta a) = \{\}$
unfolding *supp-def deltafun-def* **by** *simp*

lemma *supp-deltafun* : $b \neq 0 \implies \text{supp } (b \delta a) = \{a\}$
unfolding *supp-def deltafun-def* **by** *simp*

lemma *deltafun-is-aezfun* : $b \delta a \in \text{aezfun-set}$
proof (*cases* $b = 0$)
case *True*
hence $\text{supp } (b \delta a) = \{\}$ **using** *supp-delta0fun[of a]* **by** *fast*
thus *?thesis* **unfolding** *aezfun-set-def* **by** *simp*
next
case *False* **thus** *?thesis* **using** *supp-deltafun[of b a]* **unfolding** *aezfun-set-def* **by** *simp*
qed

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

proof (*induct* *rule: finite-induct*)
case *empty* **show** *?case* **unfolding** *supp-def* **by** *auto*
next

case (*insert a A*)
from *insert(3)* **obtain** *as*
where *as: distinct as set as = A*
 $\wedge f :: 'a \Rightarrow 'b. \text{supp } f \subseteq A$
 $\implies \exists \text{ bs. } \text{length bs} = \text{length as} \wedge f = (\sum (b,a) \leftarrow \text{zip bs as. } b \delta a)$

by *fast*
from *as(1,2) insert(2)* **have** *distinct (a#as) set (a#as) = insert a A* **by** *auto*
moreover

have $\wedge f :: 'a \Rightarrow 'b::\text{semiring-1. } \text{supp } f \subseteq \text{insert } a\ A$
 $\implies (\exists \text{ bs. } \text{length bs} = \text{length } (a\#\text{as})$
 $\wedge f = (\sum (b,a) \leftarrow \text{zip bs } (a\#\text{as}). b \delta a)$

proof –
fix $f :: 'a \Rightarrow 'b$ **assume** *supp-f* : $\text{supp } f \subseteq \text{insert } a\ A$
define g **where** $g\ x = (\text{if } x = a \text{ then } 0 \text{ else } f\ x)$ **for** x
have $\text{supp } g \subseteq A$
proof
fix x **assume** $x: x \in \text{supp } g$

with x *supp-f g-def* **have** $x \in \text{insert } a \ A$ **unfolding** *supp-def* **by** *auto*
moreover from x *g-def* **have** $x \neq a$ **unfolding** *supp-def* **by** *auto*
ultimately show $x \in A$ **by** *fast*
qed
with $as(3)$ **obtain** bs
where bs : $\text{length } bs = \text{length } as$ $g = (\sum (b,a) \leftarrow \text{zip } bs \ as. \ b \ \delta \ a)$
by *fast*
from $bs(1)$ **have** $\text{length } ((f \ a) \ \# \ bs) = \text{length } (a \ \# \ as)$ **by** *auto*
moreover from $g\text{-def } bs(2)$ **have** $f = (\sum (b,a) \leftarrow \text{zip } ((f \ a) \ \# \ bs) \ (a \ \# \ as). \ b \ \delta \ a)$
using *deltafun-apply-eq[of f a]* *deltafun-apply-neq[of - a f a]* **by** (*cases*) *auto*
ultimately
show $\exists bs. \ \text{length } bs = \text{length } (a \ \# \ as) \wedge f = (\sum (b,a) \leftarrow \text{zip } bs \ (a \ \# \ as). \ b \ \delta \ a)$
by *fast*
qed
ultimately show *?case* **by** *fast*
qed

1.5.3 Convolution of almost-everywhere-zero functions

lemma *convolution-eq-sum-over-supp-right* :

fixes $g \ f :: 'a::\text{group-add} \Rightarrow 'b::\{\text{comm-monoid-add}, \text{mult-zero}\}$
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
fix $x::'a$
define SS **where** $SS = \{y. \ x - y \in \text{supp } f \wedge y \in \text{supp } g\}$
have *finite* ($\text{supp } g$) **using** *assms* **unfolding** *aezfun-set-def* **by** *fast*
moreover have $SS \subseteq \text{supp } g$ **unfolding** *SS-def* **by** *fast*
moreover have $\bigwedge y. \ y \in \text{supp } g - SS \implies (f \ (x - y)) * g \ y = 0$ **using** *SS-def*
unfolding *supp-def* **by** *auto*
ultimately show $\text{convolution } f \ g \ x = (\sum_{y \in \text{supp } g.} (f \ (x - y)) * g \ y)$
unfolding *convolution-def*
using *SS-def* *sum.mono-neutral-left[of supp g SS $\lambda y. (f \ (x - y)) * g \ y]$
by *fast*
qed*

lemma *convolution-symm-eq-sum-over-supp-left* :

fixes $f \ g :: 'a::\text{group-add} \Rightarrow 'b::\{\text{comm-monoid-add}, \text{mult-zero}\}$
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
fix $x::'a$
define SS **where** $SS = \{y. \ y \in \text{supp } f \wedge -y + x \in \text{supp } g\}$
have *finite* ($\text{supp } f$) **using** *assms* **unfolding** *aezfun-set-def* **by** *fast*
moreover have $SS \subseteq \text{supp } f$ **using** *SS-def* **by** *fast*
moreover have $\bigwedge y. \ y \in \text{supp } f - SS \implies (f \ y) * g \ (-y + x) = 0$
using *SS-def* **unfolding** *supp-def* **by** *auto*
ultimately

have $(\sum y \in SS. (f y) * g (-y + x)) = (\sum y \in \text{supp } f. (f y) * g (-y + x))$
unfolding *convolution-def*
using *SS-def sum.mono-neutral-left*[of *supp f SS* $\lambda y. (f y) * g (-y + x)$]
by *fast*
thus *convolution f g x* = $(\sum y \in \text{supp } f. (f y) * g (-y + x))$
using *SS-def convolution-symm*[of *f g*] **by** *simp*
qed

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 *convolution (b δ a) f* = $(\lambda x. b * f (-a + x))$
proof (*cases b = 0*)
case *True*
moreover **have** *convolution (b δ a) f = 0*
proof–
from *True* **have** *convolution (b δ a) f = convolution 0 f*
using *deltafun0*[of *a*] *arg-cong*[of *0 δ a 0::'a \Rightarrow 'b*]
by (*simp add: <0 δ a = 0> <b = 0>*)
thus *?thesis* **using** *convolution-zero* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*

next
case *False* **thus** *?thesis*
using *deltafun-is-aezfun*[of *b a*] *convolution-symm-eq-sum-over-supp-left*
supp-deltafun[of *b a*] *deltafun-apply-eq*[of *b a*]
by *fastforce*
qed

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 *convolution f (b δ a)* = $(\lambda x. f (x - a) * b)$
proof (*cases b = 0*)
case *True*
moreover **have** *convolution f (b δ a) = 0*
proof–
from *True* **have** *convolution f (b δ a) = convolution f 0*
using *deltafun0*[of *a*] *arg-cong*[of *0 δ a 0::'a \Rightarrow 'b*]
by (*simp add: <0 δ a = 0>*)
thus *?thesis* **using** *convolution-zero* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*

next
case *False* **thus** *?thesis*
using *deltafun-is-aezfun*[of *b a*] *convolution-eq-sum-over-supp-right*
supp-deltafun[of *b a*] *deltafun-apply-eq*[of *b a*]
by *fastforce*

qed

lemma *convolution-delta-delta* :
 fixes $b1\ b2 :: 'b::\{comm-monoid-add,mult-zero\}$
 and $a1\ a2 :: 'a::group-add$
 shows $convolution\ (b1\ \delta\ a1)\ (b2\ \delta\ a2) = (b1 * b2)\ \delta\ (a1 + a2)$
proof
 fix $x::'a$
 have 1: $convolution\ (b1\ \delta\ a1)\ (b2\ \delta\ a2)\ x = (b1\ \delta\ a1)\ (x - a2) * b2$
 using *convolution-delta-right*[of $b1\ \delta\ a1$] **by** *simp*
 show $convolution\ (b1\ \delta\ a1)\ (b2\ \delta\ a2)\ x = ((b1 * b2)\ \delta\ (a1 + a2))\ x$
 proof (*cases* $x = a1 + a2$)
 case *True*
 hence $x - a2 = a1$ **by** (*simp add: algebra-simps*)
 with 1 **have** $convolution\ (b1\ \delta\ a1)\ (b2\ \delta\ a2)\ x = b1 * b2$
 using *deltafun-apply-eq*[of $b1\ a1$] **by** *simp*
 with *True* **show** *?thesis*
 using *deltafun-apply-eq*[of $b1 * b2\ a1 + a2$] **by** *simp*
 next
 case *False*
 hence $x - a2 \neq a1$ **by** (*simp add: algebra-simps*)
 with 1 **have** $convolution\ (b1\ \delta\ a1)\ (b2\ \delta\ a2)\ x = 0$
 using *deltafun-apply-neq*[of $x - a2\ a1\ b1$] **by** *simp*
 with *False* **show** *?thesis* using *deltafun-apply-neq* **by** *simp*
 qed
qed

lemma *convolution-of-aezfun-is-aezfun* :
 fixes $f\ g :: 'a::group-add \Rightarrow 'b::\{comm-monoid-add,times\}$
 shows $f \in aezfun-set \Longrightarrow g \in aezfun-set \Longrightarrow convolution\ f\ g \in aezfun-set$
 using *supp-convolution-subset-sum-supp*[of $f\ g$]
 finite-set-plus[of *supp* f *supp* g] *finite-subset*
 unfolding *aezfun-set-def*
 by *fastforce*

lemma *convolution-assoc* :
 fixes $f\ h\ g :: 'a::group-add \Rightarrow 'b::semiring-0$
 assumes $f\text{-aez}: f \in aezfun-set$ and $h\text{-aez}: h \in aezfun-set$
 shows $convolution\ (convolution\ f\ g)\ h = convolution\ f\ (convolution\ g\ h)$
proof
 define $fg\ gh$ where $fg = convolution\ f\ g$ and $gh = convolution\ g\ h$
 fix $x::'a$
 have $convolution\ fg\ h\ x$
 = $(\sum y \in supp\ f. (\sum z \in supp\ h. f\ y * g\ (-y + x - z) * h\ z))$
 proof –
 have $convolution\ fg\ h\ x = (\sum z \in supp\ h. fg\ (x - z) * h\ z)$
 using *h-aez convolution-eq-sum-over-supp-right*[of $h\ fg$] **by** *simp*
 moreover **have** $\bigwedge z. fg\ (x - z) * h\ z$
 = $(\sum y \in supp\ f. f\ y * g\ (-y + x - z) * h\ z)$

proof-
fix $z::'a$
have $fg (x - z) = (\sum y \in \text{supp } f. f y * g (-y + (x - z)))$
using *fg-def f-aez convolution-symm-eq-sum-over-supp-left* **by** *fastforce*
hence $fg (x - z) * h z = (\sum y \in \text{supp } f. f y * g (-y + (x - z)) * h z)$
using *sum-distrib-right* **by** *simp*
thus $fg (x - z) * h z = (\sum y \in \text{supp } f. f y * g (-y + x - z) * h z)$
by (*simp add: algebra-simps*)
qed
ultimately
have *convolution fg h x*
 $= (\sum z \in \text{supp } h. (\sum y \in \text{supp } f. f y * g (-y + x - z) * h z))$
using *sum.cong*
by *simp*
thus *?thesis using sum.swap by simp*
qed
moreover have *convolution f gh x*
 $= (\sum y \in \text{supp } f. (\sum z \in \text{supp } h. f y * g (-y + x - z) * h z))$
proof-
have *convolution f gh x = (\sum y \in \text{supp } f. f y * gh (-y + x))*
using *f-aez convolution-symm-eq-sum-over-supp-left[of f gh]* **by** *simp*
moreover have $\bigwedge y. f y * gh (-y + x)$
 $= (\sum z \in \text{supp } h. f y * g (-y + x - z) * h z)$
proof-
fix $y::'a$
have *triple-cong: $\bigwedge z. f y * (g (-y + x - z) * h z)$*
 $= f y * g (-y + x - z) * h z$
using *mult.assoc[of f y]* **by** *simp*
have $gh (-y + x) = (\sum z \in \text{supp } h. g (-y + x - z) * h z)$
using *gh-def h-aez convolution-eq-sum-over-supp-right* **by** *fastforce*
hence $f y * gh (-y + x) = (\sum z \in \text{supp } h. f y * (g (-y + x - z) * h z))$
using *sum-distrib-left* **by** *simp*
also have $\dots = (\sum z \in \text{supp } h. f y * g (-y + x - z) * h z)$
using *triple-cong sum.cong* **by** *simp*
finally
show $f y * gh (-y + x) = (\sum z \in \text{supp } h. f y * g (-y + x - z) * h z)$
by *fast*
qed
ultimately show *?thesis using sum.cong by simp*
qed
ultimately show *convolution fg h x = convolution f gh x by simp*
qed

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
define $gh \ GH$ **where** $gh = g + h$ **and** $GH = \text{supp } g \cup \text{supp } h$

have *fin-GH*: *finite GH using GH-def assms unfolding aezfun-set-def by fast*
have *gh-aezfun*: $gh \in \text{aezfun-set}$ **using** *gh-def assms sum-of-aezfun-is-aezfun by fast*
fix *x*::'a
have *zero-ext-g* : $\bigwedge y. y \in GH - \text{supp } g \implies (f (x - y)) * g y = 0$
and *zero-ext-h* : $\bigwedge y. y \in GH - \text{supp } h \implies (f (x - y)) * h y = 0$
and *zero-ext-gh*: $\bigwedge y. y \in GH - \text{supp } gh \implies (f (x - y)) * gh y = 0$
unfolding *supp-def* **by** *auto*
have *convolution f gh* $x = (\sum y \in \text{supp } gh. (f (x - y)) * gh y)$
using *assms gh-aezfun convolution-eq-sum-over-supp-right*[of *gh f*] **by** *simp*
also from *gh-def GH-def* **have** $\dots = (\sum y \in GH. (f (x - y)) * gh y)$
using *fin-GH supp-sum-subset-union-supp zero-ext-gh*
 $\text{sum.mono-neutral-left}$ [of *GH supp gh* ($\lambda y. (f (x - y)) * gh y$)]
by *fast*
also from *gh-def*
have $\dots = (\sum y \in GH. (f (x - y)) * g y) + (\sum y \in GH. (f (x - y)) * h y)$
using *sum.distrib* **by** (*simp add: algebra-simps*)
finally show *convolution f gh* $x = (\text{convolution } f g + \text{convolution } f h) x$
using *assms GH-def fin-GH zero-ext-g zero-ext-h*
 $\text{sum.mono-neutral-right}$ [of *GH supp g* ($\lambda y. (f (x - y)) * g y$)]
 $\text{sum.mono-neutral-right}$ [of *GH supp h* ($\lambda y. (f (x - y)) * h y$)]
 $\text{convolution-eq-sum-over-supp-right}$ [of *g f*]
 $\text{convolution-eq-sum-over-supp-right}$ [of *h f*]
by *fastforce*
qed

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
define *fg FG* **where** $fg = f + g$ **and** $FG = \text{supp } f \cup \text{supp } g$
have *fin-FG*: *finite FG using FG-def assms unfolding aezfun-set-def by fast*
have *fg-aezfun*: $fg \in \text{aezfun-set}$ **using** *fg-def assms sum-of-aezfun-is-aezfun by fast*
fix *x*::'a
have *zero-ext-f* : $\bigwedge y. y \in FG - \text{supp } f \implies (f y) * h (-y + x) = 0$
and *zero-ext-g* : $\bigwedge y. y \in FG - \text{supp } g \implies (g y) * h (-y + x) = 0$
and *zero-ext-fg*: $\bigwedge y. y \in FG - \text{supp } fg \implies (fg y) * h (-y + x) = 0$
unfolding *supp-def* **by** *auto*
from *assms* **have** *convolution fg h* $x = (\sum y \in \text{supp } fg. (fg y) * h (-y + x))$
using *fg-aezfun convolution-symm-eq-sum-over-supp-left*[of *fg h*] **by** *simp*
also from *fg-def FG-def* **have** $\dots = (\sum y \in FG. (fg y) * h (-y + x))$
using *fin-FG supp-sum-subset-union-supp zero-ext-fg*
 $\text{sum.mono-neutral-left}$ [of *FG supp fg* ($\lambda y. (fg y) * h (-y + x)$)]
by *fast*
also from *fg-def*
have $\dots = (\sum y \in FG. (f y) * h (-y + x)) + (\sum y \in FG. (g y) * h (-y + x))$
using *sum.distrib* **by** (*simp add: algebra-simps*)

```

finally show convolution fg h x = (convolution f h + convolution g h) x
using assms FG-def fin-FG zero-ext-f zero-ext-g
        sum.mono-neutral-right[of FG supp f ( $\lambda y. (f y) * h (-y + x)$ )]
        sum.mono-neutral-right[of FG supp g ( $\lambda y. (g y) * h (-y + x)$ )]
        convolution-symm-eq-sum-over-suppl-left[of f h]
        convolution-symm-eq-sum-over-suppl-left[of g h]
by fastforce
qed

```

1.5.4 Type definition, instantiations, and instances

```

typedef (overloaded) ('a::zero,'b) aezfun = aezfun-set :: ('b $\Rightarrow$ 'a) set
morphisms aezfun Abs-aezfun
using zerofun-is-aezfun
by fast

```

```

setup-lifting type-definition-aezfun

```

```

lemma aezfun-finite-supp : finite (supp (aezfun a))
using aezfun.aezfun unfolding aezfun-set-def by fast

```

```

lemma aezfun-transfer : aezfun a = aezfun b  $\implies$  a = b by transfer fast

```

```

instantiation aezfun :: (zero, type) zero
begin
  lift-definition zero-aezfun :: ('a,'b) aezfun is 0::'b $\Rightarrow$ 'a
    using zerofun-is-aezfun by fast
  instance ..
end

```

```

lemma zero-aezfun-transfer : Abs-aezfun ((0::'b::zero)  $\delta$  (0::'a::zero)) = 0
proof -
  define zb za where zb = (0::'b) and za = (0::'a)
  hence zb  $\delta$  za = 0 using deltafun0[of za] by fast
  moreover have aezfun 0 = 0 using zero-aezfun.rep-eq by fast
  ultimately have zb  $\delta$  za = aezfun 0 by simp
  with zb-def za-def show ?thesis using aezfun-inverse by simp
qed

```

```

lemma zero-aezfun-apply [simp]: aezfun 0 x = 0
by transfer simp

```

```

instantiation aezfun :: (monoid-add, type) plus
begin
  lift-definition plus-aezfun :: ('a, 'b) aezfun  $\Rightarrow$  ('a, 'b) aezfun  $\Rightarrow$  ('a, 'b) aezfun
    is  $\lambda f g. f + g$ 
    using sum-of-aezfun-is-aezfun
    by auto
  instance ..

```

end

lemma *plus-aezfun-apply* [*simp*]: $aezfun (a+b) x = aezfun a x + aezfun b x$
by *transfer simp*

instance *aezfun* :: (*monoid-add*, *type*) *semigroup-add*

proof

fix *a b c* :: ('*a*, '*b*) *aezfun*

have $aezfun (a + b + c) = aezfun (a + (b + c))$

proof

fix *x*::'*b* **show** $aezfun (a + b + c) x = aezfun (a + (b + c)) x$

using *add.assoc*[*of aezfun a x*] **by** *simp*

qed

thus $a + b + c = a + (b + c)$ **by** *transfer fast*

qed

instance *aezfun* :: (*monoid-add*, *type*) *monoid-add*

proof

fix *a b c* :: ('*a*, '*b*) *aezfun*

show $0 + a = a$ **by** *transfer simp*

show $a + 0 = a$ **by** *transfer simp*

qed

lemma *sum-list-aezfun-apply* [*simp*] :

$aezfun (sum-list as) x = (\sum a \leftarrow as. aezfun a x)$

by (*induct as*) *auto*

lemma *sum-list-map-aezfun-apply* [*simp*] :

$aezfun (\sum a \leftarrow as. f a) x = (\sum a \leftarrow as. aezfun (f a) x)$

by (*induct as*) *auto*

lemma *sum-list-map-aezfun* [*simp*] :

$aezfun (\sum a \leftarrow as. f a) = (\sum a \leftarrow as. aezfun (f a))$

using *sum-list-map-aezfun-apply*[*of f*] *sum-list-fun-apply*[*of aezfun* \circ *f*] **by** *auto*

lemma *sum-list-prod-map-aezfun-apply* :

$aezfun (\sum (x,y) \leftarrow xys. f x y) a = (\sum (x,y) \leftarrow xys. aezfun (f x y) a)$

by (*induct xys*) *auto*

lemma *sum-list-prod-map-aezfun* :

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

using *sum-list-prod-map-aezfun-apply*[*of f*]

sum-list-prod-fun-apply[*of* $\lambda y z. aezfun (f y z)$]

by *auto*

instance *aezfun* :: (*comm-monoid-add*, *type*) *comm-monoid-add*

proof

fix *a b* :: ('*a*, '*b*) *aezfun*

have $aezfun (a + b) = aezfun (b + a)$

```

proof
  fix  $x::'b$  show  $\text{aezfun } (a + b) x = \text{aezfun } (b + a) x$ 
    using  $\text{add.commute}[of \text{aezfun } a x]$  by  $\text{simp}$ 
qed
thus  $a + b = b + a$  by  $\text{transfer fast}$ 
show  $0 + a = a$  by  $\text{simp}$ 
qed

lemma  $\text{sum-aezfun-apply}$  [ $\text{simp}$ ] :
   $\text{finite } A \implies \text{aezfun } (\sum A) x = (\sum a \in A. \text{aezfun } a x)$ 
  by ( $\text{induct set: finite}$ )  $\text{auto}$ 

instantiation  $\text{aezfun} :: (\text{group-add, type}) \text{minus}$ 
begin
  lift-definition  $\text{minus-aezfun} :: ('a, 'b) \text{aezfun} \Rightarrow ('a, 'b) \text{aezfun} \Rightarrow ('a, 'b) \text{aezfun}$ 
    is  $\lambda f g. f - g$ 
    using  $\text{diff-of-aezfun-is-aezfun}$ 
    by  $\text{fast}$ 
  instance ..
end

lemma  $\text{minus-aezfun-apply}$  [ $\text{simp}$ ]:  $\text{aezfun } (a-b) x = \text{aezfun } a x - \text{aezfun } b x$ 
  by  $\text{transfer simp}$ 

instantiation  $\text{aezfun} :: (\text{group-add, type}) \text{uminus}$ 
begin
  lift-definition  $\text{uminus-aezfun} :: ('a, 'b) \text{aezfun} \Rightarrow ('a, 'b) \text{aezfun}$  is  $\lambda f. - f$ 
    using  $\text{neg-of-aezfun-is-aezfun}$  by  $\text{fast}$ 
  instance ..
end

lemma  $\text{uminus-aezfun-apply}$  [ $\text{simp}$ ]:  $\text{aezfun } (-a) x = - \text{aezfun } a x$ 
  by  $\text{transfer simp}$ 

lemma  $\text{aezfun-left-minus}$  [ $\text{simp}$ ] :
  fixes  $a :: ('a::\text{group-add}, 'b) \text{aezfun}$ 
  shows  $- a + a = 0$ 
  by  $\text{transfer simp}$ 

lemma  $\text{aezfun-diff-minus}$  [ $\text{simp}$ ] :
  fixes  $a b :: ('a::\text{group-add}, 'b) \text{aezfun}$ 
  shows  $a - b = a + - b$ 
  by  $\text{transfer auto}$ 

instance  $\text{aezfun} :: (\text{group-add, type}) \text{group-add}$ 
proof
  fix  $a b :: ('a::\text{group-add}, 'b) \text{aezfun}$ 
  show  $- a + a = 0$   $a + - b = a - b$  by  $\text{auto}$ 
qed

```

```

instance aezfun :: (ab-group-add, type) ab-group-add
proof
  fix a b :: ('a::ab-group-add, 'b) aezfun
  show - a + a = 0 by simp
  show a - b = a + - b using aezfun-diff-minus by fast
qed

instantiation aezfun :: ({one,zero}, zero) one
begin
  lift-definition one-aezfun :: ('a,'b) aezfun is 1  $\delta$  0
    using deltafun-is-aezfun by fast
  instance ..
end

lemma one-aezfun-transfer : Abs-aezfun (1  $\delta$  0) = 1
proof -
  define z n where z = (0::'b::zero) and n = (1::'a::{one,zero})
  hence aezfun 1 = n  $\delta$  z using one-aezfun.rep-eq by fast
  hence Abs-aezfun (n  $\delta$  z) = Abs-aezfun (aezfun 1) by simp
  with z-def n-def show ?thesis using aezfun-inverse by simp
qed

lemma one-aezfun-apply [simp]: aezfun 1 x = (1  $\delta$  0) x
  by transfer rule

lemma one-aezfun-apply-eq [simp]: aezfun 1 0 = 1
  using deltafun-apply-eq by simp

lemma one-aezfun-apply-neq [simp]: x  $\neq$  0  $\implies$  aezfun 1 x = 0
  using deltafun-apply-neq by simp

instance aezfun :: (zero-neq-one, zero) zero-neq-one
proof
  have (0::'a)  $\neq$  1 aezfun 0 0 = 0 aezfun (1::('a,'b) aezfun) 0 = 1
    using zero-neq-one one-aezfun-apply-eq by auto
  thus (0::('a,'b) aezfun)  $\neq$  1
    using zero-neq-one one-aezfun-apply-eq
      fun-eq-iff[of aezfun (0::('a,'b) aezfun) aezfun 1]
    by auto
qed

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
    using convolution-of-aezfun-is-aezfun
    by fast
  instance ..

```


end

lemma *convolution-transfer* :

assumes $f \in \text{aezfun-set } g \in \text{aezfun-set}$

shows $\text{Abs-aezfun } (\text{convolution } f g) = \text{Abs-aezfun } f * \text{Abs-aezfun } g$

proof (*rule aezfun-transfer*)

from *assms* **have** $\text{aezfun } (\text{Abs-aezfun } (\text{convolution } f g)) = \text{convolution } f g$

using *convolution-of-aezfun-is-aezfun Abs-aezfun-inverse* **by** *fast*

moreover from *assms*

have $\text{aezfun } (\text{Abs-aezfun } f * \text{Abs-aezfun } g) = \text{convolution } f g$

using *times-aezfun.rep-eq[of Abs-aezfun f] Abs-aezfun-inverse[of f]*
Abs-aezfun-inverse[of g]

by *simp*

ultimately show $\text{aezfun } (\text{Abs-aezfun } (\text{convolution } f g))$

$= \text{aezfun } (\text{Abs-aezfun } f * \text{Abs-aezfun } g)$

by *simp*

qed

instance *aezfun* :: (*comm-monoid-add,mult-zero*}, *group-add*) *mult-zero*

proof

fix $a :: ('a, 'b) \text{aezfun}$

show $0 * a = 0$ **using** *convolution-zero[of - aezfun a]* **by** *transfer fast*

show $a * 0 = 0$ **using** *convolution-zero[of aezfun a]* **by** *transfer fast*

qed

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

proof

fix $a b c :: ('a, 'b) \text{aezfun}$

show $a * b * c = a * (b * c)$

using *convolution-assoc[of aezfun a aezfun c aezfun b]* **by** *transfer*

show $(a + b) * c = a * c + b * c$

using *convolution-distrib-right[of aezfun a aezfun b aezfun c]* **by** *transfer*

show $a * (b + c) = a * b + a * c$

using *convolution-distrib-left[of aezfun b aezfun c aezfun a]* **by** *transfer*

qed

instance *aezfun* :: (*ring, group-add*) *ring ..*

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

proof

fix $a :: ('a, 'b) \text{aezfun}$

show $1 * a = a$

proof –

have $\text{aezfun } (1 * a) = \text{convolution } (1 \delta 0) (\text{aezfun } a)$ **by** *transfer fast*

hence $\text{aezfun } (1 * a) = (\text{aezfun } a)$

using *one-neq-zero convolution-delta-left[of 1 0 aezfun a]* *minus-zero* **by** *simp*

thus $1 * a = a$ **by** *transfer*

qed

show $a * 1 = a$

proof–
have $\text{aezfun } (a * 1) = \text{convolution } (\text{aezfun } a) (1 \ \delta \ 0)$ **by** *transfer fast*
hence $\text{aezfun } (a * 1) = (\text{aezfun } a)$
using *one-neq-zero convolution-delta-right*[of $\text{aezfun } a$] **by** *simp*
thus *?thesis* **by** *transfer*
qed
qed

instance $\text{aezfun} :: (\text{ring-1}, \text{group-add}) \text{ring-1} ..$

1.5.5 Transfer facts

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

lemma $\text{aezdeltafun} : \text{aezfun } (b \ \delta\delta \ a) = b \ \delta \ a$
using *deltafun-is-aezfun*[of $b \ a$] *Abs-aezfun-inverse* **by** *fast*

lemma $\text{aezdeltafun-plus} : (b+c) \ \delta\delta \ a = (b \ \delta\delta \ a) + (c \ \delta\delta \ a)$
using *aezdeltafun*[of $b+c \ a$] *deltafun-plus* *aezdeltafun*[of $b \ a$] *aezdeltafun*[of $c \ a$]
plus-aezfun.rep-eq[of $b \ \delta\delta \ a$]
aezfun-transfer[of $(b+c) \ \delta\delta \ a \ (b \ \delta\delta \ a) + (c \ \delta\delta \ a)$]
by *fastforce*

lemma *times-aezdeltafun-aezdeltafun* :
fixes $b1 \ b2 :: 'b::\{\text{comm-monoid-add}, \text{mult-zero}\}$
shows $(b1 \ \delta\delta \ a1) * (b2 \ \delta\delta \ a2) = (b1 * b2) \ \delta\delta \ (a1 + a2)$
using *deltafun-is-aezfun* *convolution-transfer*[of $b1 \ \delta \ a1$, *THEN sym*]
convolution-delta-delta[of $b1 \ a1 \ b2 \ a2$]
by *fastforce*

lemma *aezfun-restrict-and-extend0* : $(\text{aezfun } x) \downarrow A \in \text{aezfun-set}$
using *aezfun.aezfun restrict-and-extend0-aezfun-is-aezfun*[of $\text{aezfun } x$] **by** *fast*

lemma *aezdeltafun-decomp* :
fixes $b :: 'b::\text{semiring-1}$
shows $b \ \delta\delta \ a = (b \ \delta\delta \ 0) * (1 \ \delta\delta \ a)$
using *convolution-delta-delta*[of $b \ 0 \ 1 \ a$] *deltafun-is-aezfun*[of $b \ 0$]
deltafun-is-aezfun[of $1 \ a$] *convolution-transfer*
by *fastforce*

lemma *aezdeltafun-decomp'* :
fixes $b :: 'b::\text{semiring-1}$
shows $b \ \delta\delta \ a = (1 \ \delta\delta \ a) * (b \ \delta\delta \ 0)$
using *convolution-delta-delta*[of $1 \ a \ b \ 0$] *deltafun-is-aezfun*[of $b \ 0$]
deltafun-is-aezfun[of $1 \ a$] *convolution-transfer*
by *fastforce*

lemma *supp-aezfun1* :

$\text{supp} (\text{aezfun} (1 :: ('a::\text{zero-neq-one}, 'b::\text{zero}) \text{aezfun})) = 0$
using *supp-deltafun*[of 1::'a 0::'b] **by** *transfer simp*

lemma *supp-aezfun-diff* :

$\text{supp} (\text{aezfun} (x - y)) \subseteq \text{supp} (\text{aezfun} x) \cup \text{supp} (\text{aezfun} y)$

proof –

have $\text{supp} (\text{aezfun} (x - y)) = \text{supp} ((\text{aezfun} x) - (\text{aezfun} y))$ **by** *transfer fast*
thus *?thesis* **using** *supp-diff-subset-union-supp* **by** *fast*

qed

lemma *supp-aezfun-times* :

$\text{supp} (\text{aezfun} (x * y)) \subseteq \text{supp} (\text{aezfun} x) + \text{supp} (\text{aezfun} y)$

proof –

have $\text{supp} (\text{aezfun} (x * y)) = \text{supp} (\text{convolution} (\text{aezfun} x) (\text{aezfun} y))$
by *transfer fast*

thus *?thesis* **using** *supp-convolution-subset-sum-supp* **by** *fast*

qed

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-setspan* :: 'a set \Rightarrow ('b::zero, 'a) *aezfun* set

where *aezfun-setspan* A = {x. $\text{supp} (\text{aezfun} x) \subseteq A$ }

lemma *aezfun-setspanD* : $x \in \text{aezfun-setspan} A \Longrightarrow \text{supp} (\text{aezfun} x) \subseteq A$

unfolding *aezfun-setspan-def* **by** *fast*

lemma *aezfun-setspanI* : $\text{supp} (\text{aezfun} x) \subseteq A \Longrightarrow x \in \text{aezfun-setspan} A$

unfolding *aezfun-setspan-def* **by** *fast*

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

assumes *finite* A

shows $\exists \text{as. } \text{distinct } \text{as} \wedge \text{set } \text{as} = A \wedge ($

$\forall x::('b::\text{semiring-1}, 'a) \text{aezfun} \in \text{aezfun-setspan} A.$

$\exists \text{bs. } \text{length } \text{bs} = \text{length } \text{as} \wedge x = (\sum (b,a) \leftarrow \text{zip } \text{bs } \text{as. } b \ \delta \ \delta \ a)$

)

proof –

from *assms* *aezfun-common-supp-spanning-set*'[of A] **obtain** *as*

where *as*: *distinct* *as* *set* *as* = A

$\forall f::'a \Rightarrow 'b. \text{supp } f \subseteq A$

$\longrightarrow (\exists \text{bs. } \text{length } \text{bs} = \text{length } \text{as} \wedge f = (\sum (b,a) \leftarrow \text{zip } \text{bs } \text{as. } b \ \delta \ \delta \ a))$

by *fast*

have $\bigwedge x::('b, 'a) \text{aezfun. } x \in \text{aezfun-setspan} A \Longrightarrow$

$(\exists \text{bs. } \text{length } \text{bs} = \text{length } \text{as} \wedge x = (\sum (b,a) \leftarrow \text{zip } \text{bs } \text{as. } b \ \delta \ \delta \ a))$

proof –

fix *x*::('b, 'a) *aezfun* **assume** $x \in \text{aezfun-setspan} A$

with *as*(\exists) **obtain** *bs*

where bs : $length\ bs = length\ as\ aezfun\ x = (\sum (b,a) \leftarrow zip\ bs\ as.\ b\ \delta\ a)$
using *aezfun-setspanD*
by *fast*
have $\bigwedge b\ a.\ (b,a) \in set\ (zip\ bs\ as) \implies b\ \delta\ a = aezfun\ (b\ \delta\delta\ a)$
proof–
fix $b\ a$ **assume** $(b,a) \in set\ (zip\ bs\ as)$
show $b\ \delta\ a = aezfun\ (b\ \delta\delta\ a)$ **using** *aezdeltafun[of b a]* **by** *simp*
qed
with bs **show** $\exists bs.\ length\ bs = length\ as \wedge x = (\sum (b,a) \leftarrow zip\ bs\ as.\ b\ \delta\delta\ a)$
using *sum-list-prod-cong[of zip bs as deltafun $\lambda b\ a.\ aezfun\ (b\ \delta\delta\ a)$]*
sum-list-prod-map-aezfun[of aezdeltafun zip bs as]
aezfun-transfer[of x]
by *fastforce*
qed
with $as(1,2)$ **show** *?thesis* **by** *fast*
qed

lemma *aezfun-common-supp-spanning-set-decomp* :
fixes $G :: 'g::group-add\ set$
assumes *finite G*
shows $\exists gs.\ distinct\ gs \wedge set\ gs = G \wedge (\forall x::('r::semiring-1,'g)\ aezfun \in aezfun-setspan\ G.\ \exists rs.\ length\ rs = length\ gs \wedge x = (\sum (r,g) \leftarrow zip\ rs\ gs.\ (r\ \delta\delta\ 0) * (1\ \delta\delta\ g))$
proof–
from *aezfun-common-supp-spanning-set[OF assms]* **obtain** gs
where gs : *distinct gs set gs = G*
 $\forall x::('r,'g)\ aezfun \in aezfun-setspan\ G.$
 $\exists rs.\ length\ rs = length\ gs$
 $\wedge x = (\sum (r,g) \leftarrow zip\ rs\ gs.\ r\ \delta\delta\ g)$
by *fast*
have $\bigwedge x::('r,'g)\ aezfun.\ x \in aezfun-setspan\ G$
 $\implies \exists rs.\ length\ rs = length\ gs$
 $\wedge x = (\sum (r,g) \leftarrow zip\ rs\ gs.\ (r\ \delta\delta\ 0) * (1\ \delta\delta\ g))$
proof–
fix $x::('r,'g)\ aezfun$ **assume** $x \in aezfun-setspan\ G$
with $gs(3)$ **obtain** rs
where $length\ rs = length\ gs\ x = (\sum (r,g) \leftarrow zip\ rs\ gs.\ r\ \delta\delta\ g)$
using *aezfun-setspanD*
by *fast*
thus $\exists rs.\ length\ rs = length\ gs$
 $\wedge x = (\sum (r,g) \leftarrow zip\ rs\ gs.\ (r\ \delta\delta\ 0) * (1\ \delta\delta\ g))$
using *aezdeltafun-decomp sum-list-prod-cong[*
*of zip rs gs $\lambda r\ g.\ r\ \delta\delta\ g\ \lambda r\ g.\ (r\ \delta\delta\ 0) * (1\ \delta\delta\ g)$*
 $\left. \right]$
by *auto*
qed
with $gs(1,2)$ **show** *?thesis* **by** *fast*

qed

lemma *aezfun-decomp-aezdeltafun* :

fixes $c :: ('r::\text{semiring-1}, 'a) \text{aezfun}$

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

proof –

from *aezfun-finite-supp*[of c] **obtain** as

where $as: \text{set } as = \text{supp (aezfun c)}$

$\forall x::('r, 'a) \text{aezfun} \in \text{aezfun-setspace (supp (aezfun c))}.$

$\exists bs. \text{length } bs = \text{length } as$

$\wedge x = (\sum (b,a) \leftarrow \text{zip } bs \ as. \ b \ \delta \delta \ a)$

using *aezfun-common-supp-spanning-set*[of supp (aezfun c)]

by *fast*

from $as(2)$ **obtain** bs

where $bs: \text{length } bs = \text{length } as \ c = (\sum (b,a) \leftarrow \text{zip } bs \ as. \ b \ \delta \delta \ a)$

using *aezfun-setspaceI*[of $c \ \text{supp (aezfun c)}$]

by *fast*

from $bs(1) \ as(1)$ **have** $\text{set (map snd (zip } bs \ as)) = \text{supp (aezfun c)}$ **by** *simp*

with $bs(2)$ **show** *?thesis* **by** *fast*

qed

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

fixes $c :: ('r::\text{semiring-1}, 'a) \text{aezfun}$

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

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

using *aezfun-setspaceD* *aezfun-decomp-aezdeltafun*

by *fast*

lemma *aezdelta0fun-commutes'* :

fixes $b1 \ b2 :: 'b::\text{comm-semiring-1}$

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

using *times-aezdeltafun-aezdeltafun*[of $b1 \ a$]

times-aezdeltafun-aezdeltafun[of $b2 \ 0 \ b1 \ a$]

by (*simp add: algebra-simps*)

lemma *aezdelta0fun-commutes* :

fixes $b :: 'b::\text{comm-semiring-1}$

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

proof –

from *aezfun-decomp-aezdeltafun* **obtain** ras

where $c: c = (\sum (r,a) \leftarrow \text{ras. } r \ \delta \delta \ a)$

by *fast*

thus *?thesis*

using *sum-list-mult-const-prod*[of $\lambda r \ a. \ r \ \delta \delta \ a \ \text{ras}$] *aezdelta0fun-commutes'*

sum-list-prod-cong[of $\text{ras } \lambda r \ a. \ r \ \delta \delta \ a * (b \ \delta \delta \ 0) \ \lambda r \ a. \ b \ \delta \delta \ 0 * (r \ \delta \delta \ a)$]

sum-list-const-mult-prod[of $b \ \delta \delta \ 0 \ \lambda r \ a. \ r \ \delta \delta \ a \ \text{ras}$]

by *auto*

qed

The following definition constrains the support of arbitrary almost-everywhere-

zero functions, as a sort of projection onto a *aezfun-sets*pan.

definition *aezfun-sets*pan-proj :: 'a set \Rightarrow ('b::zero,'a) *aezfun* \Rightarrow ('b::zero,'a) *aezfun*
where *aezfun-sets*pan-proj A x \equiv Abs-*aezfun* ((*aezfun* x) \downarrow A)

lemma *aezfun-sets*pan-projD1 :

a \in A \implies *aezfun* (*aezfun-sets*pan-proj A x) a = *aezfun* x a

using *aezfun-restrict-and-extend0*[of A x] Abs-*aezfun-inverse*[of (*aezfun* x) \downarrow A]

unfolding *aezfun-sets*pan-proj-def

by *simp*

lemma *aezfun-sets*pan-projD2 :

a \notin A \implies *aezfun* (*aezfun-sets*pan-proj A x) a = 0

using *aezfun-restrict-and-extend0*[of A x] Abs-*aezfun-inverse*[of (*aezfun* x) \downarrow A]

unfolding *aezfun-sets*pan-proj-def

by *simp*

lemma *aezfun-sets*pan-proj-in-setspan :

*aezfun-sets*pan-proj A x \in *aezfun-sets*pan A

using *aezfun-sets*pan-projD2[of - A]

*suppD-contr*a[of *aezfun* (*aezfun-sets*pan-proj A x)]

*aezfun-sets*panI[of *aezfun-sets*pan-proj A x A]

by *auto*

lemma *aezfun-sets*pan-proj-zero : *aezfun-sets*pan-proj A 0 = 0

proof–

have *aezfun* (*aezfun-sets*pan-proj A 0) = *aezfun* 0

proof

fix a **show** *aezfun* (*aezfun-sets*pan-proj A 0) a = *aezfun* 0 a

using *aezfun-sets*pan-projD1[of a A 0] *aezfun-sets*pan-projD2[of a A 0]

by (cases a \in A) *auto*

qed

thus ?thesis **using** *aezfun-transfer* **by** *fast*

qed

lemma *aezfun-sets*pan-proj-aezdeltafun :

*aezfun-sets*pan-proj A (b $\delta\delta$ a) = (if a \in A then b $\delta\delta$ a else 0)

proof–

have *aezfun* (*aezfun-sets*pan-proj A (b $\delta\delta$ a))

= *aezfun* (if a \in A then b $\delta\delta$ a else 0)

proof

fix x **show** *aezfun* (*aezfun-sets*pan-proj A (b $\delta\delta$ a)) x

= *aezfun* (if a \in A then b $\delta\delta$ a else 0) x

proof (cases x \in A)

case True **thus** ?thesis

using *aezfun-sets*pan-projD1[of x A b $\delta\delta$ a] *aezdeltafun*[of b a]

deltafun-apply-neq[of x]

by *fastforce*

next

case False

hence $aezfun$ ($aezfun$ - $setspan$ - $proj$ A ($b \delta \delta a$)) $x = 0$
using $aezfun$ - $setspan$ - $projD2$ [of x A] **by** $simp$
moreover from $False$
have $a \in A \implies aezfun$ (if $a \in A$ $then$ $b \delta \delta a$ $else$ 0) $x = 0$
using $aezdeltafun$ [of b a] $deltafun$ - $apply$ - neq [of x a b] **by** $auto$
ultimately show $?thesis$ **by** $auto$
qed
qed
thus $?thesis$ **using** $aezfun$ - $transfer$ **by** $fast$
qed

lemma $aezfun$ - $setspan$ - $proj$ - add :

$aezfun$ - $setspan$ - $proj$ A ($x+y$)
 $= aezfun$ - $setspan$ - $proj$ A x + $aezfun$ - $setspan$ - $proj$ A y

proof –

have $aezfun$ ($aezfun$ - $setspan$ - $proj$ A ($x+y$))
 $= aezfun$ ($aezfun$ - $setspan$ - $proj$ A x + $aezfun$ - $setspan$ - $proj$ A y)

proof

fix a **show** $aezfun$ ($aezfun$ - $setspan$ - $proj$ A ($x+y$)) a
 $= aezfun$ ($aezfun$ - $setspan$ - $proj$ A x + $aezfun$ - $setspan$ - $proj$ A y) a
using $aezfun$ - $setspan$ - $projD1$ [of a A $x+y$] $aezfun$ - $setspan$ - $projD2$ [of a A $x+y$]
 $aezfun$ - $setspan$ - $projD1$ [of a A x] $aezfun$ - $setspan$ - $projD1$ [of a A y]
 $aezfun$ - $setspan$ - $projD2$ [of a A x] $aezfun$ - $setspan$ - $projD2$ [of a A y]

by ($cases$ $a \in A$) $auto$

qed

thus $?thesis$ **using** $aezfun$ - $transfer$ **by** $auto$

qed

lemma $aezfun$ - $setspan$ - $proj$ - sum - $list$:

$aezfun$ - $setspan$ - $proj$ A ($\sum x \leftarrow xs. f x$) $= (\sum x \leftarrow xs. aezfun$ - $setspan$ - $proj$ A ($f x$))

proof ($induct$ xs)

case Nil **show** $?case$ **using** $aezfun$ - $setspan$ - $proj$ - $zero$ **by** $simp$

next

case ($Cons$ x xs) **thus** $?case$ **using** $aezfun$ - $setspan$ - $proj$ - add [of A f x] **by** $simp$

qed

lemma $aezfun$ - $setspan$ - $proj$ - sum - $list$ - $prod$:

$aezfun$ - $setspan$ - $proj$ A ($\sum (x,y) \leftarrow xys. f x y$)
 $= (\sum (x,y) \leftarrow xys. aezfun$ - $setspan$ - $proj$ A ($f x y$))

using $aezfun$ - $setspan$ - $proj$ - sum - $list$ [of A $\lambda xy. case$ - $prod$ f xy]
 $prod$ - $case$ - $distrib$ [of $aezfun$ - $setspan$ - $proj$ A f]

by $simp$

1.6 Polynomials

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

using $coeffs$ - $Poly$ [of as] $strip$ - $while$ - 0 - $nnil$ [of as] **by** $fastforce$

lemma $const$ - $poly$ - $nonzero$ - $coeff$:

```

assumes degree p = 0 p ≠ 0
shows coeff p 0 ≠ 0
proof
  assume z: coeff p 0 = 0
  have  $\bigwedge n. \text{coeff } p \ n = 0$ 
  proof -
    fix n from z assms show coeff p n = 0
    using coeff-eq-0[of p] by (cases n = 0) auto
  qed
  with assms(2) show False using poly-eqI[of p 0] by simp
qed

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 (induct p arbitrary: q)
  case 0
  show ?case
  proof (cases q)
    fix b q' assume q = pCons b q'
    with 00 rpCons show ?thesis by (cases b ≠ 0 ∨ q' ≠ 0) auto
  qed
next
  case (pCons a p)
  show ?case
  proof (cases q)
    fix b q' assume q = pCons b q'
    with pCons lpCons pCons2 show ?thesis by (cases b ≠ 0 ∨ q' ≠ 0) auto
  qed
qed

```

1.7 Algebra of sets

1.7.1 General facts

```

lemma zeroset-eqI:  $0 \in A \implies (\bigwedge a. \ a \in A \implies a = 0) \implies A = 0$ 
  by auto

```

```

lemma sum-list-sets-single :  $(\sum X \leftarrow [A]. \ X) = A$ 
  using add-0-right[of A] by simp

```

```

lemma sum-list-sets-double :  $(\sum X \leftarrow [A,B]. \ X) = A + B$ 
  using add-0-right[of B] by simp

```


1.7.2 Additive independence of sets

primrec *add-independentS* :: 'a::monoid-add set list \Rightarrow 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 $\bigwedge b a. b \in B \Longrightarrow a \in A \Longrightarrow a + b = 0 \Longrightarrow a = 0$

shows *add-independentS* [A,B]

using *assms sum-list-sets-single*[of B] **by** *simp*

lemma *add-independentS-doubleD*:

assumes *add-independentS* [A,B]

shows $\bigwedge b a. b \in B \Longrightarrow a \in A \Longrightarrow a + b = 0 \Longrightarrow a = 0$

using *assms sum-list-sets-single*[of B] **by** *simp*

lemma *add-independentS-double-iff* :

add-independentS [A,B] = $(\forall b \in B. \forall a \in A. a + b = 0 \longrightarrow a = 0)$

using *add-independentS-doubleI add-independentS-doubleD* **by** *fast*

lemma *add-independentS-Cons-conv-sum-right* :

add-independentS (A#As)

= (*add-independentS* [A, $\sum B \leftarrow As. B$] \wedge *add-independentS* As)

using *add-independentS-double-iff*[of A] **by** *auto*

lemma *add-independentS-double-sum-conv-append* :

$\llbracket \forall X \in \text{set } As. 0 \in X; \text{add-independentS } As; \text{add-independentS } Bs;$

add-independentS [$\sum X \leftarrow As. X, \sum X \leftarrow Bs. X$] \rrbracket

\Longrightarrow *add-independentS* (As@Bs)

proof (*induct* As)

case (*Cons* A As)

have *add-independentS* [$\sum X \leftarrow As. X, \sum X \leftarrow Bs. X$]

proof (*rule add-independentS-doubleI*)

fix b a **assume** ba: $b \in (\sum X \leftarrow Bs. X) \ a \in (\sum X \leftarrow As. X) \ a + b = 0$

from *Cons*(2) ba(2) **have** $a \in (\sum X \leftarrow A \# As. X)$

using *set-plus-intro*[of 0 A a] **by** *simp*

with ba(1,3) *Cons*(5) **show** $a = 0$

using *add-independentS-doubleD*[of $\sum X \leftarrow A \# As. X \ \sum X \leftarrow Bs. X \ b \ a$]

by *simp*

qed

moreover **have** $\bigwedge x a. \llbracket x \in (\sum X \leftarrow As @ Bs. X); a \in A; a + x = 0 \rrbracket$

$\Longrightarrow a = 0$

proof–

fix x a **assume** x-a: $x \in (\sum X \leftarrow As @ Bs. X) \ a \in A \ a + x = 0$

from x-a(1) **obtain** xa xb

where xa-xb: $x = xa + xb \ xa \in (\sum X \leftarrow As. X) \ xb \in (\sum X \leftarrow Bs. X)$

using *set-plus-elim*[of x $\sum X \leftarrow As. X$]

by *auto*

from x-a(2) xa-xb(2) **have** $a + xa \in A + (\sum X \leftarrow As. X)$

using *set-plus-intro* **by** *auto*
with *Cons(3,5) xa-xb x-a(2,3)* **show** $a = 0$
using *add-independentS-doubleD*[
of $\sum X \leftarrow A \# As. X \sum X \leftarrow Bs. X$ xb $a+xa$
]
add.assoc[of a] *add-independentS-doubleD*
by *simp*
qed
ultimately show *add-independentS ((A#As)@Bs)* **using** *Cons* **by** *simp*
qed *simp*

lemma *add-independentS-ConsI* :
assumes *add-independentS As*
 $\bigwedge x a. \llbracket x \in (\sum X \leftarrow As. X); a \in A; a+x = 0 \rrbracket \implies a = 0$
shows *add-independentS (A#As)*
using *assms* **by** *simp*

lemma *add-independentS-append-reduce-right* :
add-independentS (As@Bs) \implies add-independentS Bs
by (*induct As*) *auto*

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

proof (*induct As*)
case (*Cons A As*) **show** *add-independentS (A#As)*
proof (*rule add-independentS-ConsI*)
from *Cons* **show** *add-independentS As* **by** *simp*
next
fix $x a$ **assume** $x: x \in (\sum X \leftarrow As. X)$ **and** $a: a \in A$ **and** $sum: a+x = 0$
from x *Cons(3)* **have** $x + 0 \in (\sum X \leftarrow As. X) + (\sum X \leftarrow Bs. X)$ **by** *fast*
with a sum *Cons(2)* **show** $a = 0$ **by** *simp*
qed
qed *simp*

lemma *add-independentS-append-conv-double-sum* :
add-independentS (As@Bs) $\implies add-independentS [\sum X \leftarrow As. X, \sum X \leftarrow Bs. X]$
proof (*induct As*)
case (*Cons A As*)
show *add-independentS [\sum X \leftarrow (A#As). X, \sum X \leftarrow Bs. X]*
proof (*rule add-independentS-doubleI*)
fix $b x$ **assume** $bx: b \in (\sum X \leftarrow Bs. X)$ $x \in (\sum X \leftarrow A \# As. X)$ $x + b = 0$
from $bx(2)$ **obtain** $a as$
where $a-as: a \in A$ $as \in listset As$ $x = a + (\sum z \leftarrow as. z)$
using *set-sum-list-element-Cons*
by *fast*
from $Cons(2)$ **have** *add-independentS [A, \sum X \leftarrow As@Bs. X]*
using *add-independentS-Cons-conv-sum-right*[of A $As@Bs$] **by** *simp*
moreover from $a-as(2)$ $bx(1)$
have $(\sum z \leftarrow as. z) + b \in (\sum X \leftarrow (As@Bs). X)$

```

    using sum-list-listset set-plus-intro
    by auto
  ultimately have a = 0
    using a-as(1,3) bx(3) add-independentS-doubleD[of A - - a] add.assoc[of a]
    by auto
  with a-as(2,3) bx(1,3) Cons show x = 0
    using sum-list-listset
      add-independentS-doubleD[of  $\sum X \leftarrow As. X \sum X \leftarrow Bs. X b \sum z \leftarrow as. z$ ]
    by auto
  qed
qed simp

```

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
((3 $\oplus$   $\leftarrow$ -. -) [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 = ( $\sum N \leftarrow Ns. N$ )  $\Longrightarrow$  add-independentS Ns  $\Longrightarrow$  M = ( $\oplus N \leftarrow Ns. N$ )
unfolding inner-dirsum-def by simp

```

lemma *inner-dirsum-doubleI* :

```

M = A + B  $\Longrightarrow$  add-independentS [A,B]  $\Longrightarrow$  M = A  $\oplus$  B
using inner-dirsumI[of M [A,B]] sum-list-sets-double[of A] by simp

```

lemma *inner-dirsumD* :

```

add-independentS Ms  $\Longrightarrow$  ( $\oplus M \leftarrow Ms. M$ ) = ( $\sum M \leftarrow Ms. M$ )
unfolding inner-dirsum-def by simp

```

lemma *inner-dirsumD2* : \neg add-independentS Ms \Longrightarrow ($\oplus M \leftarrow Ms. M$) = 0

```

unfolding inner-dirsum-def by simp

```

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

```

unfolding inner-dirsum-def by simp

```

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

```

using inner-dirsumD[of [M]] sum-list-sets-single[of M] by simp

```

lemma *inner-dirsum-doubleD* : $add-independentS [M,N] \implies M \oplus N = M + N$
using *inner-dirsumD*[of $[M,N]$] *sum-list-sets-double*[of $M N$] **by** *simp*

lemma *inner-dirsum-Cons* :
 $add-independentS (A \# As) \implies (\bigoplus X \leftarrow (A \# As). X) = A \oplus (\bigoplus X \leftarrow As. X)$
using *inner-dirsumD*[of $A \# As$] *add-independentS-Cons-conv-sum-right*[of A]
inner-dirsum-doubleD[of A] *inner-dirsumD*[of As]
by *simp*

lemma *inner-dirsum-append* :
 $add-independentS (As @ Bs) \implies 0 \in (\sum X \leftarrow Bs. X)$
 $\implies (\bigoplus X \leftarrow (As @ Bs). X) = (\bigoplus X \leftarrow As. X) \oplus (\bigoplus X \leftarrow Bs. X)$
using *inner-dirsumD*[of $As @ Bs$] *add-independentS-append-reduce-left*[of As]
inner-dirsumD[of As] *inner-dirsumD*[of Bs]
add-independentS-append-reduce-right[of $As Bs$]
add-independentS-append-conv-double-sum[of As]
inner-dirsum-doubleD[of $\sum X \leftarrow As. X$]
by *simp*

lemma *inner-dirsum-double-left0*: $0 \oplus A = A$
using *add-independentS-doubleD* *inner-dirsum-doubleI*[of $0+A$] *add-0-left*[of A]
by *simp*

lemma *add-independentS-Cons-conv-dirsum-right* :
 $add-independentS (A \# As) = (add-independentS [A, \bigoplus B \leftarrow As. B]$
 $\wedge add-independentS As)$
using *add-independentS-Cons-conv-sum-right*[of $A As$] *inner-dirsumD* **by** *auto*

lemma *sum-list-listset-dirsum* :
 $add-independentS As \implies as \in listset As \implies sum-list as \in (\bigoplus A \leftarrow As. A)$
using *inner-dirsumD* *sum-list-listset* **by** *fast*

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
by *unfold-locales auto*

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

+ **assumes** *finite*: *finite G*

lemma (in *FinGroup*) *Group : Group G by unfold-locales*

lemma (in *Group*) *FinGroupI : finite G \implies FinGroup G by unfold-locales*

context *Group*
begin

abbreviation *Subgroup* ::
'g set \implies bool where Subgroup H \equiv Group H \wedge H \subseteq G

lemma *SubgroupD1 : Subgroup H \implies Group H by fast*

lemma *zero-closed : 0 \in G*
proof –
from *nonempty obtain g where g \in G by fast*
hence *g - g \in G using diff-closed by fast*
thus *?thesis by simp*
qed

lemma *obtain-nonzero: assumes G \neq 0 obtains g where g \in G and g \neq 0*
using *assms zero-closed by auto*

lemma *zeroS-closed : 0 \subseteq G*
using *zero-closed by simp*

lemma *neg-closed : g \in G \implies -g \in G*
using *zero-closed diff-closed[of 0 g] by simp*

lemma *add-closed : g \in G \implies h \in G \implies g + h \in G*
using *neg-closed[of h] diff-closed[of g -h] by simp*

lemma *neg-add-closed : g \in G \implies h \in G \implies -g + h \in G*
using *neg-closed add-closed by fast*

lemma *sum-list-closed : set (map f as) \subseteq G \implies ($\sum a \leftarrow as. f a$) \in G*
using *zero-closed add-closed by (induct as) auto*

lemma *sum-list-closed-prod :*
set (map (case-prod f) xys) \subseteq G \implies ($\sum (x,y) \leftarrow xys. f x y$) \in G
using *sum-list-closed by fast*

lemma *set-plus-closed : A \subseteq G \implies B \subseteq G \implies A + B \subseteq G*
using *set-plus-def[of A B] add-closed by force*

lemma *zip-add-closed :*
set as \subseteq G \implies set bs \subseteq G \implies set [a + b. (a,b) \leftarrow zip as bs] \subseteq G
using *add-closed by (induct as bs rule: list-induct2') auto*

```

lemma list-diff-closed :
  set gs ⊆ G ⇒ set hs ⊆ G ⇒ set [x-y. (x,y)←zip gs hs] ⊆ G
  using diff-closed by (induct gs hs rule: list-induct2') auto

lemma add-closed-converse-right : g+x ∈ G ⇒ g ∈ G ⇒ x ∈ G
  using neg-add-closed add.assoc[of -g g x] by fastforce

lemma add-closed-inverse-right : x ∉ G ⇒ g ∈ G ⇒ g+x ∉ G
  using add-closed-converse-right by fast

lemma add-closed-converse-left : g+x ∈ G ⇒ x ∈ G ⇒ g ∈ G
  using diff-closed add.assoc[of g] by fastforce

lemma add-closed-inverse-left : g ∉ G ⇒ x ∈ G ⇒ g+x ∉ G
  using add-closed-converse-left by fast

lemma right-translate-bij :
  assumes g ∈ G
  shows bij-betw (λx. x + g) G G
unfolding bij-betw-def proof
  show inj-on (λx. x + g) G by (rule inj-onI) simp
  show (λx. x + g) ' G = G
  proof
    show (λx. x + g) ' G ⊆ G using assms add-closed by fast
    show (λx. x + g) ' G ⊇ G
    proof
      fix x assume x ∈ G
      with assms have x - g ∈ G x = (λx. x + g) (x - g)
      using diff-closed diff-add-cancel[of x] by auto
      thus x ∈ (λx. x + g) ' G by fast
    qed
  qed
qed

lemma right-translate-sum : g ∈ G ⇒ (∑ h ∈ G. f h) = (∑ h ∈ G. f (h + g))
  using right-translate-bij[of g] bij-betw-def[of λh. h + g]
    sum.reindex[of λh. h + g G]
  by simp

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* $\implies f ` A \subseteq G \implies (\sum a \in A. f a) \in G$
proof (*induct set: finite*)
case *empty* **show** ?*case* **using** *zero-closed* **by** *simp*
next
case (*insert a A*) **thus** ?*case* **using** *add-closed* **by** *simp*
qed

lemma *subset-plus-right* : $A \subseteq G + A$
using *zero-closed set-zero-plus2* **by** *fast*

lemma *subset-plus-left* : $A \subseteq A + G$
using *subset-plus-right add.commute* **by** *fast*

end

2.2 Right cosets

context *Group*
begin

definition *rcoset-rel* :: '*g set* \implies ('*g* × '*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* : $g \in G$
shows (*rcoset-rel H*) ' $\{g\} = H + \{g\}$
proof (*rule seteqI*)
fix *x* **assume** $x \in (\text{rcoset-rel } H) \{g\}$
hence $x \in G$ $g - x \in H$ **using** *rcoset-rel-def* **by** *auto*
with *subgrp* **have** $x - g \in H$
using *Group.neg-closed minus-diff-eq*[*of g x*] **by** *fastforce*
from *this* **obtain** *h* **where** $h: h \in H$ $x - g = h$ **by** *fast*
from *h(2)* **have** $x = h + g$ **by** (*simp add: algebra-simps*)
with *h(1)* **show** $x \in H + \{g\}$ **using** *set-plus-def* **by** *fast*
next
fix *x* **assume** $x \in H + \{g\}$
from *this* **obtain** *h* **where** $h: h \in H$ $x = h + g$ **unfolding** *set-plus-def* **by** *fast*
with *subgrp g* **have** $1: x \in G$ **using** *add-closed* **by** *fast*
from *h(2)* **have** $x - g = h$ **by** (*simp add: algebra-simps*)
with *subgrp h(1)* **have** $g - x \in H$ **using** *Group.neg-closed* **by** *fastforce*
with *g 1* **show** $x \in (\text{rcoset-rel } H) \{g\}$ **using** *rcoset-rel-def* **by** *fast*
qed

lemma *rcoset-equiv* :

```

assumes Subgroup  $H$ 
shows equiv  $G$  (rcoset-rel  $H$ )
proof (rule equivI)
show refl-on  $G$  (rcoset-rel  $H$ )
proof (rule refl-onI)
  show (rcoset-rel  $H$ )  $\subseteq G \times G$  using rcoset-rel-def by auto
next
  fix  $x$  assume  $x \in G$ 
  with assms show  $(x,x) \in$  (rcoset-rel  $H$ )
    using rcoset-rel-def Group.zero-closed by auto
qed
show sym (rcoset-rel  $H$ )
proof (rule symI)
  fix  $a$   $b$  assume  $(a,b) \in$  (rcoset-rel  $H$ )
  with assms show  $(b,a) \in$  (rcoset-rel  $H$ )
    using rcoset-rel-def Group.neg-closed[of  $H$   $a - b$ ] minus-diff-eq by simp
qed
show trans (rcoset-rel  $H$ )
proof (rule transI)
  fix  $x$   $y$   $z$  assume  $(x,y) \in$  (rcoset-rel  $H$ )  $(y,z) \in$  (rcoset-rel  $H$ )
  with assms show  $(x,z) \in$  (rcoset-rel  $H$ )
    using rcoset-rel-def Group.add-closed[of  $H$   $x - y$   $y - z$ ]
    by (simp add: algebra-simps)
qed
qed

```

```

lemma rcoset0 : Subgroup  $H \implies$  (rcoset-rel  $H$ )“{0} =  $H$ 
  using zero-closed rcosets unfolding set-plus-def by simp

```

```

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  $gs \subseteq G$ 
  unfolding is-rcoset-replist-def by fast

```

```

lemma is-rcoset-replistD-distinct :
  is-rcoset-replist  $H$   $gs \implies$  distinct (map ( $\lambda g.$  (rcoset-rel  $H$ )“{ $g$ })  $gs$ )
  unfolding is-rcoset-replist-def by fast

```

```

lemma is-rcoset-replistD-cosets :
  is-rcoset-replist  $H$   $gs \implies G = (\bigcup_{g \in \text{set } gs.}$  (rcoset-rel  $H$ )“{ $g$ })
  unfolding is-rcoset-replist-def by fast

```

```

lemma group-eq-subgrp-rcoset-un :
  Subgroup  $H \implies$  is-rcoset-replist  $H$   $gs \implies G = (\bigcup_{g \in \text{set } gs.}$   $H + \{g\}$ )
  using is-rcoset-replistD-set is-rcoset-replistD-cosets rcosets
  by (auto, smt UN-E subsetCE, blast)

```


lemma *is-rcoset-replist-imp-nrelated-nth* :
assumes *Subgroup H is-rcoset-replist H gs*
shows $\bigwedge i j. i < \text{length } gs \implies j < \text{length } gs \implies i \neq j \implies gs!i - gs!j \notin H$
proof
fix *i j* **assume** *ij: i < length gs j < length gs i ≠ j gs!i - gs!j ∈ H*
from *assms(2) ij(1,2,4)* **have** $(gs!i, gs!j) \in \text{rcoset-rel } H$
using *set-conv-nth is-rcoset-replistD-set rcoset-rel-def* **by** *fastforce*
with *assms(1) ij(1,2)*
have $(\text{map } (\lambda g. (\text{rcoset-rel } H) \{\! \{g\}\! \}) gs)!i$
 $= (\text{map } (\lambda g. (\text{rcoset-rel } H) \{\! \{g\}\! \}) gs)!j$
using *rcoset-equiv equiv-class-eq*
by *fastforce*
with *assms(2) ij(1-3)* **show** *False*
using *is-rcoset-replistD-distinct distinct-conv-nth*
 $\text{of map } (\lambda g. (\text{rcoset-rel } H) \{\! \{g\}\! \}) gs$
 $\quad]$
by *auto*
qed

lemma *is-rcoset-replist-Cons* :
 $\text{is-rcoset-replist } H (g \# gs) \longleftrightarrow$
 $g \in G \wedge \text{set } gs \subseteq G$
 $\wedge (\text{rcoset-rel } H) \{\! \{g\}\! \} \notin \text{set } (\text{map } (\lambda x. (\text{rcoset-rel } H) \{\! \{x\}\! \}) gs)$
 $\wedge \text{distinct } (\text{map } (\lambda x. (\text{rcoset-rel } H) \{\! \{x\}\! \}) gs)$
 $\wedge G = (\text{rcoset-rel } H) \{\! \{g\}\! \} \cup (\bigcup x \in \text{set } gs. (\text{rcoset-rel } H) \{\! \{x\}\! \})$
using *is-rcoset-replist-def* $[\text{of } H \text{ } g \# gs]$ **by** *auto*

lemma *rcoset-replist-Hrep* :
assumes *Subgroup H is-rcoset-replist H gs*
shows $\exists g \in \text{set } gs. g \in H$
proof –
from *assms(2)* **obtain** *g* **where** $g: g \in \text{set } gs \ 0 \in (\text{rcoset-rel } H) \{\! \{g\}\! \}$
using *zero-closed is-rcoset-replistD-cosets* **by** *fast*
from *assms(1) g(2)* **have** $g \in (\text{rcoset-rel } H) \{\! \{0\}\! \}$
using *rcoset-equiv equivE sym-def* $[\text{of } \text{rcoset-rel } H]$ **by** *force*
with *assms(1) g* **show** $\exists g \in \text{set } gs. g \in H$ **using** *rcoset0* **by** *fast*
qed

lemma *rcoset-replist-reorder* :
 $\text{is-rcoset-replist } H (gs @ g \# gs') \implies \text{is-rcoset-replist } H (g \# gs @ gs')$
using *is-rcoset-replist-def* **by** *auto*

lemma *rcoset-replist-replacehd* :
assumes *Subgroup H g' ∈ (rcoset-rel H) {g} is-rcoset-replist H (g # gs)*
shows $\text{is-rcoset-replist } H (g' \# gs)$
proof –
from *assms(2)* **have** $g' \in G$ **using** *rcoset-rel-def* **by** *simp*
moreover from *assms(3)* **have** $\text{set } gs \subseteq G$

using *is-rcoset-replistD-set* **by** *fastforce*
moreover from *assms(1-3)*
have $(rcoset-rel\ H)^{\{\!|g|\!\}}$ \notin *set* $(map\ (\lambda x. (rcoset-rel\ H)^{\{x\}})\ gs)$
using *set-conv-nth[of gs] rcoset-equiv equiv-class-eq-iff[of G] is-rcoset-replistD-distinct*
by *fastforce*
moreover from *assms(3)* **have** *distinct* $(map\ (\lambda g. (rcoset-rel\ H)^{\{g\}})\ gs)$
using *is-rcoset-replistD-distinct* **by** *fastforce*
moreover from *assms(1-3)*
have $G = (rcoset-rel\ H)^{\{\!|g|\!\}} \cup (\bigcup_{x \in set\ gs} (rcoset-rel\ H)^{\{x\}})$
using *is-rcoset-replistD-cosets[of H g#gs] rcoset-equiv equiv-class-eq-iff[of G]*
by *simp*
ultimately show *?thesis* **using** *is-rcoset-replist-Cons* **by** *auto*
qed
end

lemma (in *FinGroup*) *ex-rcoset-replist* :
assumes *Subgroup H*
shows $\exists gs. is-rcoset-replist\ H\ gs$
proof –
define *r* **where** $r = rcoset-rel\ H$
hence *equiv-r: equiv G r* **using** *rcoset-equiv[OF assms]* **by** *fast*
have $\forall x \in G//r. \exists g. g \in x$
proof
fix *x* **assume** $x \in G//r$
from *this* **obtain** *g* **where** $g \in G\ x = r^{\{g\}}$
using *quotient-def[of G r]* **by** *auto*
hence $g \in x$ **using** *equiv-r equivE[of G r] refl-onD[of G r]* **by** *auto*
thus $\exists g. g \in x$ **by** *fast*
qed
from *this* **obtain** *f* **where** $f: \forall x \in G//r. f\ x \in x$ **using** *bchoice* **by** *force*
from *r-def* **have** $r \subseteq G \times G$ **using** *rcoset-rel-def* **by** *auto*
with *finite* **have** *finite* $(f^{\{G//r\}})$ **using** *finite-quotient* **by** *auto*
from *this* **obtain** *gs* **where** $gs: distinct\ gs\ set\ gs = f^{\{G//r\}}$
using *finite-distinct-list* **by** *force*

have $1: set\ gs \subseteq G$
proof
fix *g* **assume** $g \in set\ gs$
with *gs(2)* **obtain** *x* **where** $x \in G//r\ g = f\ x$ **by** *fast*
with *f* **show** $g \in G$ **using** *equiv-r Union-quotient* **by** *fast*
qed

moreover have *distinct* $(map\ (\lambda g. r^{\{g\}})\ gs)$
proof –
have $\bigwedge i\ j. \llbracket i < length\ (map\ (\lambda g. r^{\{g\}})\ gs);$
 $j < length\ (map\ (\lambda g. r^{\{g\}})\ gs); i \neq j \rrbracket$
 $\implies (map\ (\lambda g. r^{\{g\}})\ gs)!i \neq (map\ (\lambda g. r^{\{g\}})\ gs)!j$
proof

fix $i\ j$ **assume** ij :
 $i < \text{length} (\text{map} (\lambda g. r^{\{\!|g\}}) gs)$
 $j < \text{length} (\text{map} (\lambda g. r^{\{\!|g\}}) gs)$
 $i \neq j$
 $(\text{map} (\lambda g. r^{\{\!|g\}}) gs)!i = (\text{map} (\lambda g. r^{\{\!|g\}}) gs)!j$
from $ij(1,2)$ **have** $gs!i \in \text{set } gs\ gs!j \in \text{set } gs$ **using** *set-conv-nth* **by** *auto*
from *this* $gs(2)$ **obtain** $x\ y$
where $x: x \in G//r\ gs!i = f\ x$ **and** $y: y \in G//r\ gs!j = f\ y$
by *auto*
have $x = y$
using *equiv-r* $x(1)\ y(1)$
proof (*rule quotient-eqI*[of $G\ r$])
from $ij(1,2,4)$ **have** $r^{\{\!|gs!i\}} = r^{\{\!|gs!j\}}$ **by** *simp*
with $ij(1,2)\ 1$ **show** $(gs!i,gs!j) \in r$
using *eq-equiv-class-iff*[*OF equiv-r*] **by** *force*
from $x\ y\ f$ **show** $gs!i \in x\ gs!j \in y$ **by** *auto*
qed
with $x(2)\ y(2)\ ij(1-3)\ gs(1)$ **show** *False* **using** *distinct-conv-nth* **by** *fastforce*
qed
thus *?thesis* **using** *distinct-conv-nth* **by** *fast*
qed

moreover **have** $G = (\bigcup_{g \in \text{set } gs} r^{\{\!|g\}})$
proof (*rule subset-antisym*, *rule subsetI*)
fix g **assume** $g \in G$
hence $rg: r^{\{\!|g\}} \in G//r$ **using** *quotientI* **by** *fast*
with $gs(2)$ **obtain** g' **where** $g': g' \in \text{set } gs\ g' = f (r^{\{\!|g\}})$ **by** *fast*
from $f\ g'(2)\ rg$ **have** $g \in r^{\{\!|g'\}}$ **using** *equiv-r equivE sym-def*[of r] **by** *force*
with $g'(1)$ **show** $g \in (\bigcup_{g \in \text{set } gs} r^{\{\!|g\}})$ **by** *fast*
next
from *r-def* **show** $G \supseteq (\bigcup_{g \in \text{set } gs} r^{\{\!|g\}})$ **using** *rcoset-rel-def* **by** *auto*
qed

ultimately **show** *?thesis* **using** *r-def unfolding is-rcoset-replist-def* **by** *fastforce*
qed

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

assumes *Subgroup H*

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

proof –

from *assms* **obtain** xs **where** $xs: \text{is-rcoset-replist } H\ xs$

using *ex-rcoset-replist* **by** *fast*

with *assms* **obtain** x **where** $x: x \in \text{set } xs\ x \in H$

using *rcoset-replist-Hrep* **by** *fast*

from $x(2)$ **have** $(0, x) \in \text{rcoset-rel } H$ **using** *rcoset0*[*OF assms*] **by** *auto*

moreover **have** *sym* (*rcoset-rel H*)

using *rcoset-equiv*[*OF assms*] *equivE*[of $G\ \text{rcoset-rel } H$] **by** *simp*

ultimately **have** $0 \in (\text{rcoset-rel } H)^{\{\!|x\}}$ **using** *sym-def* **by** *fast*

with $x(1)\ xs\ \text{assms}$ **show** $\exists gs. \text{is-rcoset-replist } H\ (0 \# gs)$

using *split-list rcoset-replist-reorder rcoset-replist-replacehd* by *fast*
 qed

2.3 Group homomorphisms

2.3.1 Preliminaries

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

lemma $\ker I : f a = 0 \implies a \in \ker f$
 unfolding *ker-def* by *fast*

lemma $\ker D : a \in \ker f \implies f a = 0$
 unfolding *ker-def* by *fast*

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

proof

assume $A: A \neq \{\} \wedge A \subseteq \ker f$

show $f ' A = 0$

proof (*rule zeroset-eqI*)

from A obtain a where $a: a \in A$ by *fast*

with A have $f a = 0$ using *kerD* by *fastforce*

from *this*[*THEN sym*] a show $0 \in f ' A$ by *fast*

next

fix b assume $b \in f ' A$

from *this* obtain a where $a \in A$ $b = f a$ by *fast*

with A show $b = 0$ using *kerD* by *fast*

qed

next

assume $fA: f ' A = 0$

have $A \neq \{\}$

proof–

from fA obtain a where $a \in A$ $f a = 0$ by *force*

thus *?thesis* by *fast*

qed

moreover have $A \subseteq \ker f$

proof

fix a assume $a \in A$

with fA have $f a = 0$ by *auto*

thus $a \in \ker f$ using *kerI* by *fast*

qed

ultimately show $A \neq \{\} \wedge A \subseteq \ker f$ by *fast*

qed

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::group-add set
+ fixes T :: 'g ⇒ 'h::group-add
  assumes hom :  $\bigwedge g g'. g \in G \implies g' \in G \implies T (g + g') = T g + T g'$ 
  and supp: supp T  $\subseteq$  G

```

```

abbreviation (in GroupHom) Ker  $\equiv$  ker T  $\cap$  G
abbreviation (in GroupHom) ImG  $\equiv$  T ' G

```

```

locale GroupEnd = GroupHom G T
  for G :: 'g::group-add set
  and T :: 'g ⇒ 'g
+ assumes endomorph: ImG  $\subseteq$  G

```

```

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

```

2.3.3 Basic facts

```

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

```

```

proof

```

```

  fix g g'
  define z z-map where z = (0::'h) and z-map = (0::'g⇒'h)
  thus z-map (g + g') = z-map g + z-map g' by simp
qed (rule supp-zerofun-subset-any)

```

```

lemma (in Group) GroupHom-idhom : GroupHom G (id $\downarrow$ G)
  using add-closed supp-restrict0 by unfold-locales simp

```

```

context GroupHom
begin

```

```

lemma im-zero : T 0 = 0
  using zero-closed hom[of 0 0] add-diff-cancel[of T 0 T 0] by simp

```

```

lemma zero-in-Ker : 0  $\in$  Ker
  using im-zero kerI zero-closed by fast

```

```

lemma comp-zero : T  $\circ$  0 = 0
  using im-zero by auto

```

```

lemma im-neg : T (- g) = - T g
  using im-zero hom[of g - g] neg-closed[of g] minus-unique[of T g]
    neg-closed[of -g] supp suppI-contr[of g T] suppI-contr[of -g T]
  by fastforce

```

lemma *im-diff* : $g \in G \implies g' \in G \implies T (g - g') = T g - T g'$
using *hom neg-closed hom*[of $g - g'$] *im-neg* **by** *simp*

lemma *eq-im-imp-diff-in-Ker* : $\llbracket g \in G; h \in G; T g = T h \rrbracket \implies g - h \in \text{Ker}$
using *im-diff kerI diff-closed*[of $g h$] **by** *force*

lemma *im-sum-list-prod* :

set (*map* (*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 (*induct xys*)

case *Nil*

show *?case* **using** *im-zero* **by** *simp*

next

case (*Cons xy xys*)

define *Tf* **where** $Tf = T \circ (\text{case-prod } f)$

have $T (\sum_{(x,y) \leftarrow (xy \# xys)}. f x y) = T ((\text{case-prod } f) xy + (\sum_{(x,y) \leftarrow xys}. f x y))$

by *simp*

moreover from *Cons(2)* **have** $(\text{case-prod } f) xy \in G$ *set* (*map* (*case-prod* *f*) *xys*) $\subseteq G$

by *auto*

ultimately have $T (\sum_{(x,y) \leftarrow (xy \# xys)}. f x y) = Tf xy + (\sum_{(x,y) \leftarrow xys}. Tf (x,y))$

using *Tf-def sum-list-closed*[of *case-prod* *f*] *hom Cons* **by** *auto*

also have $\dots = (\sum_{(x,y) \leftarrow (xy \# xys)}. Tf (x,y))$ **by** *simp*

finally show *?case* **using** *Tf-def* **by** *simp*

qed

lemma *distrib-comp-sum-left* :

range *S* $\subseteq G \implies \text{range } S' \subseteq G \implies T \circ (S + S') = (T \circ S) + (T \circ S')$

using *hom* **by** (*auto simp add: fun-eq-iff*)

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

using *nonempty ker-im-iff*[of *G T*] **by** *fast*

lemma *Ker0-imp-inj-on* :

assumes $\text{Ker} \subseteq 0$

shows *inj-on* *T G*

proof (*rule inj-onI*)

fix *x y* **assume** *xy*: $x \in G y \in G T x = T y$

hence $T (x - y) = 0$ **using** *im-diff* **by** *simp*

with *xy(1,2)* **have** $x - y \in \text{Ker}$ **using** *diff-closed kerI* **by** *fast*

with *assms* **show** $x = y$ **using** *zero-in-Ker* **by** *auto*

qed

lemma *inj-on-imp-Ker0* :

assumes *inj-on* *T G*

shows $\text{Ker} = 0$

using *zero-in-Ker kerD zero-closed im-zero inj-onD*[*OF assms*]

by *fastforce*

lemma *nonzero-Ker-el-imp-n-inj* :
 $g \in G \implies g \neq 0 \implies T g = 0 \implies \neg \text{inj-on } T G$
using *inj-on-imp-Ker0 kerI[of T]* **by** *auto*

lemma *Group-Ker* : *Group Ker*
proof
show $Ker \neq \{\}$ **using** *zero-in-Ker* **by** *fast*
next
fix $g h$ **assume** $g \in Ker h \in Ker$ **thus** $g - h \in Ker$
using *im-diff kerD[of g T] kerD[of h T] diff-closed kerI[of T]* **by** *auto*
qed

lemma *Group-Im* : *Group ImG*
proof
show $ImG \neq \{\}$ **using** *nonempty* **by** *fast*
next
fix $g' h'$ **assume** $g' \in ImG h' \in ImG$
from *this* **obtain** $g h$ **where** $gh: g \in G g' = T g h \in G h' = T h$ **by** *fast*
thus $g' - h' \in ImG$ **using** *im-diff diff-closed* **by** *force*
qed

lemma *GroupHom-restrict0-subgroup* :
assumes *Subgroup H*
shows *GroupHom H (T ↓ H)*
proof (*intro-locales, rule SubgroupD1[OF assms], unfold-locales*)
show $\text{supp } (T \downarrow H) \subseteq H$ **using** *supp-restrict0* **by** *fast*
next
fix $h h'$ **assume** $hh': h \in H h' \in H$
with *assms* **show** $(T \downarrow H) (h + h') = (T \downarrow H) h + (T \downarrow H) h'$
using *Group.add-closed hom[of h h']* **by** *auto*
qed

lemma *im-subgroup* :
assumes *Subgroup H*
shows *Group.Subgroup ImG (T ‘ H)*
proof
from *assms* **have** *Group ((T ↓ H) ‘ H)*
using *GroupHom-restrict0-subgroup GroupHom.Group-Im* **by** *fast*
moreover **have** $(T \downarrow H) ‘ H = T ‘ H$ **by** *auto*
ultimately **show** *Group (T ‘ H)* **by** *simp*
from *assms* **show** $T ‘ H \subseteq ImG$ **by** *fast*
qed

lemma *GroupHom-composite-left* :
assumes $ImG \subseteq H$ *GroupHom H S*
shows *GroupHom G (S ◦ T)*
proof

```

fix  $g\ g'$  assume  $g \in G\ g' \in G$ 
with  $hom\ assms(1)$  show  $(S \circ T)\ (g + g') = (S \circ T)\ g + (S \circ T)\ g'$ 
  using  $GroupHom.hom[OF\ assms(2)]$  by fastforce
next
from  $supp$  have  $\bigwedge g. g \notin G \implies (S \circ T)\ g = 0$ 
  using  $suppI\ contra\ GroupHom.im\ zero[OF\ assms(2)]$  by fastforce
thus  $supp\ (S \circ T) \subseteq G$  using  $suppD\ contra$  by fast
qed

lemma  $idhom\ left : T \upharpoonright G \subseteq H \implies (id \downarrow H) \circ T = T$ 
  using  $supp\ suppI\ contra$  by fastforce

end

```

2.3.4 Basic facts about endomorphisms

```

context  $GroupEnd$ 
begin

```

```

lemmas  $hom = hom$ 

```

```

lemma  $range : range\ T \subseteq G$ 

```

```

proof ( $rule\ image\ subsetI$ )

```

```

  fix  $x$  show  $T\ x \in G$ 

```

```

  proof ( $cases\ x \in G$ )

```

```

    case  $True$  with  $endomorph$  show  $?thesis$  by fast

```

```

  next

```

```

    case  $False$  with  $supp$  show  $?thesis$  using  $suppI\ contra\ zero\ closed$  by fastforce

```

```

  qed

```

```

qed

```

```

lemma  $proj\ decomp :$ 

```

```

  assumes  $\bigwedge g. g \in G \implies T\ (T\ g) = T\ g$ 

```

```

  shows  $G = Ker \oplus ImG$ 

```

```

proof ( $rule\ inner\ dirsum\ doubleI, rule\ subset\ antisym, rule\ subsetI$ )

```

```

  fix  $g$  assume  $g : g \in G$ 

```

```

  have  $g = (g - T\ g) + T\ g$  using  $diff\ add\ cancel[of\ g]$  by simp

```

```

  moreover have  $g - T\ g \in Ker$ 

```

```

  proof

```

```

    from  $g\ endomorph\ assms$  have  $T\ (g - T\ g) = 0$  using  $im\ diff$  by auto

```

```

    thus  $g - T\ g \in ker\ T$  using  $kerI$  by fast

```

```

    from  $g\ endomorph$  show  $g - T\ g \in G$  using  $diff\ closed$  by fast

```

```

  qed

```

```

  moreover from  $g$  have  $T\ g \in ImG$  by fast

```

```

  ultimately show  $g \in Ker + ImG$ 

```

```

    using  $set\ plus\ intro[of\ g - T\ g\ Ker\ T\ g]$  by simp

```

```

next

```

```

  from  $endomorph$  show  $G \supseteq Ker + ImG$  using  $set\ plus\ closed$  by simp

```

```

  show  $add\ independentS\ [Ker, ImG]$ 

```


proof (*rule add-independentS-doubleI*)
fix $g\ h$ **assume** $gh: h \in \text{Im}G\ g \in \text{Ker}\ g + h = 0$
from $gh(1)$ **obtain** g' **where** $g' \in G\ h = T\ g'$ **by** *fast*
with $gh(2,3)$ *endomorph assms* **have** $h = 0$
using *im-zero hom[of g T g'] kerD* **by** *fastforce*
with $gh(3)$ **show** $g = 0$ **by** *simp*
qed
qed
end

2.3.5 Basic facts about isomorphisms

context *GroupIso*
begin

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

lemma $\text{Im}G : \text{Im}G = H$ **using** *bijjective bij-betw-imp-surj-on* **by** *fast*

lemma $\text{Group}H : \text{Group } H$ **using** *ImG Group-Im* **by** *fast*

lemma $\text{inv}T\text{-onto} : \text{inv}T \text{ ' } H = G$
using *bijjective bij-betw-imp-inj-on[of T] ImG the-inv-into-onto[of T]* **by** *force*

lemma $\text{inj-on-inv}T : \text{inj-on } \text{inv}T\ H$
using *bijjective bij-betw-imp-inj-on[of T G] ImG inj-on-the-inv-into[of T]*
unfolding *inj-on-def*
by *force*

lemma $\text{bijjective-inv}T : \text{bij-betw } \text{inv}T\ H\ G$
using *inj-on-invT invT-onto unfolding bij-betw-def* **by** *fast*

lemma $\text{inv}T\text{-into} : h \in H \implies \text{inv}T\ h \in G$
using *bijjective bij-betw-imp-inj-on ImG the-inv-into-into[of T]* **by** *force*

lemma $T\text{-inv}T : h \in H \implies T\ (\text{inv}T\ h) = h$
using *bijjective bij-betw-imp-inj-on ImG f-the-inv-into-f[of T]* **by** *force*

lemma $\text{inv}T\text{-eq} : g \in G \implies T\ g = h \implies \text{inv}T\ h = g$
using *bijjective bij-betw-imp-inj-on ImG the-inv-into-f-eq[of T]* **by** *force*

lemma $\text{inv} : \text{GroupIso } H\ \text{inv}T\ G$

proof (*intro-locales, rule GroupH, unfold-locales*)

show $\text{supp } \text{inv}T \subseteq H$ **using** *supp-restrict0* **by** *fast*

show $\text{bij-betw } \text{inv}T\ H\ G$ **using** *bijjective-invT* **by** *fast*

next

fix $h\ h'$ **assume** $h \in H\ h' \in H$

thus $\text{inv}T\ (h + h') = \text{inv}T\ h + \text{inv}T\ h'$

using *invT-into hom T-invT add-closed invT-eq* by *simp*
 qed
 end

2.3.6 Hom-sets

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

lemma *GroupHomSetI* :
GroupHom *G T* \Longrightarrow *T* ' *G* \subseteq *H* \Longrightarrow *T* \in *GroupHomSet* *G H*
 unfolding *GroupHomSet-def* by *fast*

lemma *GroupHomSetD-GroupHom* :
T \in *GroupHomSet* *G H* \Longrightarrow *GroupHom* *G T*
 unfolding *GroupHomSet-def* by *fast*

lemma *GroupHomSetD-Im* : *T* \in *GroupHomSet* *G H* \Longrightarrow *T* ' *G* \subseteq *H*
 unfolding *GroupHomSet-def* by *fast*

lemma (in *Group*) *Group-GroupHomSet* :
 fixes *H* :: '*h*::ab-group-add set
 assumes *AbGroup* *H*
 shows *Group* (*GroupHomSet* *G H*)

proof
 show *GroupHomSet* *G H* \neq {}
 using *trivial-GroupHom AbGroup.zero-closed[OF assms]* *GroupHomSetI*
 by *fastforce*

next
 fix *S T* assume *ST*: *S* \in *GroupHomSet* *G H* *T* \in *GroupHomSet* *G H*
 show *S* - *T* \in *GroupHomSet* *G H*

proof (rule *GroupHomSetI*, *unfold-locales*)
 from *ST* show *supp* (*S* - *T*) \subseteq *G*
 using *GroupHomSetD-GroupHom[of S]* *GroupHomSetD-GroupHom[of T]*
 GroupHom.supp[of G S] *GroupHom.supp[of G T]*
 supp-diff-subset-union-supp[of S T]
 by *auto*

show (*S* - *T*) ' *G* \subseteq *H*

proof (rule *image-subsetI*)
 fix *g* assume *g* \in *G*
 with *ST* have *S g* \in *H* *T g* \in *H*
 using *GroupHomSetD-Im[of S G]* *GroupHomSetD-Im[of T G]* by *auto*
 thus (*S* - *T*) *g* \in *H* using *AbGroup.diff-closed[OF assms]* by *simp*

qed

next
 fix *g g'* assume *g* \in *G* *g'* \in *G*
 with *ST* show (*S* - *T*) (*g* + *g'*) = (*S* - *T*) *g* + (*S* - *T*) *g'*
 using *GroupHomSetD-GroupHom[of S]* *GroupHom.hom[of G S]*

by $\text{GroupHomSetD-GroupHom[of } T \text{]} \text{ GroupHom.hom[of } G \text{ } T \text{]}$
 simp
 qed
 qed

2.4 Facts about collections of groups

lemma *listset-Group-plus-closed* :

$\llbracket \forall G \in \text{set } Gs. \text{ Group } G; as \in \text{listset } Gs; bs \in \text{listset } Gs \rrbracket$
 $\implies [a+b. (a,b) \leftarrow \text{zip } as \text{ } bs] \in \text{listset } Gs$

proof–

have $\llbracket \text{length } as = \text{length } bs; \text{length } bs = \text{length } Gs;$
 $as \in \text{listset } Gs; bs \in \text{listset } Gs; \forall G \in \text{set } Gs. \text{ Group } G \rrbracket$
 $\implies [a+b. (a,b) \leftarrow \text{zip } as \text{ } bs] \in \text{listset } Gs$

proof (*induct as bs Gs rule: list-induct3*)

case (*Cons a as b bs G Gs*)

thus $[x+y. (x,y) \leftarrow \text{zip } (a\#as) (b\#bs)] \in \text{listset } (G\#Gs)$

using *listset-ConsD[of a] listset-ConsD[of b] Group.add-closed*
listset-ConsI[of a+b G]

by *fastforce*

qed *simp*

thus $\llbracket \forall G \in \text{set } Gs. \text{ Group } G; as \in \text{listset } Gs; bs \in \text{listset } Gs \rrbracket$
 $\implies [a+b. (a,b) \leftarrow \text{zip } as \text{ } bs] \in \text{listset } Gs$

using *listset-length[of as Gs] listset-length[of bs Gs, THEN sym]* **by** *fastforce*

qed

lemma *AbGroup-set-plus* :

assumes *AbGroup H AbGroup G*

shows *AbGroup (H + G)*

proof

from *assms show H + G \neq {}* **using** *AbGroup.nonempty* **by** *blast*

next

fix *x y* **assume** $x \in H + G \ y \in H + G$

from *this* **obtain** *xh xg yh yg*

where *xy: xh \in H xg \in G x = xh + xg yh \in H yg \in G y = yh + yg*

unfolding *set-plus-def* **by** *fast*

hence $x - y = (xh - yh) + (xg - yg)$ **by** *simp*

thus $x - y \in H + G$ **using** *assms xy(1,2,4,5) AbGroup.diff-closed* **by** *auto*

qed

lemma *AbGroup-sum-list* :

$(\forall G \in \text{set } Gs. \text{ AbGroup } G) \implies \text{AbGroup } (\sum G \leftarrow Gs. G)$

using *trivial-Group AbGroup.intro AbGroup-set-plus*

by (*induct Gs*) *auto*

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 (*induct Gs arbitrary: H*)

case (*Cons G Gs*)

```

show  $H \subseteq (\sum X \leftarrow (G \# Gs). X)$ 
proof (cases  $H = G$ )
  case True with Cons(2) show ?thesis
    using AbGroup-sum-list AbGroup.subset-plus-left by auto
  next
  case False
    with Cons have  $H \subseteq (\sum G \leftarrow Gs. G)$  by simp
    with Cons(2) show ?thesis using AbGroup.subset-plus-right by auto
qed
qed simp

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 (induct Gs arbitrary: G G')
  case (Cons H Hs)
  from Cons(1-3) have  $\bigwedge A B. \llbracket A \in \text{set } Hs; B \in \text{set } Hs; A \neq B \rrbracket$ 
     $\implies A \cap B \subseteq 0$ 
    by simp
  moreover have  $\bigwedge A. A \in \text{set } Hs \implies A \neq H \implies A \cap H \subseteq 0$ 
  proof
    fix A g assume A:  $A \in \text{set } Hs$   $A \neq H$  and g:  $g \in A \cap H$ 
    from A(1) g Cons(2) have  $-g \in (\sum X \leftarrow Hs. X)$ 
      using AbGroup.neg-closed AbGroup-subset-sum-list by force
    moreover have  $g + (-g) = 0$  by simp
    ultimately show  $g \in 0$  using g Cons(3) by simp
  qed
  ultimately have  $\bigwedge A B. \llbracket A \in \text{set } (H \# Hs); B \in \text{set } (H \# Hs); A \neq B \rrbracket$ 
     $\implies A \cap B \subseteq 0$ 
    by auto
  with Cons(4-6) have  $G \cap G' \subseteq 0$  by fast
  moreover from Cons(2,4,5) have  $0 \subseteq G \cap G'$ 
    using AbGroup.zero-closed by auto
  ultimately show ?case by fast
qed simp

lemma independent-AbGroups-pairwise-int0-double :
  assumes AbGroup G AbGroup G' add-independentS [G,G']
  shows  $G \cap G' = 0$ 
proof (cases  $G = 0$ )
  case True with assms(2) show ?thesis using AbGroup.zero-closed by auto
  next
  case False show ?thesis
    proof (rule independent-AbGroups-pairwise-int0)
      from assms(1,2) show  $\forall G \in \text{set } [G, G'] . \text{AbGroup } G$  by simp
      from assms(3) show add-independentS [G,G'] by fast
      show  $G \in \text{set } [G, G']$   $G' \in \text{set } [G, G']$  by auto
      show  $G \neq G'$ 
    proof

```

```

assume  $GG': G = G'$ 
from False assms(1) obtain  $g$  where  $g: g \in G \ g \neq 0$ 
  using AbGroup.nonempty by auto
moreover from assms(2)  $GG' \ g(1)$  have  $-g \in G'$ 
  using AbGroup.neg-closed by fast
moreover have  $g + (-g) = 0$  by simp
ultimately show False using assms(3) by force
qed
qed
qed

```

2.5 Inner direct sums of Abelian groups

2.5.1 General facts

```

lemma AbGroup-inner-dirsum :
   $\forall G \in \text{set } Gs. \text{AbGroup } G \implies \text{AbGroup } (\bigoplus G \leftarrow Gs. G)$ 
using inner-dirsumD[of Gs] inner-dirsumD2[of Gs] AbGroup-sum-list AbGroup.intro
  trivial-Group
by (cases add-independentS Gs) auto

```

```

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

```

```

  assumes Group  $A \ B \neq \{\}$   $A = A \oplus B$ 
  shows  $B = 0$ 

```

```

proof (cases add-independentS [A,B])

```

```

  case True

```

```

    with assms(3) have  $1: A = A + B$  using inner-dirsum-doubleD by fast

```

```

    have  $\bigwedge b. b \in B \implies b = 0$ 

```

```

    proof-

```

```

      fix  $b$  assume  $b: b \in B$ 

```

```

      from assms(1) obtain  $a$  where  $a: a \in A$  using Group.nonempty by fast

```

```

      with  $b \ 1$  have  $a + b \in A$  by fast

```

```

      from this obtain  $a'$  where  $a': a' \in A \ a + b = a'$  by fast

```

```

      hence  $(-a' + a) + b = 0$  by (simp add: add.assoc)

```

```

      with assms(1) True  $a \ a'(1) \ b$  show  $b = 0$ 

```

```

        using Group.neg-add-closed[of A] add-independentS-doubleD[of A B b -a'+a]

```

```

        by simp

```

```

    qed

```

```

    with assms(2) show ?thesis by auto

```

```

next

```

```

  case False

```

```

    hence  $1: A \oplus B = 0$  unfolding inner-dirsum-def by auto

```

```

    moreover with assms(3) have  $A = 0$  by fast

```

```

    ultimately show ?thesis using inner-dirsum-double-left0 by auto

```

```

qed

```

```

lemma AbGroup-subset-inner-dirsum :

```

```

  [  $\forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; H \in \text{set } Gs ]$ 
   $\implies H \subseteq (\bigoplus G \leftarrow Gs. G)$ 

```

```

using AbGroup-subset-sum-list inner-dirsumD by fast

```

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

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

$$\implies Gs!n \subseteq (\bigoplus G \leftarrow Gs. G)$$
using *AbGroup-subset-inner-dirsum* **by** *force*

lemma *AbGroup-inner-dirsum-el-decomp-ex1-double* :
assumes *AbGroup* *G* *AbGroup* *H* *add-independentS* [*G,H*] *x* \in $G \oplus H$
shows $\exists !gh. \text{fst } gh \in G \wedge \text{snd } gh \in H \wedge x = \text{fst } gh + \text{snd } gh$
proof (*rule ex-ex1I*)
from *assms*(3,4) **obtain** *g h* **where** $x = g + h$ $g \in G$ $h \in H$
using *inner-dirsum-doubleD* *set-plus-elim* **by** *blast*
from *this* **have** $1: \text{fst } (g,h) \in G$ $\text{snd } (g,h) \in H$ $x = \text{fst } (g,h) + \text{snd } (g,h)$
by *auto*
thus $\exists gh. \text{fst } gh \in G \wedge \text{snd } gh \in H \wedge x = \text{fst } gh + \text{snd } gh$ **by** *fast*

next
fix *gh gh'* **assume** *A*:
 $\text{fst } gh \in G \wedge \text{snd } gh \in H \wedge x = \text{fst } gh + \text{snd } gh$
 $\text{fst } gh' \in G \wedge \text{snd } gh' \in H \wedge x = \text{fst } gh' + \text{snd } gh'$
show $gh = gh'$
proof
from *A* *assms*(1,2) **have** $\text{fst } gh - \text{fst } gh' \in G$ $\text{snd } gh - \text{snd } gh' \in H$
using *AbGroup.diff-closed* **by** *auto*
moreover from *A* **have** $z: (\text{fst } gh - \text{fst } gh') + (\text{snd } gh - \text{snd } gh') = 0$
by (*simp add: algebra-simps*)
ultimately show $\text{fst } gh = \text{fst } gh'$
using *assms*(3)
 $\text{add-independentS-doubleD}[of\ G\ H\ \text{snd } gh - \text{snd } gh'\ \text{fst } gh - \text{fst } gh']$
by *simp*
with *z* **show** $\text{snd } gh = \text{snd } gh'$ **by** *simp*

qed
qed

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

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

$$\implies \forall x \in (\bigoplus G \leftarrow Gs. G). \exists !gs \in \text{listset } Gs. x = \text{sum-list } gs$$
proof (*induct Gs*)
case *Nil*
have $\bigwedge x::'a. x \in (\bigoplus H \leftarrow []. H) \implies \exists !gs \in \text{listset } []. x = \text{sum-list } gs$
proof
fix $x::'a$ **assume** $x \in (\bigoplus G \leftarrow []. G)$
moreover define $f :: 'a \Rightarrow 'a \text{ list}$ **where** $f\ x = []$ **for** x
ultimately show $f\ x \in \text{listset } [] \wedge x = \text{sum-list } (f\ x)$
using *inner-dirsum-Nil* **by** *auto*

next
fix $x::'a$ **and** gs
assume $x: x \in (\bigoplus G \leftarrow []. G)$
and $gs: gs \in \text{listset } [] \wedge x = \text{sum-list } gs$
thus $gs = []$ **by** *simp*

qed
thus $\forall x::'a \in (\bigoplus H \leftarrow []). H). \exists !gs \in \text{listset } []. x = \text{sum-list } gs$ **by fast**
next
case $(\text{Cons } G \ Gs)$
hence $\text{prevcase}: \forall x \in (\bigoplus H \leftarrow Gs. H). \exists !gs \in \text{listset } Gs. x = \text{sum-list } gs$ **by auto**
from $\text{Cons}(2)$ **have** $grps: \text{AbGroup } G \ \text{AbGroup } (\bigoplus H \leftarrow Gs. H)$
using $\text{AbGroup-inner-dirsum}$ **by auto**
from $\text{Cons}(3)$ **have** $\text{ind}: \text{add-independentS } [G, \bigoplus H \leftarrow Gs. H]$
using $\text{add-independentS-Cons-conv-dirsum-right}$ **by fast**
have $\bigwedge x. x \in (\bigoplus H \leftarrow (G \# Gs). H) \implies \exists !gs \in \text{listset } (G \# Gs). x = \text{sum-list } gs$
proof (rule ex-ex1I)
fix x **assume** $x \in (\bigoplus H \leftarrow (G \# Gs). H)$
with $\text{Cons}(3)$ **have** $x \in G \oplus (\bigoplus H \leftarrow Gs. H)$
using inner-dirsum-Cons **by fast**
with $grps$ ind **obtain** gh
where $gh: \text{fst } gh \in G \ \text{snd } gh \in (\bigoplus H \leftarrow Gs. H) \ x = \text{fst } gh + \text{snd } gh$
using $\text{AbGroup-inner-dirsum-el-decomp-ex1-double}$
by blast
from $gh(2)$ prevcase **obtain** gs **where** $gs: gs \in \text{listset } Gs \ \text{snd } gh = \text{sum-list } gs$
by fast
with $gh(1)$ $gs(1)$ **have** $\text{fst } gh \# gs \in \text{listset } (G \# Gs)$
using set-Cons-def **by fastforce**
moreover from $gh(3)$ $gs(2)$ **have** $x = \text{sum-list } (\text{fst } gh \# gs)$ **by simp**
ultimately show $\exists gs. gs \in \text{listset } (G \# Gs) \wedge x = \text{sum-list } gs$ **by fast**
next
fix $x \ gs \ hs$
assume $x \in (\bigoplus H \leftarrow (G \# Gs). H)$
and $gs: gs \in \text{listset } (G \# Gs) \wedge x = \text{sum-list } gs$
and $hs: hs \in \text{listset } (G \# Gs) \wedge x = \text{sum-list } hs$
hence $gs \in \text{set-Cons } G \ (\text{listset } Gs) \ hs \in \text{set-Cons } G \ (\text{listset } Gs)$ **by auto**
from this **obtain** $a \ as \ b \ bs$
where $a-as: gs = a \# as \ a \in G \ as \in \text{listset } Gs$
and $b-bs: hs = b \# bs \ b \in G \ bs \in \text{listset } Gs$
unfolding set-Cons-def
by fast
from $a-as(3)$ $b-bs(3)$ $\text{Cons}(3)$
have $as: \text{sum-list } as \in (\bigoplus H \leftarrow Gs. H)$ **and** $bs: \text{sum-list } bs \in (\bigoplus H \leftarrow Gs. H)$
using $\text{sum-list-listset-dirsum}$
by auto
with $a-as(2)$ $b-bs(2)$ $grps$
have $a - b \in G \ \text{sum-list } as - \text{sum-list } bs \in (\bigoplus H \leftarrow Gs. H)$
using $\text{AbGroup.diff-closed}$
by auto
moreover from $gs \ hs \ a-as(1)$ $b-bs(1)$
have $z: (a - b) + (\text{sum-list } as - \text{sum-list } bs) = 0$
by $(\text{simp add: algebra-simps})$
ultimately have $a - b = 0$ **using** $\text{ind add-independentS-doubleD}$ **by blast**
with z **have** $1: a = b$ **and** $z': \text{sum-list } as = \text{sum-list } bs$ **by auto**
from z' prevcase $as \ a-as(3)$ $bs \ b-bs(3)$ **have** $2: as = bs$ **by fast**

from 1 2 a-as(1) b-bs(1) **show** $gs = hs$ **by fast**
qed
thus $\forall x \in (\bigoplus H \leftarrow (G \# Gs). H). \exists !gs. gs \in \text{listset } (G \# Gs) \wedge x = \text{sum-list } gs$
by fast
qed

lemma *AbGroup-inner-dirsum-pairwise-int0* :
 $\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; G \in \text{set } Gs; G' \in \text{set } Gs;$
 $G \neq G' \rrbracket \implies G \cap G' = 0$
proof (*induct Gs arbitrary: G G'*)
case (*Cons H Hs*)
from *Cons(1-3)* **have** $\bigwedge A B. \llbracket A \in \text{set } Hs; B \in \text{set } Hs; A \neq B \rrbracket$
 $\implies A \cap B \subseteq 0$
by simp
moreover **have** $\bigwedge A. A \in \text{set } Hs \implies A \neq H \implies A \cap H \subseteq 0$
proof
fix $A g$ **assume** $A: A \in \text{set } Hs \ A \neq H$ **and** $g: g \in A \cap H$
from *A(1) g Cons(2)* **have** $-g \in (\sum X \leftarrow Hs. X)$
using *AbGroup.neg-closed AbGroup-subset-sum-list* **by force**
moreover **have** $g + (-g) = 0$ **by simp**
ultimately **show** $g \in 0$ **using** g *Cons(3)* **by simp**
qed
ultimately **have** $\bigwedge A B. \llbracket A \in \text{set } (H \# Hs); B \in \text{set } (H \# Hs); A \neq B \rrbracket$
 $\implies A \cap B \subseteq 0$
by auto
with *Cons(4-6)* **have** $G \cap G' \subseteq 0$ **by fast**
moreover **from** *Cons(2,4,5)* **have** $0 \subseteq G \cap G'$
using *AbGroup.zero-closed* **by auto**
ultimately **show** *?case* **by fast**
qed simp

lemma *AbGroup-inner-dirsum-pairwise-int0-double* :
assumes *AbGroup G AbGroup G' add-independentS [G,G']*
shows $G \cap G' = 0$
proof (*cases G = 0*)
case *True* **with** *assms(2)* **show** *?thesis* **using** *AbGroup.zero-closed* **by auto**
next
case *False* **show** *?thesis*
proof (*rule AbGroup-inner-dirsum-pairwise-int0*)
from *assms(1,2)* **show** $\forall G \in \text{set } [G, G']. \text{AbGroup } G$ **by simp**
from *assms(3)* **show** *add-independentS [G,G']* **by fast**
show $G \in \text{set } [G, G'] \ G' \in \text{set } [G, G']$ **by auto**
show $G \neq G'$
proof
assume $GG': G = G'$
from *False assms(1)* **obtain** g **where** $g: g \in G \ g \neq 0$
using *AbGroup.nonempty* **by auto**
moreover **from** *assms(2) GG' g(1)* **have** $-g \in G'$
using *AbGroup.neg-closed* **by fast**

moreover have $g + (-g) = 0$ by *simp*
 ultimately show *False* using *assms(3)* by *force*
 qed
 qed
 qed

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 $\bigoplus Gs \leftarrow = (\lambda x. \text{if } x \in (\bigoplus G \leftarrow Gs. G)$
 then THE } gs. gs \in \text{listset } Gs \wedge x = \text{sum-list } gs \text{ else []})

abbreviation *inner-dirsum-el-decomp-double* ::
 $'g::\text{ab-group-add set} \Rightarrow 'g \text{ set} \Rightarrow ('g \Rightarrow 'g \text{ list}) (-\bigoplus \leftarrow)$ **where** $G \bigoplus H \leftarrow \equiv \bigoplus [G, H] \leftarrow$

abbreviation *inner-dirsum-el-decomp-nth* ::
 $'g::\text{ab-group-add set list} \Rightarrow \text{nat} \Rightarrow ('g \Rightarrow 'g) (\bigoplus \downarrow)$
where $\bigoplus Gs \downarrow n \equiv \text{restrict0 } (\lambda x. (\bigoplus Gs \leftarrow x)!n) (\bigoplus G \leftarrow Gs. G)$

lemma *AbGroup-inner-dirsum-el-decompI* :
 $\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; x \in (\bigoplus G \leftarrow Gs. G) \rrbracket$
 $\implies (\bigoplus Gs \leftarrow x) \in \text{listset } Gs \wedge x = \text{sum-list } (\bigoplus Gs \leftarrow x)$
using *AbGroup-inner-dirsum-el-decomp-ex1* *theI* [
 of } \lambda gs. gs \in \text{listset } Gs \wedge x = \text{sum-list } gs
]
unfolding *inner-dirsum-el-decomp-def*
by *fastforce*

lemma (*in AbGroup*) *abSubgroup-inner-dirsum-el-decomp-set* :
 $\llbracket \forall H \in \text{set } Hs. \text{Subgroup } H; \text{add-independentS } Hs; x \in (\bigoplus H \leftarrow Hs. H) \rrbracket$
 $\implies \text{set } (\bigoplus Hs \leftarrow x) \subseteq G$
using *AbGroup.intro* *AbGroup-inner-dirsum-el-decompI* [*of } Hs } x]*
 set-listset-el-subset [*of } (\bigoplus Hs \leftarrow x) } Hs } G]*
by *fast*

lemma *AbGroup-inner-dirsum-el-decomp-eq* :
 $\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; x \in (\bigoplus G \leftarrow Gs. G);$
 $gs \in \text{listset } Gs; x = \text{sum-list } gs \rrbracket \implies (\bigoplus Gs \leftarrow x) = gs$
using *AbGroup-inner-dirsum-el-decomp-ex1* [*of } Gs]*
 inner-dirsum-el-decomp-def [*of } Gs]*
by *force*

lemma *AbGroup-inner-dirsum-el-decomp-plus* :
assumes $\forall G \in \text{set } Gs. \text{AbGroup } G \text{ add-independentS } Gs$ $x \in (\bigoplus G \leftarrow Gs. G)$
 $y \in (\bigoplus G \leftarrow Gs. G)$
shows $(\bigoplus Gs \leftarrow (x+y)) = [a+b. (a,b) \leftarrow \text{zip } (\bigoplus Gs \leftarrow x) (\bigoplus Gs \leftarrow y)]$
proof –
define *xs ys* **where** $xs = (\bigoplus Gs \leftarrow x)$ **and** $ys = (\bigoplus Gs \leftarrow y)$

with *assms*
have *xy*: $xs \in \text{listset } Gs$ $x = \text{sum-list } xs$ $ys \in \text{listset } Gs$ $y = \text{sum-list } ys$
using *AbGroup-inner-dirsum-el-decompI*
by *auto*
from *assms(1)* *xy(1,3)* **have** $[a+b. (a,b)\leftarrow\text{zip } xs \ ys] \in \text{listset } Gs$
using *AbGroup.axioms* *listset-Group-plus-closed* **by** *fast*
moreover from *xy* **have** $x + y = \text{sum-list } [a+b. (a,b)\leftarrow\text{zip } xs \ ys]$
using *listset-length[of xs Gs]* *listset-length[of ys Gs, THEN sym]* *sum-list-plus*
by *simp*
ultimately show $(\bigoplus Gs\leftarrow(x+y)) = [a+b. (a,b)\leftarrow\text{zip } xs \ ys]$
using *assms* *AbGroup-inner-dirsum* *AbGroup.add-closed*
AbGroup-inner-dirsum-el-decomp-eq
by *fast*
qed

lemma *AbGroup-length-inner-dirsum-el-decomp* :
 $\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; x \in (\bigoplus G\leftarrow Gs. G) \rrbracket$
 $\implies \text{length } (\bigoplus Gs\leftarrow x) = \text{length } Gs$
using *AbGroup-inner-dirsum-el-decompI* *listset-length* **by** *fastforce*

lemma *AbGroup-inner-dirsum-el-decomp-in-nth* :
assumes $\forall G \in \text{set } Gs. \text{AbGroup } G$ *add-independentS* Gs $n < \text{length } Gs$
 $x \in Gs!n$
shows $(\bigoplus Gs\leftarrow x) = (\text{replicate } (\text{length } Gs) \ 0)[n := x]$
proof –
from *assms* **have** $x: x \in (\bigoplus G\leftarrow Gs. G)$
using *AbGroup-nth-subset-inner-dirsum* **by** *fast*
define *xgs* **where** $xgs = (\text{replicate } (\text{length } Gs) \ (0::'a))[n := x]$
hence $\text{length } xgs = \text{length } Gs$ **by** *simp*
moreover have $\forall k < \text{length } xgs. xgs!k \in Gs!k$
proof –
have $\bigwedge k. k < \text{length } xgs \implies xgs!k \in Gs!k$
proof –
fix *k* **assume** $k < \text{length } xgs$
with *assms(1,4)* *xgs-def* **show** $xgs!k \in Gs!k$
using *AbGroup.zero-closed[of Gs!k]* **by** (*cases k = n*) *auto*
qed
thus *?thesis* **by** *fast*
qed
ultimately have $xgs \in \text{listset } Gs$ **using** *listsetI-nth* **by** *fast*
moreover from *xgs-def* *assms(3)* **have** $x = \text{sum-list } xgs$
using *sum-list-update[of n replicate (length Gs) 0 x]* *nth-replicate* *sum-list-replicate0*
by *simp*
ultimately show $(\bigoplus Gs\leftarrow x) = xgs$
using *assms(1,2)* *x* *xgs-def* *AbGroup-inner-dirsum-el-decomp-eq* **by** *fast*
qed

lemma *AbGroup-inner-dirsum-el-decomp-nth-in-nth* :
 $\llbracket \forall G \in \text{set } Gs. \text{AbGroup } G; \text{add-independentS } Gs; k < \text{length } Gs; \rrbracket$

$n < \text{length } Gs; x \in Gs!n \implies (\bigoplus Gs \downarrow k) x = (\text{if } k = n \text{ then } x \text{ else } 0)$
using *AbGroup-nth-subset-inner-dirsum*
 $AbGroup\text{-inner-dirsum-el-decomp-in-nth}[of\ Gs\ n\ x]$
by *auto*

lemma *AbGroup-inner-dirsum-el-decomp-nth-id-on-nth* :
 $\llbracket \forall G \in \text{set } Gs. AbGroup\ G; \text{add-independentS } Gs; n < \text{length } Gs; x \in Gs!n \rrbracket$
 $\implies (\bigoplus Gs \downarrow n) x = x$
using *AbGroup-inner-dirsum-el-decomp-nth-in-nth* **by** *fastforce*

lemma *AbGroup-inner-dirsum-el-decomp-nth-onto-nth* :
assumes $\forall G \in \text{set } Gs. AbGroup\ G\ \text{add-independentS } Gs\ n < \text{length } Gs$
shows $(\bigoplus Gs \downarrow n) \text{ ‘ } (\bigoplus G \leftarrow Gs. G) = Gs!n$
proof
from *assms* **show** $(\bigoplus Gs \downarrow n) \text{ ‘ } (\bigoplus G \leftarrow Gs. G) \supseteq Gs!n$
using *AbGroup-nth-subset-inner-dirsum*[*of Gs n*]
 $AbGroup\text{-inner-dirsum-el-decomp-nth-id-on-nth}[of\ Gs\ n]$
by *force*
from *assms* **show** $(\bigoplus Gs \downarrow n) \text{ ‘ } (\bigoplus G \leftarrow Gs. G) \subseteq Gs!n$
using *AbGroup-inner-dirsum-el-decompI listset-length listsetD-nth* **by** *fastforce*
qed

lemma *AbGroup-inner-dirsum-subset-proj-eq-0* :
assumes $Gs \neq \llbracket \forall G \in \text{set } Gs. AbGroup\ G\ \text{add-independentS } Gs$
 $X \subseteq (\bigoplus G \leftarrow Gs. G) \forall i < \text{length } Gs. (\bigoplus Gs \downarrow i) \text{ ‘ } X = 0$
shows $X = 0$
proof–
have $X \subseteq 0$
proof
fix x **assume** $x \in X$
with *assms*(2–4) **have** $x = (\sum i=0..<\text{length } Gs. (\bigoplus Gs \downarrow i) x)$
using *AbGroup-inner-dirsum-el-decompI sum-list-sum-nth*[*of* $(\bigoplus Gs \leftarrow x)$]
 $AbGroup\text{-length-inner-dirsum-el-decomp}$
by *fastforce*
moreover from x *assms*(5) **have** $\forall i < \text{length } Gs. (\bigoplus Gs \downarrow i) x = 0$ **by** *auto*
ultimately show $x \in 0$ **by** *simp*
qed
moreover from *assms*(1,5) **have** $X \neq \{\}$ **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *GroupEnd-inner-dirsum-el-decomp-nth* :
assumes $\forall G \in \text{set } Gs. AbGroup\ G\ \text{add-independentS } Gs\ n < \text{length } Gs$
shows $GroupEnd\ (\bigoplus G \leftarrow Gs. G) (\bigoplus Gs \downarrow n)$
proof (*intro-locales*)
from *assms*(1) **show** $grp: Group\ (\bigoplus G \leftarrow Gs. G)$
using *AbGroup-inner-dirsum AbGroup.axioms* **by** *fast*
show $GroupHom\text{-axioms } (\bigoplus G \leftarrow Gs. G) \bigoplus Gs \downarrow n$
proof

```

  show  $\text{supp } (\bigoplus Gs \downarrow n) \subseteq (\bigoplus G \leftarrow Gs. G)$  using supp-restrict0 by fast
next
  fix  $x y$  assume  $xy: x \in (\bigoplus G \leftarrow Gs. G) y \in (\bigoplus G \leftarrow Gs. G)$ 
  with assms(1,2) have  $(\bigoplus Gs \leftarrow (x+y)) = [x+y. (x,y) \leftarrow \text{zip } (\bigoplus Gs \leftarrow x) (\bigoplus Gs \leftarrow y)]$ 
    using AbGroup-inner-dirsum-el-decomp-plus by fast
  hence  $(\bigoplus Gs \leftarrow (x+y)) = \text{map } (\text{case-prod } (+)) (\text{zip } (\bigoplus Gs \leftarrow x) (\bigoplus Gs \leftarrow y))$ 
    using concat-map-split-eq-map-split-zip by simp
  moreover from assms xy
    have  $n < \text{length } (\bigoplus Gs \leftarrow x) \ n < \text{length } (\bigoplus Gs \leftarrow y)$ 
       $n < \text{length } (\text{zip } (\bigoplus Gs \leftarrow x) (\bigoplus Gs \leftarrow y))$ 
    using AbGroup-length-inner-dirsum-el-decomp[of Gs x]
      AbGroup-length-inner-dirsum-el-decomp[of Gs y]
    by auto
  ultimately show  $(\bigoplus Gs \downarrow n) (x + y) = (\bigoplus Gs \downarrow n) x + (\bigoplus Gs \downarrow n) y$ 
    using xy assms(1) AbGroup-inner-dirsum
      AbGroup.add-closed[of \bigoplus G \leftarrow Gs. G]
    by auto
  qed
  show GroupEnd-axioms  $(\bigoplus G \leftarrow Gs. G) \bigoplus Gs \downarrow n$ 
    using assms AbGroup-inner-dirsum-el-decomp-nth-onto-nth AbGroup-nth-subset-inner-dirsum
    by unfold-locales fast
  qed

```

2.6 Rings

2.6.1 Preliminaries

```

lemma (in ring-1) map-times-neg1-eq-map-uminus :  $[(-1)*r. r \leftarrow rs] = [-r. r \leftarrow rs]$ 
  using map-eq-conv by simp

```

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  $R :: 'r::\text{ring-1 set}$ 
+ 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
  using Ring1-axioms Ring1.axioms(1) AbGroup.intro by fast

```

```

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 \Rightarrow bool **where** *Subring1* S \equiv *Ring1* S \wedge S \subseteq R

lemma *Subring1D1* : *Subring1* S \Longrightarrow *Ring1* S **by** *fast*

end

lemma (in *ring-1*) *full-Ring1* : *Ring1* UNIV
by *unfold-locales auto*

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* \equiv *aezfun-setspace* G

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

lemma *RG-one-closed* : (1::('r::zero-neq-one, 'g) aezfun) \in *group-ring*

proof–

have *supp* (*aezfun* (1::('r, 'g) aezfun)) \subseteq G

using *supp-aezfun1 zeroS-closed* **by** *fast*

thus *?thesis* **using** *group-ringD* **by** *fast*

qed

lemma *RG-zero-closed* : (0::('r::zero, 'g) aezfun) \in *group-ring*

proof–

have *aezfun* (0::('r, 'g) aezfun) = (0::'g \Rightarrow 'r) **using** *zero-aezfun.rep-eq* **by** *fast*

hence *supp* (*aezfun* (0::('r, 'g) aezfun)) = *supp* (0::'g \Rightarrow 'r) **by** *simp*

moreover **have** *supp* (0::'g \Rightarrow 'r) \subseteq G **using** *supp-zerofun-subset-any* **by** *fast*

ultimately show *?thesis* **using** *group-ringD* **by** *fast*

qed

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

using *RG-one-closed zero-neq-one* **by** *force*

```

lemma RG-mult-closed :
  defines RG:  $RG \equiv \text{group-ring} :: ('r::\text{ring-1}, 'g) \text{ aezfun set}$ 
  shows  $x \in RG \implies y \in RG \implies x * y \in RG$ 
  using RG supp-aezfun-times[of x y]
          set-plus-closed[of supp (aezfun x) supp (aezfun y)]
          group-ringD
  by blast

lemma Ring1-RG :
  defines RG:  $RG \equiv \text{group-ring} :: ('r::\text{ring-1}, 'g) \text{ aezfun set}$ 
  shows Ring1 RG
proof
  from RG show  $RG \neq \{\}$   $1 \in RG \wedge x y. x \in RG \implies y \in RG \implies x * y \in RG$ 
    using RG-one-closed RG-mult-closed by auto
next
  fix x y assume  $x \in RG y \in RG$ 
  with RG show  $x - y \in RG$  using supp-aezfun-diff[of x y] group-ringD by blast
qed

lemma RG-aezdeltafun-closed :
  defines RG:  $RG \equiv \text{group-ring} :: ('r::\text{ring-1}, 'g) \text{ aezfun set}$ 
  assumes  $g \in G$ 
  shows  $r \delta \delta g \in RG$ 
proof -
  have supp:  $\text{supp} (\text{aezfun} (r \delta \delta g)) = \text{supp} (r \delta g)$ 
    using aezdeltafun arg-cong[of - - supp] by fast
  have  $\text{supp} (\text{aezfun} (r \delta \delta g)) \subseteq G$ 
  proof (cases r = 0)
    case True with supp show ?thesis using supp-delta0fun by fast
  next
    case False with assms supp show ?thesis using supp-deltafun[of r g] by fast
  qed
with RG show ?thesis using group-ringD by fast
qed

lemma RG-aezdelta0fun-closed :  $(r::'\text{ring-1}) \delta \delta 0 \in \text{group-ring}$ 
  using zero-closed RG-aezdeltafun-closed[of 0] by fast

lemma RG-sum-list-rddg-closed :
  defines RG:  $RG \equiv \text{group-ring} :: ('r::\text{ring-1}, 'g) \text{ aezfun set}$ 
  assumes  $\text{set} (\text{map} \text{snd} \text{rgs}) \subseteq G$ 
  shows  $(\sum (r,g) \leftarrow \text{rgs}. r \delta \delta g) \in RG$ 
proof (rule Ring1.sum-list-closed-prod)
  from RG show Ring1 RG using Ring1-RG by fast
  from assms show  $\text{set} (\text{map} (\text{case-prod} \text{aezdeltafun}) \text{rgs}) \subseteq RG$ 
  using RG-aezdeltafun-closed by fastforce
qed

```

lemmas $RG\text{-el-decomp-aezdeltafun} = \text{aezfun-sets\text{-}span-el-decomp-aezdeltafun}[\text{of } G]$

lemma $Subgroup\text{-}imp\text{-}Subring$:

fixes $H :: 'g \text{ set}$

and $FH :: ('r::ring\text{-}1, 'g) \text{ aezfun set}$

and $FG :: ('r, 'g) \text{ aezfun set}$

defines $FH \equiv Group.group\text{-}ring H$

and $FG \equiv group\text{-}ring$

shows $Subgroup H \implies Ring1.Subring1 FG FH$

using $assms Group.Ring1\text{-}RG Group.RG\text{-}el\text{-}decomp\text{-}aezdeltafun RG\text{-}sum\text{-}list\text{-}rddg\text{-}closed$

by $fast$

end

lemma (**in** $FinGroup$) $group\text{-}ring\text{-}spanning\text{-}set$:

$\exists gs. distinct\ gs \wedge set\ gs = G$

$\wedge (\forall f \in (group\text{-}ring :: ('b::semiring\text{-}1, 'g) \text{ aezfun set}). \exists bs.$

$length\ bs = length\ gs \wedge f = (\sum (b, g) \leftarrow zip\ bs\ gs. (b\ \delta\delta\ 0) * (1\ \delta\delta\ g)))$

using $finite\ aezfun\text{-}common\text{-}supp\text{-}spanning\text{-}set\text{-}decomp[\text{of } G] group\text{-}ringD$

by $fast$

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

context $Group$

begin

abbreviation $RG\text{-}proj \equiv \text{aezfun-sets\text{-}span-proj } G$

lemmas $RG\text{-}proj\text{-}in\text{-}RG = \text{aezfun-sets\text{-}span-proj-in-sets\text{-}span}$

lemmas $RG\text{-}proj\text{-}sum\text{-}list\text{-}prod = \text{aezfun-sets\text{-}span-proj-sum\text{-}list-prod}[\text{of } G]$

lemma $RG\text{-}proj\text{-}mult\text{-}leftdelta'$:

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

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

using $add\text{-}closed\ add\text{-}closed\text{-}inverse\text{-}right\ \text{times-aezdeltafun-aezdeltafun}[\text{of } r\ g]$

$aezfun\text{-}sets\text{-}span\text{-}proj\text{-}aezdeltafun[\text{of } G\ r*s]$

$aezfun\text{-}sets\text{-}span\text{-}proj\text{-}aezdeltafun[\text{of } G\ s]$

by $simp$

lemma $RG\text{-}proj\text{-}mult\text{-}leftdelta$:

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

assumes $g \in G$

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

proof –

from $aezfun\text{-}decomp\text{-}aezdeltafun$ **obtain** rgs

where $rgs: x = (\sum (s, h) \leftarrow rgs. s\ \delta\delta\ h)$

using $RG\text{-}el\text{-}decomp\text{-}aezdeltafun$

by $fast$

hence $RG\text{-proj} ((r \delta\delta g) * x) = (\sum (s,h)\leftarrow rgs. RG\text{-proj} ((r \delta\delta g) * (s \delta\delta h)))$
using $sum\text{-list-const-mult-prod}[of\ r\ \delta\delta\ g\ \lambda s\ h. s\ \delta\delta\ h]$ $RG\text{-proj-sum-list-prod}$
by $simp$
also from $assms\ rgs$ **have** $\dots = (r \delta\delta g) * RG\text{-proj}\ x$
using $RG\text{-proj-mult-leftdelta}'[of\ g\ r]$
 $sum\text{-list-const-mult-prod}[of\ r\ \delta\delta\ g\ \lambda s\ h. RG\text{-proj}\ (s\ \delta\delta\ h)]$
 $RG\text{-proj-sum-list-prod}[of\ \lambda s\ h. s\ \delta\delta\ h\ rgs]$
by $simp$
finally show $?thesis$ **by** $fast$
qed

lemma $RG\text{-proj-mult-rightdelta}' :$
fixes $r\ s :: 'r::\{comm\text{-monoid-add,mult-zero}\}$
assumes $g' \in G$
shows $RG\text{-proj}\ (r\ \delta\delta\ g * (s\ \delta\delta\ g')) = RG\text{-proj}\ (r\ \delta\delta\ g) * (s\ \delta\delta\ g')$
using $assms\ times\ aezdeltafun\ aezdeltafun[of\ r\ g]$
 $aezfun\ setspan\ proj\ aezdeltafun[of\ G\ r*s]$
 $add\ closed\ add\ closed\ inverse\ left\ aezfun\ setspan\ proj\ aezdeltafun[of\ G\ r]$
by $simp$

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

proof –
from $aezfun\ decomp\ aezdeltafun$ **obtain** rgs
where $rgs: x = (\sum (s,h)\leftarrow rgs. s\ \delta\delta\ h)$
using $RG\text{-el-decomp-aezdeltafun}$
by $fast$
hence $RG\text{-proj}\ (x * (r \delta\delta\ g)) = (\sum (s,h)\leftarrow rgs. RG\text{-proj}\ ((s \delta\delta\ h) * (r \delta\delta\ g)))$
using $sum\text{-list-mult-const-prod}[of\ \lambda s\ h. s\ \delta\delta\ h\ rgs]$ $RG\text{-proj-sum-list-prod}$
by $simp$
with $assms\ rgs$ **show** $?thesis$
using $RG\text{-proj-mult-rightdelta}'[of\ g\ -\ -\ r]$
 $sum\text{-list-prod-cong}[of$
 $rgs\ \lambda s\ h. RG\text{-proj}\ ((s\ \delta\delta\ h) * (r\ \delta\delta\ g))$
 $\lambda s\ h. RG\text{-proj}\ (s\ \delta\delta\ h) * (r\ \delta\delta\ g)$
 $]$
 $sum\text{-list-mult-const-prod}[of\ \lambda s\ h. RG\text{-proj}\ (s\ \delta\delta\ h)\ rgs]$
 $RG\text{-proj-sum-list-prod}[of\ \lambda s\ h. s\ \delta\delta\ h\ rgs]$
 $sum\text{-list-mult-const-prod}[of\ \lambda s\ h. RG\text{-proj}\ (s\ \delta\delta\ h)\ rgs\ r\ \delta\delta\ g]$
 $RG\text{-proj-sum-list-prod}[of\ \lambda s\ h. s\ \delta\delta\ h\ rgs]$
by $simp$
qed

lemma $RG\text{-proj-mult-right} :$
 $x \in (group\text{-ring} :: ('r::ring-1, 'g)\ aezfun\ set)$
 $\implies RG\text{-proj}\ (y * x) = RG\text{-proj}\ y * x$
using $RG\text{-el-decomp-aezdeltafun}\ sum\text{-list-const-mult-prod}[of\ y\ \lambda r\ g. r\ \delta\delta\ g]$


```

    RG-proj-sum-list-prod[of  $\lambda r g. y * (r \delta\delta g)$ ] RG-proj-mult-rightdelta[of - y]
    sum-list-prod-cong[
      of -  $\lambda r g. RG\text{-proj } (y * (r \delta\delta g)) \lambda r g. RG\text{-proj } y * (r \delta\delta g)$ 
    ]
  sum-list-const-mult-prod[of  $RG\text{-proj } y \lambda r g. r \delta\delta g$ ]
  by fastforce

```

end

3 Modules

3.1 Locales and basic facts

3.1.1 Locales

```

locale scalar-mult =
  fixes smult :: 'r::ring-1  $\Rightarrow$  'm::ab-group-add  $\Rightarrow$  'm (infixr  $\cdot$  70)

```

```

locale R-scalar-mult = scalar-mult smult + Ring1 R
  for R    :: 'r::ring-1 set
  and smult :: 'r  $\Rightarrow$  'm::ab-group-add  $\Rightarrow$  'm (infixr  $\cdot$  70)

```

```

lemma (in scalar-mult) R-scalar-mult : R-scalar-mult UNIV
  using full-Ring1 R-scalar-mult.intro by fast

```

```

lemma (in R-scalar-mult) Ring1 : Ring1 R ..

```

```

locale RModule = R-scalars?: R-scalar-mult R smult + VecGroup?: Group M
  for R    :: 'r::ring-1 set
  and smult :: 'r  $\Rightarrow$  'm::ab-group-add  $\Rightarrow$  'm (infixr  $\cdot$  70)
  and M    :: 'm set
+ assumes smult-closed :  $\llbracket r \in R; m \in M \rrbracket \Longrightarrow r \cdot m \in M$ 
  and smult-distrib-left [simp] :  $\llbracket r \in R; m \in M; n \in M \rrbracket$ 
     $\Longrightarrow r \cdot (m + n) = r \cdot m + r \cdot n$ 
  and smult-distrib-right [simp] :  $\llbracket r \in R; s \in R; m \in M \rrbracket$ 
     $\Longrightarrow (r + s) \cdot m = r \cdot m + s \cdot m$ 
  and smult-assoc [simp] :  $\llbracket r \in R; s \in R; m \in M \rrbracket$ 
     $\Longrightarrow r \cdot s \cdot m = (r * s) \cdot m$ 
  and one-smult [simp] :  $m \in M \Longrightarrow 1 \cdot m = m$ 

```

```

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 (*rule* *RModuleI*, *rule* *assms(1)*, *rule* *trivial-Group*, *unfold-locales*)

define *Z* **where** *Z* = ($0::'m\ set$)

fix *r s m n* **assume** *rsmn*: $r \in R\ s \in R\ m \in Z\ n \in Z$

from *rsmn(1,3)* *Z-def* *assms(2)* **show** $r \cdot m \in Z$ **by** *simp*

from *rsmn(1,3,4)* *Z-def* *assms(2)* **show** $r \cdot (m+n) = r \cdot m + r \cdot n$ **by** *simp*

from *rsmn(1-3)* *Z-def* *assms* **show** $(r + s) \cdot m = r \cdot m + s \cdot m$

using *Ring1.add-closed* **by** *auto*

from *rsmn(1-3)* *Z-def* *assms* **show** $r \cdot (s \cdot m) = (r*s) \cdot m$

using *Ring1.mult-closed* **by** *auto*

next

define *Z* **where** *Z* = ($0::'m\ set$)

fix *m* **assume** $m \in Z$ **with** *Z-def* *assms* **show** $1 \cdot m = m$

using *Ring1.one-closed* **by** *auto*

qed

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*

using *RModule-axioms* *RModule.axioms(2)* **by** *fast*

lemma *Subgroup-RSubmodule* : *RSubmodule* *N* \Longrightarrow *Subgroup* *N*

using *RModule.Group* **by** *fast*

lemma *AbGroup* : *AbGroup* *M*

using *AbGroup.intro* *Group* **by** *fast*

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* :

$r \in R \Longrightarrow set\ ms \subseteq M \Longrightarrow set\ (map\ ((\cdot)\ r)\ ms) \subseteq M$

using *smult-closed* **by** (*induct* *ms*) *auto*

lemma *zero-smult* : $m \in M \Longrightarrow 0 \cdot m = 0$

using *R-scalars.zero-closed* *smult-distrib-right[of 0]* *add-left-imp-eq* **by** *simp*

lemma *smult-zero* : $r \in R \Longrightarrow r \cdot 0 = 0$

using *zero-closed* *smult-distrib-left[of r 0]* **by** *simp*

lemma *neg-smult* : $r \in R \implies m \in M \implies (-r) \cdot m = - (r \cdot m)$
using *R-scalars.neg-closed smult-distrib-right*[of $r -r m$]
zero-smult minus-unique[of $r \cdot m$]
by *simp*

lemma *neg-eq-neg1-smult* : $m \in M \implies (-1) \cdot m = - m$
using *one-closed neg-smult one-smult* **by** *fastforce*

lemma *smult-neg* : $r \in R \implies m \in M \implies r \cdot (-m) = - (r \cdot m)$
using *neg-eq-neg1-smult one-closed R-scalars.neg-closed smult-assoc*[of $r - 1$]
smult-closed
by *force*

lemma *smult-distrib-left-diff* :
 $\llbracket r \in R; m \in M; n \in M \rrbracket \implies r \cdot (m - n) = r \cdot m - r \cdot n$
using *neg-closed smult-distrib-left*[of $r m -n$] *smult-neg* **by** (*simp add: algebra-simps*)

lemma *smult-distrib-right-diff* :
 $\llbracket r \in R; s \in R; m \in M \rrbracket \implies (r - s) \cdot m = r \cdot m - s \cdot m$
using *R-scalars.neg-closed smult-distrib-right*[of $r -s$] *neg-smult*
by (*simp add: algebra-simps*)

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 (*induct set: finite*)
case *empty* **from** *assms* **show** *?case* **using** *smult-zero* **by** *simp*
next
case (*insert a A*) **with** *assms* **show** *?case* **using** *sum-closed*[of A] **by** *simp*
qed

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 (*induct set: finite*)
case *empty* **from** *assms* **show** *?case* **using** *zero-smult* **by** *simp*
next
case (*insert a A*) **with** *assms* **show** *?case* **using** *R-scalars.sum-closed*[of A] **by** *simp*
qed

lemma *smult-sum-list-distrib* :
 $r \in R \implies \text{set } ms \subseteq M \implies r \cdot (\text{sum-list } ms) = (\sum m \leftarrow ms. r \cdot m)$
using *smult-zero sum-list-closed*[of *id*] **by** (*induct ms*) *auto*

lemma *sum-list-prod-map-smult-distrib* :
 $m \in M \implies \text{set } (\text{map } (\text{case-prod } f) \text{ } xys) \subseteq R$

$\implies (\sum (x,y)\leftarrow xys. f x y) \cdot m = (\sum (x,y)\leftarrow xys. f x y \cdot m)$
using *zero-smult R-scalars.sum-list-closed-prod[of f]*
by *(induct xys) auto*

lemma *RSubmoduleI* :

assumes *Subgroup N* $\bigwedge r n. r \in R \implies n \in N \implies r \cdot n \in N$
shows *RSubmodule N*

proof

show *RModule R smult N*

proof *(intro-locales, rule SubgroupD1[OF assms(1)], unfold-locales)*

from *assms(2)* **show** $\bigwedge r m. r \in R \implies m \in N \implies r \cdot m \in N$ **by** *fast*

from *assms(1)*

show $\bigwedge r m n. \llbracket r \in R; m \in N; n \in N \rrbracket \implies r \cdot (m + n) = r \cdot m + r \cdot n$
using *smult-distrib-left*

by *blast*

from *assms(1)*

show $\bigwedge r s m. \llbracket r \in R; s \in R; m \in N \rrbracket \implies (r + s) \cdot m = r \cdot m + s \cdot m$
using *smult-distrib-right*

by *blast*

from *assms(1)*

show $\bigwedge r s m. \llbracket r \in R; s \in R; m \in N \rrbracket \implies r \cdot s \cdot m = (r * s) \cdot m$
using *smult-assoc*

by *blast*

from *assms(1)* **show** $\bigwedge m. m \in N \implies 1 \cdot m = m$ **using** *one-smult* **by** *blast*

qed

from *assms(1)* **show** $N \subseteq M$ **by** *fast*

qed

end

lemma *(in R-scalar-mult) listset-RModule-Rsmult-closed* :

$\llbracket \forall M \in \text{set } Ms. RModule R smult M; r \in R; ms \in \text{listset } Ms \rrbracket$
 $\implies [r \cdot m. m \leftarrow ms] \in \text{listset } Ms$

proof–

have $\llbracket \text{length } ms = \text{length } Ms; ms \in \text{listset } Ms;$

$\forall M \in \text{set } Ms. RModule R smult M; r \in R \rrbracket$

$\implies [r \cdot m. m \leftarrow ms] \in \text{listset } Ms$

proof *(induct ms Ms rule: list-induct2)*

case *(Cons m ms M Ms)* **thus** *?case*

using *listset-ConsD[of m] RModule.smult-closed listset-ConsI[of r \cdot m M]*

by *fastforce*

qed *simp*

thus $\llbracket \forall M \in \text{set } Ms. RModule R smult M; r \in R; ms \in \text{listset } Ms \rrbracket$

$\implies [r \cdot m. m \leftarrow ms] \in \text{listset } Ms$

using *listset-length[of ms Ms]* **by** *fast*

qed

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* :
 $(\bigwedge r. r \in R \implies r \cdot 0 = 0) \implies RModule\ R\ smult\ 0$
using *trivial-Group add-closed mult-closed one-closed* **by** *unfold-locales auto*

context *RModule*
begin

lemma *trivial-RSubmodule* : *RSubmodule 0*
using *zeroS-closed smult-zero trivial-RModule* **by** *fast*

lemma *RSubmodule-set-plus* :
assumes *RSubmodule L RSubmodule N*
shows *RSubmodule (L + N)*
proof (rule *RSubmoduleI*)
from *assms* **have** *Group (L + N)*
using *RModule.AbGroup AbGroup-set-plus[of L N] AbGroup.axioms* **by** *fast*
moreover from *assms* **have** $L + N \subseteq M$
using *Group Group.set-plus-closed* **by** *auto*
ultimately show *Subgroup (L + N)* **by** *fast*
next
fix *r x* **assume** *rx: r \in R x \in L + N*
from *rx(2)* **obtain** *m n* **where** *mn: m \in L n \in N x = m + n*
using *set-plus-def[of L N]* **by** *fast*
with *assms rx(1)* **show** $r \cdot x \in L + N$
using *RModule.smult-closed[of R smult L] RModule.smult-closed[of R smult N]*
smult-distrib-left set-plus-def
by *fast*
qed

lemma *RSubmodule-sum-list* :
 $(\forall N \in set\ Ns. RSubmodule\ N) \implies RSubmodule\ (\sum N \leftarrow Ns.\ N)$
using *trivial-RSubmodule RSubmodule-set-plus*
by (induct *Ns*) *auto*

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

```

case True with assms show ?thesis
  using RSubmodule-sum-list inner-dirsumD by fastforce
next
  case False thus ?thesis
  using inner-dirsumD2[of Ns] trivial-RSubmodule by simp
qed

```

```

lemma RModule-inner-dirsum :
   $(\forall N \in \text{set } Ns. \text{RSubmodule } N) \implies \text{RModule } R \text{ smult } (\bigoplus N \leftarrow Ns. N)$ 
  using RSubmodule-inner-dirsum by fast

```

```

lemma SModule-restrict-scalars :
  assumes Subring1 S
  shows RModule S smult M
proof (rule RModuleI, rule Subring1D1[OF assms], rule Group, unfold-locales)
  from assms show
     $\bigwedge r m. r \in S \implies m \in M \implies r \cdot m \in M$ 
     $\bigwedge r m n. r \in S \implies m \in M \implies n \in M \implies r \cdot (m + n) = r \cdot m + r \cdot n$ 
     $\bigwedge m. m \in M \implies 1 \cdot m = m$ 
    using smult-closed smult-distrib-left
    by auto
  from assms
    show  $\bigwedge r s m. r \in S \implies s \in S \implies m \in M \implies (r + s) \cdot m = r \cdot m + s \cdot m$ 
    using Ring1.add-closed smult-distrib-right
    by fast
  from assms
    show  $\bigwedge r s m. r \in S \implies s \in S \implies m \in M \implies r \cdot s \cdot m = (r * s) \cdot m$ 
    using Ring1.mult-closed smult-assoc
    by fast
qed

```

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  $\cdot\cdot$  70)
  where  $rs \cdot\cdot ms = (\sum (r,m) \leftarrow \text{zip } rs \ ms. r \cdot m)$ 

```

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

```

lemma lincomb-Nil :  $rs = [] \vee ms = [] \implies rs \cdot\cdot ms = 0$ 
  unfolding lincomb-def by auto

```

```

lemma lincomb-singles :  $[a] \cdot\cdot [m] = a \cdot m$ 
  using lincomb-def by simp

```

lemma *lincomb-Cons* : $(r \# rs) \cdot (m \# ms) = r \cdot m + rs \cdot ms$
unfolding *lincomb-def* **by** *simp*

lemma *lincomb-append* :
 $length\ rs = length\ ms \implies (rs@ss) \cdot (ms@ns) = rs \cdot ms + ss \cdot ns$
unfolding *lincomb-def* **by** *simp*

lemma *lincomb-append-left* :
 $(rs @ ss) \cdot ms = rs \cdot ms + ss \cdot drop\ (length\ rs)\ ms$
using *zip-append-left*[of *rs ss ms*] **unfolding** *lincomb-def* **by** *simp*

lemma *lincomb-append-right* :
 $rs \cdot (ms@ns) = rs \cdot ms + (drop\ (length\ ms)\ rs) \cdot ns$
using *zip-append-right*[of *rs ms*] **unfolding** *lincomb-def* **by** *simp*

lemma *lincomb-conv-take-right* : $rs \cdot ms = rs \cdot take\ (length\ rs)\ ms$
using *lincomb-Nil lincomb-Cons* **by** (*induct rs ms rule: list-induct2'*) *auto*

end

context *RModule*
begin

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

lemma *lincomb-closed* : $set\ rs \subseteq R \implies set\ ms \subseteq M \implies rs \cdot ms \in M$
proof (*induct ms arbitrary: rs*)
case *Nil* **show** ?*case* **using** *lincomb-Nil zero-closed* **by** *simp*
next
case (*Cons m ms*)
hence *Cons1*: $\bigwedge rs. set\ rs \subseteq R \implies rs \cdot ms \in M\ m \in M\ set\ rs \subseteq R$ **by** *auto*
show $rs \cdot (m\#\ms) \in M$
proof (*cases rs*)
case *Nil* **thus** ?*thesis* **using** *lincomb-Nil zero-closed* **by** *simp*
next
case *Cons* **with** *Cons1* **show** ?*thesis*
using *lincomb-Cons smult-closed add-closed* **by** *fastforce*
qed
qed

lemma *smult-lincomb* :
 $\llbracket set\ rs \subseteq R; s \in R; set\ ms \subseteq M \rrbracket \implies s \cdot (rs \cdot ms) = [s*r. r\leftarrow rs] \cdot ms$
using *lincomb-Nil smult-zero lincomb-Cons smult-closed lincomb-closed*
by (*induct rs ms rule: list-induct2'*) *auto*

lemma *neg-lincomb* :
 $set\ rs \subseteq R \implies set\ ms \subseteq M \implies -(rs \cdot ms) = [-r. r\leftarrow rs] \cdot ms$
using *lincomb-closed neg-eq-neg1-smult one-closed R-scalars.neg-closed*[of *1*]

$\text{smult-lincomb}[of\ rs - 1]\ \text{map-times-neg1-eq-map-uminus}$
by *auto*

lemma *lincomb-sum-left* :
 $\llbracket\ \text{set}\ rs \subseteq R; \text{set}\ ss \subseteq R; \text{set}\ ms \subseteq M; \text{length}\ rs \leq \text{length}\ ss \rrbracket$
 $\implies [r + s. (r,s)\leftarrow\text{zip}\ rs\ ss] \cdot ms = rs \cdot ms + (\text{take}\ (\text{length}\ rs)\ ss) \cdot ms$

proof (*induct rs ss arbitrary: ms rule: list-induct2'*)
case 1 show ?case using lincomb-Nil by simp
next
case (2 r rs)
show $\bigwedge ms. \text{length}\ (r\#\!rs) \leq \text{length}\ []$
 $\implies [a + b. (a,b)\leftarrow\text{zip}\ (r\#\!rs)\ []] \cdot ms$
 $= (r\#\!rs) \cdot ms + (\text{take}\ (\text{length}\ (r\#\!rs))\ []) \cdot ms$
by *simp*
next
case 3 show ?case using lincomb-Nil by simp
next
case (4 r rs s ss)
thus $[a+b. (a,b)\leftarrow\text{zip}\ (r\#\!rs)\ (s\#\!ss)] \cdot ms$
 $= (r\#\!rs) \cdot ms + (\text{take}\ (\text{length}\ (r\#\!rs))\ (s\#\!ss)) \cdot ms$
using *lincomb-Nil lincomb-Cons* **by** (*cases ms*) *auto*
qed

lemma *lincomb-sum* :
assumes $\text{set}\ rs \subseteq R\ \text{set}\ ss \subseteq R\ \text{set}\ ms \subseteq M\ \text{length}\ rs \leq \text{length}\ ss$
shows $rs \cdot ms + ss \cdot ms$
 $= ([a + b. (a,b)\leftarrow\text{zip}\ rs\ ss] @ (\text{drop}\ (\text{length}\ rs)\ ss)) \cdot ms$

proof –
define *zs fss bss*
where $zs = [a + b. (a,b)\leftarrow\text{zip}\ rs\ ss]$
and $fss = \text{take}\ (\text{length}\ rs)\ ss$
and $bss = \text{drop}\ (\text{length}\ rs)\ ss$
from *assms(4) zs-def fss-def* **have** $\text{length}\ zs = \text{length}\ rs\ \text{length}\ fss = \text{length}\ rs$
using *length-concat-map-split-zip*[of $\lambda a\ b. a + b\ rs$] **by** *auto*
hence $(zs @ bss) \cdot ms = rs \cdot ms + (fss @ bss) \cdot ms$
using *assms(1,2,3) zs-def fss-def lincomb-sum-left lincomb-append-left*
by *simp*
thus *thesis* **using** *fss-def bss-def zs-def* **by** *simp*
qed

lemma *lincomb-diff-left* :
 $\llbracket\ \text{set}\ rs \subseteq R; \text{set}\ ss \subseteq R; \text{set}\ ms \subseteq M; \text{length}\ rs \leq \text{length}\ ss \rrbracket$
 $\implies [r - s. (r,s)\leftarrow\text{zip}\ rs\ ss] \cdot ms = rs \cdot ms - (\text{take}\ (\text{length}\ rs)\ ss) \cdot ms$

proof (*induct rs ss arbitrary: ms rule: list-induct2'*)
case 1 show ?case using lincomb-Nil by simp
next
case (2 r rs)
show $\bigwedge ms. \text{length}\ (r\#\!rs) \leq \text{length}\ []$
 $\implies [a - b. (a,b)\leftarrow\text{zip}\ (r\#\!rs)\ []] \cdot ms$

$$= (r\#rs) \cdot ms - (\text{take } (\text{length } (r\#rs)) \ []) \cdot ms$$

by *simp*
next
case 3 show ?case using lincomb-Nil by simp
next
case (4 r rs s ss)
thus $[a-b. (a,b)\leftarrow\text{zip } (r\#rs) (s\#ss)] \cdot ms$
 $= (r\#rs) \cdot ms - (\text{take } (\text{length } (r\#rs)) (s\#ss)) \cdot ms$
using *lincomb-Nil lincomb-Cons smult-distrib-right-diff* **by** *(cases ms) auto*
qed

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 *(induct k arbitrary: ms)*
case 0 thus ?case using lincomb-Nil smult-zero by simp
next
case (Suc k)
show ?case
proof *(cases ms)*
case Nil with Suc(2) show ?thesis using lincomb-Nil smult-zero by simp
next
case (Cons m ms) with Suc show ?thesis
using *lincomb-Cons set-take-subset[of k ms] sum-list-closed[of id]*
by auto
qed
qed

lemma lincomb-replicate0-left : set ms \subseteq M \implies (replicate k 0) \cdot ms = 0
proof –
assume *ms: set ms \subseteq M*
hence $(\text{replicate } k \ 0) \cdot ms = 0 \cdot (\sum m\leftarrow(\text{take } k \ ms). \ m)$
using *R-scalars.zero-closed lincomb-replicate-left* **by fast**
moreover from ms have $(\sum m\leftarrow(\text{take } k \ ms). \ m) \in M$
using *set-take-subset sum-list-closed* **by fastforce**
ultimately show $(\text{replicate } k \ 0) \cdot ms = 0$ **using zero-smult by simp**
qed

lemma lincomb-0coeffs : set ms \subseteq M $\implies \forall s \in \text{set } rs. s = 0 \implies rs \cdot ms = 0$
using *lincomb-Nil lincomb-Cons zero-smult*
by *(induct rs ms rule: list-induct2') auto*

lemma delta-scalars-lincomb-eq-nth :
 $\text{set } ms \subseteq M \implies n < \text{length } ms$
 $\implies ((\text{replicate } (\text{length } ms) \ 0)[n := 1]) \cdot ms = ms!n$
proof *(induct ms arbitrary: n)*
case (Cons m ms) thus ?case
using *lincomb-Cons lincomb-replicate0-left zero-smult* **by** *(cases n) auto*
qed simp

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

$set\ rs \subseteq R \implies set\ ms \subseteq M$
 $\implies \exists ss. set\ ss \subseteq R \wedge length\ ss = length\ ms$
 $\wedge take\ (length\ rs)\ ss = take\ (length\ ms)\ rs \wedge rs \cdot ms = ss \cdot ms$

proof (*induct rs ms rule: list-induct2'*)

case 1 show *?case using lincomb-Nil by simp*

next

case 2 thus *?case using lincomb-Nil by simp*

next

case (*3 m ms*)

define *ss where* $ss = replicate\ (Suc\ (length\ ms))\ (0::'r)$

from *3(2) ss-def*

have $set\ ss \subseteq R\ length\ ss = length\ (m\#\ms) \ [] \cdot (m\#\ms) = ss \cdot (m\#\ms)$

using *R-scalars.zero-closed lincomb-Nil*
lincomb-replicate0-left[of m#ms Suc (length ms)]

by *auto*

thus *?case by auto*

next

case (*4 r rs m ms*)

from *this obtain ss*

where $ss: set\ ss \subseteq R\ length\ ss = length\ ms$
 $take\ (length\ rs)\ ss = take\ (length\ ms)\ rs\ rs \cdot ms = ss \cdot ms$

by *auto*

from *4(2) ss have*

$set\ (r\#\ss) \subseteq R\ length\ (r\#\ss) = length\ (m\#\ms)$
 $take\ (length\ (r\#\rs))\ (r\#\ss) = take\ (length\ (m\#\ms))\ (r\#\rs)$
 $(r\#\rs) \cdot (m\#\ms) = (r\#\ss) \cdot (m\#\ms)$

using *lincomb-Cons*

by *auto*

thus *?case by fast*

qed

lemma *lincomb-concat* :

list-all2 ($\lambda rs\ ms. length\ rs = length\ ms$) *rss mss*
 $\implies (concat\ rss) \cdot (concat\ mss) = (\sum (rs,ms) \leftarrow zip\ rss\ mss. rs \cdot ms)$

using *lincomb-Nil lincomb-append by (induct rss mss rule: list-induct2')* *auto*

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

using *lincomb-append-left set-drop-subset lincomb-replicate0-left[of - 1]* **by** *fast-force*

lemma *lincomb-strip-while-0coeffs* :

assumes $set\ ms \subseteq M$

shows $(strip_while\ ((=)\ 0)\ as) \cdot ms = as \cdot ms$

proof (*induct as rule: rev-induct*)

case (*snoc a as*)

hence *caseassm: strip-while ((=) 0) as · ms = as · ms by fast*

show *?case*

proof (*cases a = 0*)

```

case True
moreover with assms have  $(as@[a]) \cdot ms = as \cdot ms$ 
  using lincomb-snoc0 by fast
ultimately show strip-while  $((=) 0) (as @ [a]) \cdot ms = (as@[a]) \cdot ms$ 
  using caseasm by simp
qed simp
qed simp

```

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) = {  $r \cdot m \mid r. r \in R$  } + RSpan ms

```

```

lemma RSpan-single : RSpan [m] = {  $r \cdot m \mid r. r \in R$  }
  using add-0-right[of {  $r \cdot m \mid r. r \in R$  }] by simp

```

```

lemma RSpan-Cons : RSpan (m#ms) = RSpan [m] + RSpan ms
  using RSpan-single by simp

```

```

lemma in-RSpan-obtain-same-length-coeffs :
   $n \in RSpan\ ms \implies \exists rs. set\ rs \subseteq R \wedge length\ rs = length\ ms \wedge n = rs \cdot ms$ 

```

```

proof (induct ms arbitrary: n)

```

```

  case Nil

```

```

    hence  $n = 0$  by simp

```

```

    thus  $\exists rs. set\ rs \subseteq R \wedge length\ rs = length\ [] \wedge n = rs \cdot []$ 

```

```

    using lincomb-Nil by simp

```

```

  next

```

```

    case (Cons m ms)

```

```

    from this obtain r rs

```

```

    where  $set\ (r\#\rs) \subseteq R \wedge length\ (r\#\rs) = length\ (m\#\ms) \wedge n = (r\#\rs) \cdot (m\#\ms)$ 

```

```

    using set-plus-def[of - RSpan ms] lincomb-Cons

```

```

    by fastforce

```

```

    thus  $\exists rs. set\ rs \subseteq R \wedge length\ rs = length\ (m\#\ms) \wedge n = rs \cdot (m\#\ms)$  by fast

```

```

  qed

```

```

lemma in-RSpan-Cons-obtain-same-length-coeffs :

```

```

   $n \in RSpan\ (m\#\ms) \implies \exists r\ rs. set\ (r\#\rs) \subseteq R \wedge length\ rs = length\ ms$ 

```

```

   $\wedge n = r \cdot m + rs \cdot ms$ 

```

```

proof –

```

```

  assume  $n \in RSpan\ (m\#\ms)$ 

```

from this obtain $x y$ **where** $x \in RSpan\ [m]$ $y \in RSpan\ ms$ $n = x + y$
using *RSpan-Cons set-plus-def*[of *RSpan [m]*] **by** *auto*
thus $\exists r\ rs.\ set\ (r\ \# \ rs) \subseteq R \wedge length\ rs = length\ ms \wedge n = r \cdot m + rs \cdot ms$
using *RSpan-single in-RSpan-obtain-same-length-coeffs*[of $y\ ms$] **by** *auto*
qed

lemma *RSpanD-lincomb* :

$RSpan\ ms = \{ rs \cdot ms \mid rs.\ set\ rs \subseteq R \wedge length\ rs = length\ ms \}$

proof

show $RSpan\ ms \subseteq \{ rs \cdot ms \mid rs.\ set\ rs \subseteq R \wedge length\ rs = length\ ms \}$

using *in-RSpan-obtain-same-length-coeffs* **by** *fast*

show $\{ rs \cdot ms \mid rs.\ set\ rs \subseteq R \wedge length\ rs = length\ ms \} \subseteq RSpan\ ms$

proof

fix x **assume** $x \in \{ rs \cdot ms \mid rs.\ set\ rs \subseteq R \wedge length\ rs = length\ ms \}$

from this obtain rs **where** $rs:\ set\ rs \subseteq R \wedge length\ rs = length\ ms \wedge x = rs \cdot ms$

by *fast*

from $rs(2)$ **have** $set\ rs \subseteq R \implies rs \cdot ms \in RSpan\ ms$

using *lincomb-Nil lincomb-Cons* **by** (*induct rs ms rule: list-induct2*) *auto*

with $rs(1,3)$ **show** $x \in RSpan\ ms$ **by** *fast*

qed

qed

lemma *RSpan-append* : $RSpan\ (ms\ @\ ns) = RSpan\ ms + RSpan\ ns$

proof (*induct ms*)

case *Nil* **show** *?case* **using** *add-0-left*[of *RSpan ns*] **by** *simp*

next

case (*Cons m ms*) **thus** *?case*

using *RSpan-Cons*[of $m\ ms@ns$] *add.assoc* **by** *fastforce*

qed

end

context *scalar-mult*

begin

abbreviation $Span \equiv R\text{-scalar-mult}.RSpan\ UNIV\ smult$

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 :  $m \in M \implies m \in \text{RSpan } [m]$ 
  using one-closed RSpan-single by fastforce

lemma RSpan-single-nonzero :  $m \in M \implies m \neq 0 \implies \text{RSpan } [m] \neq 0$ 
  using RSpan-contains-spanset-single by auto

lemma Group-RSpan-single :
  assumes  $m \in M$ 
  shows  $\text{Group } (\text{RSpan } [m])$ 
proof
  from assms show  $\text{RSpan } [m] \neq \{ \}$  using RSpan-contains-spanset-single by fast
next
  fix  $x y$  assume  $x \in \text{RSpan } [m] \ y \in \text{RSpan } [m]$ 
  from this obtain  $r s$  where  $rs$ :  $r \in R \ x = r \cdot m \ s \in R \ y = s \cdot m$ 
  using RSpan-single by auto
  with assms have  $x - y = (r - s) \cdot m$  using smult-distrib-right-diff by simp
  with  $rs(1,3)$  show  $x - y \in \text{RSpan } [m]$ 
  using R-scalars.diff-closed[of r s] RSpan-single[of m] by auto
qed

lemma Group-RSpan :  $\text{set } ms \subseteq M \implies \text{Group } (\text{RSpan } ms)$ 
proof (induct ms)
  case Nil show ?case using trivial-Group by simp
next
  case (Cons m ms)
  hence  $\text{Group } (\text{RSpan } [m]) \ \text{Group } (\text{RSpan } ms)$ 
  using Group-RSpan-single[of m] by auto
  thus ?case
  using RSpan-Cons[of m ms] AbGroup.intro AbGroup-set-plus AbGroup.axioms(1)
  by fastforce
qed

lemma RSpanD-lincomb-arb-len-coeffs :
   $\text{set } ms \subseteq M \implies \text{RSpan } ms = \{ rs \cdot ms \mid rs. \text{set } rs \subseteq R \}$ 
proof
  show  $\text{RSpan } ms \subseteq \{ rs \cdot ms \mid rs. \text{set } rs \subseteq R \}$  using RSpanD-lincomb by fast
  show  $\text{set } ms \subseteq M \implies \text{RSpan } ms \supseteq \{ rs \cdot ms \mid rs. \text{set } rs \subseteq R \}$ 
  proof (induct ms)
    case Nil show ?case using lincomb-Nil by auto
  next
    case (Cons m ms) show ?case
    proof
      fix  $x$  assume  $x \in \{ rs \cdot (m\#ms) \mid rs. \text{set } rs \subseteq R \}$ 
      from this obtain  $rs$  where  $rs$ :  $\text{set } rs \subseteq R \ x = rs \cdot (m\#ms)$  by fast
      with Cons show  $x \in \text{RSpan } (m\#ms)$ 
      using lincomb-Nil Group-RSpan[of m\#ms] Group.zero-closed lincomb-Cons
      by (cases  $rs$ ) auto
    end
  end
end

```

qed
 qed
 qed

lemma *RSpanI-lincomb-arb-len-coeffs* :
 $set\ rs \subseteq R \implies set\ ms \subseteq M \implies rs \cdot ms \in RSpan\ ms$
using *RSpanD-lincomb-arb-len-coeffs* **by** *fast*

lemma *RSpan-contains-RSpans-Cons-left* :
 $set\ ms \subseteq M \implies RSpan\ [m] \subseteq RSpan\ (m\#\ms)$
using *RSpan-Cons Group-RSpan AbGroup.intro AbGroup.subset-plus-left* **by** *fast*

lemma *RSpan-contains-RSpans-Cons-right* :
 $m \in M \implies RSpan\ ms \subseteq RSpan\ (m\#\ms)$
using *RSpan-Cons Group-RSpan-single AbGroup.intro AbGroup.subset-plus-right*
by *fast*

lemma *RSpan-contains-RSpans-append-left* :
 $set\ ns \subseteq M \implies RSpan\ ms \subseteq RSpan\ (ms@ns)$
using *RSpan-append Group-RSpan AbGroup.intro AbGroup.subset-plus-left*
by *fast*

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

proof (*induct ms*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons m ms*) **thus** *?case*
using *RSpan-contains-spanset-single*
 $RSpan\ contains\ RSpans\ Cons\ left\ [of\ ms\ m]$
 $RSpan\ contains\ RSpans\ Cons\ right\ [of\ m\ ms]$
by *auto*

qed

lemma *RSpan-contains-spanset-append-left* :
 $set\ ms \subseteq M \implies set\ ns \subseteq M \implies set\ ms \subseteq RSpan\ (ms@ns)$
using *RSpan-contains-spanset[of ms@ns]* **by** *simp*

lemma *RSpan-contains-spanset-append-right* :
 $set\ ms \subseteq M \implies set\ ns \subseteq M \implies set\ ns \subseteq RSpan\ (ms@ns)$
using *RSpan-contains-spanset[of ms@ns]* **by** *simp*

lemma *RSpan-zero-closed* : $set\ ms \subseteq M \implies 0 \in RSpan\ ms$
using *Group-RSpan Group.zero-closed* **by** *fast*

lemma *RSpan-single-closed* : $m \in M \implies RSpan\ [m] \subseteq M$
using *RSpan-single smult-closed* **by** *auto*

lemma *RSpan-closed* : $set\ ms \subseteq M \implies RSpan\ ms \subseteq M$
proof (*induct ms*)

```

  case Nil show ?case using zero-closed by simp
next
  case (Cons m ms) thus ?case
    using RSpan-single-closed RSpan-Cons Group Group.set-plus-closed[of M]
    by simp
qed

```

```

lemma RSpan-smult-closed :
  assumes r ∈ R set ms ⊆ M n ∈ RSpan ms
  shows r · n ∈ RSpan ms
proof -
  from assms(2,3) obtain rs where rs: set rs ⊆ R n = rs · ms
    using RSpanD-lincomb-arb-len-coeffs by fast
  with assms(1,2) show ?thesis
    using smult-lincomb[OF rs(1) assms(1,2)] mult-closed
      RSpanI-lincomb-arb-len-coeffs[of [r*a. a←rs] ms]
    by auto
qed

```

```

lemma RSpan-add-closed :
  assumes set ms ⊆ M n ∈ RSpan ms n' ∈ RSpan ms
  shows n + n' ∈ RSpan ms
proof -
  from assms (2,3) obtain rs ss
    where rs: set rs ⊆ R length rs = length ms n = rs · ms
      and ss: set ss ⊆ R length ss = length ms n' = ss · ms
    using RSpanD-lincomb by auto
  with assms(1) have n + n' = [r + s. (r,s)←zip rs ss] · ms
    using lincomb-sum-left by simp
  moreover from rs(1) ss(1) have set [r + s. (r,s)←zip rs ss] ⊆ R
    using set-zip-leftD[of - - rs ss] set-zip-rightD[of - - rs ss]
      R-scalars.add-closed R-scalars.zip-add-closed by blast
  ultimately show n + n' ∈ RSpan ms
    using assms(1) RSpanI-lincomb-arb-len-coeffs by simp
qed

```

```

lemma RSpan-lincomb-closed :
  [ set rs ⊆ R; set ms ⊆ M; set ns ⊆ RSpan ms ] ⇒ rs · ns ∈ RSpan ms
using lincomb-Nil RSpan-zero-closed lincomb-Cons RSpan-smult-closed RSpan-add-closed
by (induct rs ns rule: list-induct2') auto

```

```

lemma RSpanI : set ms ⊆ M ⇒ M ⊆ RSpan ms ⇒ M = RSpan ms
using RSpan-closed by fast

```

```

lemma RSpan-contains-RSpan-take :
  set ms ⊆ M ⇒ RSpan (take k ms) ⊆ RSpan ms
using append-take-drop-id set-drop-subset
  RSpan-contains-RSpans-append-left[of drop k ms]
by fastforce

```

```

lemma RSubmodule-RSpan-single :
  assumes  $m \in M$ 
  shows RSubmodule (RSpan [m])
proof (rule RSubmoduleI)
  from assms show Subgroup (RSpan [m])
    using Group-RSpan-single RSpan-closed[of [m]] by simp
next
  fix  $r\ n$  assume  $rn: r \in R\ n \in RSpan\ [m]$ 
  from  $rn(2)$  obtain  $s$  where  $s \in R\ n = s \cdot m$  using RSpan-single by fast
  with assms  $rn(1)$  have  $r * s \in R\ r \cdot n = (r * s) \cdot m$ 
    using mult-closed by auto
  thus  $r \cdot n \in RSpan\ [m]$  using RSpan-single by fast
qed

lemma RSubmodule-RSpan :  $set\ ms \subseteq M \implies RSubmodule\ (RSpan\ ms)$ 
proof (induct ms)
  case Nil show ?case using trivial-RSubmodule by simp
next
  case (Cons m ms)
  hence RSubmodule (RSpan [m]) RSubmodule (RSpan ms)
    using RSubmodule-RSpan-single by auto
  thus ?case using RSpan-Cons RSubmodule-set-plus by simp
qed

lemma RSpan-RSpan-closed :
   $set\ ms \subseteq M \implies set\ ns \subseteq RSpan\ ms \implies RSpan\ ns \subseteq RSpan\ ms$ 
  using RSpanD-lincomb[of ns] RSpan-lincomb-closed by auto

lemma spanset-reduce-Cons :
   $set\ ms \subseteq M \implies m \in RSpan\ ms \implies RSpan\ (m\#\ms) = RSpan\ ms$ 
  using RSpan-Cons RSpan-RSpan-closed[of ms [m]]
    RSpan-contains-RSpans-Cons-right[of m ms]
    RSubmodule-RSpan[of ms]
    RModule.set-plus-closed[of R smult RSpan ms RSpan [m] RSpan ms]
  by auto

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
  fix  $x$  assume  $x \in RSpan\ (m\#\ms)$ 
  from this assms(3) obtain  $r\ rs\ s\ ss$ 
    where  $r\text{-rs}: r \in R\ set\ rs \subseteq R\ length\ rs = length\ ms\ x = r \cdot m + rs \cdot ms$ 
    and  $s\text{-ss}: s \in R\ set\ ss \subseteq R\ length\ ss = length\ ms\ m = s \cdot n + ss \cdot ms$ 
  using in-RSpan-Cons-obtain-same-length-coeffs[of x m ms]
    in-RSpan-Cons-obtain-same-length-coeffs[of m n ms]
  by fastforce
  from  $r\text{-rs}(1)\ s\text{-ss}(2)$  have  $set1: set\ [r*a.\ a\leftarrow ss] \subseteq R$  using mult-closed by auto

```



```

have x = ((r * s) # [a + b. (a,b)←zip [r*a. a←ss] rs]) · (n # ms)
proof-
  from r-rs(2,3) s-ss(3) assms(2)
  have [r*a. a←ss] · ms + rs · ms
    = [a + b. (a,b)←zip [r*a. a←ss] rs] · ms
  using set1 lincomb-sum
  by simp
moreover from assms(1,2) r-rs(1,2,4) s-ss(1,2,4)
  have x = (r * s) · n + ([r*a. a←ss] · ms + rs · ms)
  using smult-closed lincomb-closed smult-lincomb mult-closed lincomb-sum
  by simp
ultimately show ?thesis using lincomb-Cons by simp
qed
moreover have set ((r * s) # [a + b. (a,b)←zip [r*a. a←ss] rs]) ⊆ R
proof-
  from r-rs(2) have set [a + b. (a,b)←zip [r*a. a←ss] rs] ⊆ R
  using set1 R-scalars.zip-add-closed by fast
  with r-rs(1) s-ss(1) show ?thesis using mult-closed by simp
qed
ultimately show x ∈ RSpan (n # ms)
  using assms(1,2) RSpanI-lincomb-arb-len-coeffs[of - n#ms] by fastforce
qed
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 $R\text{-fingen } M \equiv (\exists ms. \text{ set } ms \subseteq M \wedge R\text{Span } ms = M)$

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

definition $dim\text{-}R :: 'm \text{ set} \Rightarrow nat$

where $dim\text{-}R M = (\text{if } R\text{-fingen } M \text{ then } ($
 $LEAST n. \exists ms. \text{ length } ms = n \wedge \text{ set } ms \subseteq M \wedge R\text{Span } ms = M$
 $) \text{ else } 0)$

lemma $dim\text{-}R\text{-nonzero} :$

assumes $dim\text{-}R M > 0$

shows $M \neq 0$

proof

assume $M: M = 0$

hence $dim\text{-}R M$

$= (LEAST n. \exists ms. \text{ length } ms = n \wedge \text{ set } ms \subseteq M \wedge R\text{Span } ms = M)$

using $dim\text{-}R\text{-def}$ **by** $simp$

moreover from M **have** $\text{length } [] = 0 \wedge \text{ set } [] \subseteq M \wedge R\text{Span } [] = M$ **by** $simp$

ultimately show $False$ **using** $assms$ **by** $simp$

qed

end

hide-const $real\text{-vector}.dim$

hide-const (**open**) $Real\text{-Vector-Spaces}.dim$

abbreviation (**in** $scalar\text{-mult}$) $fingen \equiv R\text{-scalar-mult}.R\text{-fingen } UNIV \text{ smult}$

abbreviation (**in** $scalar\text{-mult}$) $dim \equiv R\text{-scalar-mult}.dim\text{-}R \text{ UNIV } \text{ smult}$

lemmas (**in** $Module$) $dim\text{-nonzero} = dim\text{-}R\text{-nonzero}$

3.2.4 R -linear independence

context $R\text{-scalar-mult}$

begin

primrec $R\text{-lin-independent} :: 'm \text{ list} \Rightarrow bool$ **where**

$R\text{-lin-independent-Nil}: R\text{-lin-independent } [] = True$ |

$R\text{-lin-independent-Cons}:$

$R\text{-lin-independent } (m\#ms) = (R\text{-lin-independent } ms$

$\wedge (\forall r \text{ rs}. (\text{set } (r\#\text{rs}) \subseteq R \wedge (r\#\text{rs}) \cdot (m\#ms) = 0) \longrightarrow r = 0))$

lemma $R\text{-lin-independent-ConsI} :$

assumes $R\text{-lin-independent } ms$

$\bigwedge r \text{ rs}. \text{ set } (r\#\text{rs}) \subseteq R \implies (r\#\text{rs}) \cdot (m\#ms) = 0 \implies r = 0$

```

shows R-lin-independent (m#ms)
using assms R-lin-independent-Cons
by fast

lemma R-lin-independent-ConsD1 :
  R-lin-independent (m#ms)  $\implies$  R-lin-independent ms
by simp

lemma R-lin-independent-ConsD2 :
   $\llbracket$  R-lin-independent (m#ms); set (r#rs)  $\subseteq$  R; (r#rs)  $\cdot$  (m#ms) = 0  $\rrbracket$ 
 $\implies$  r = 0
by auto

end

context RModule
begin

lemma R-lin-independent-imp-same-scalars :
   $\llbracket$  length rs = length ss; length rs  $\leq$  length ms; set rs  $\subseteq$  R; set ss  $\subseteq$  R;
  set ms  $\subseteq$  M; R-lin-independent ms; rs  $\cdot$  ms = ss  $\cdot$  ms  $\rrbracket \implies$  rs = ss
proof (induct rs ss arbitrary: ms rule: list-induct2)
  case (Cons r rs s ss)
  from Cons(3) have ms  $\neq$  [] by auto
  from this obtain n ns where ms: ms = n#ns
  using neq-Nil-conv[of ms] by fast
  from Cons(4,5) have set ([a-b. (a,b) $\leftarrow$ zip (r#rs) (s#ss)])  $\subseteq$  R
  using Ring1 Ring1.list-diff-closed by fast
  hence set ((r-s)#[a-b. (a,b) $\leftarrow$ zip rs ss])  $\subseteq$  R by simp
  moreover from Cons(1,4-6,8) ms
  have 1: ((r-s)#[a-b. (a,b) $\leftarrow$ zip rs ss])  $\cdot$  (n#ns) = 0
  using lincomb-diff-left[of r#rs s#ss]
  by simp
  ultimately have r - s = 0 using Cons(7) ms R-lin-independent-Cons by fast
  hence 2: r = s by simp
  with 1 Cons(1,4-6) ms have rs  $\cdot$  ns = ss  $\cdot$  ns
  using lincomb-Cons zero-smult lincomb-diff-left by simp
  with Cons(2-7) ms have rs = ss by simp
  with 2 show ?case by fast
qed fast

lemma R-lin-independent-obtain-unique-scalars :
   $\llbracket$  set ms  $\subseteq$  M; R-lin-independent ms; n  $\in$  RSpan ms  $\rrbracket$ 
 $\implies$  ( $\exists!$  rs. set rs  $\subseteq$  R  $\wedge$  length rs = length ms  $\wedge$  n = rs  $\cdot$  ms)
using in-RSpan-obtain-same-length-coeffs[of n ms]
R-lin-independent-imp-same-scalars[of - - ms]
by auto

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 (*induct ms*)
case (*Cons m ms*) **show** *?case*
proof (*rule R-lin-independent-ConsI*)
have $\bigwedge rs. \llbracket set\ rs \subseteq R; length\ rs = length\ ms; rs \cdot ms = 0 \rrbracket \implies set\ rs \subseteq 0$
proof–
fix *rs* **assume** *rs: set rs ⊆ R length rs = length ms rs · ms = 0*
with *Cons(2)* **have** $set\ (0\#rs) \subseteq R\ length\ (0\#rs)$
 $= length\ (m\#ms)\ (0\#rs) \cdot (m\#ms) = 0$
using *R-scalars.zero-closed lincomb-Cons zero-smult* **by** *auto*
with *Cons(3)* **have** $set\ (0\#rs) \subseteq 0$ **by** *fast*
thus $set\ rs \subseteq 0$ **by** *simp*
qed
with *Cons(1,2)* **show** *R-lin-independent ms by simp*
next
fix *r rs* **assume** *r-rs: set (r # rs) ⊆ R (r # rs) · (m # ms) = 0*
from *r-rs(1) Cons(2)* **obtain** *ss*
where *ss: set ss ⊆ R length ss = length ms rs · ms = ss · ms*
using *lincomb-obtain-same-length-Rcoeffs[of rs ms]*
by *auto*
with *r-rs* **have** $(r\#ss) \cdot (m\#ms) = 0$ **using** *lincomb-Cons* **by** *simp*
moreover **from** *r-rs(1) ss(1)* **have** $set\ (r\#ss) \subseteq R$ **by** *simp*
moreover **from** *ss(2)* **have** $length\ (r\#ss) = length\ (m\#ms)$ **by** *simp*
ultimately **have** $set\ (r\#ss) \subseteq 0$ **using** *Cons(3)* **by** *fast*
thus $r = 0$ **by** *simp*
qed
qed *simp*

lemma *R-lin-independentI-concat-all-scalars* :
defines *eq-len: eq-len ≡ λxs 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-lin-independent (concat mss)*
using *assms(2)*
proof (*rule R-lin-independentI-all-scalars*)
have $\bigwedge rs. \llbracket set\ rs \subseteq R; length\ rs = length\ (concat\ mss); rs \cdot concat\ mss = 0 \rrbracket$
 $\implies set\ rs \subseteq 0$
proof–
fix *rs*
assume *rs: set rs ⊆ R length rs = length (concat mss) rs · concat mss = 0*
from *rs(2) eq-len* **obtain** *rss* **where** $rs = concat\ rss\ list\text{-all2}\ eq\text{-len}\ rss\ mss$
using *match-concat* **by** *fast*
with *rs(1,3) assms(3)* **show** $set\ rs \subseteq 0$ **by** *auto*
qed
thus $\forall rs. set\ rs \subseteq R \wedge length\ rs = length\ (concat\ mss) \wedge rs \cdot concat\ mss = 0$
 $\longrightarrow set\ rs \subseteq 0$

by *auto*
qed

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

$\llbracket \text{set } rs \subseteq R; \text{set } ms \subseteq M; \text{length } rs \leq \text{length } ms; R\text{-lin-independent } ms; rs \cdot ms = 0 \rrbracket \implies \text{set } rs \subseteq 0$

proof (*induct rs ms rule: list-induct2'*)

case ($\lambda r rs m ms$)

from $\lambda(2,5,6)$ have $r = 0$ by *auto*

moreover with λ have $\text{set } rs \subseteq 0$ using *lincomb-Cons zero-smult* by *simp*
ultimately show *?case* by *simp*

qed *auto*

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

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

shows $rs!k = 0$

proof –

from *assms(1,2)* obtain *ss*

where *ss*: $\text{set } ss \subseteq R \text{ length } ss = \text{length } ms$

$\text{take } (\text{length } rs) \text{ } ss = \text{take } (\text{length } ms) \text{ } rs \cdot ms = ss \cdot ms$

using *lincomb-obtain-same-length-Rcoeffs[of rs ms]*

by *fast*

from *ss(1,2,4)* *assms(2,3,4)* have $\text{set } ss \subseteq 0$

using *R-lin-independentD-all-scalars* by *auto*

moreover from *assms(5)* *ss(3)* have $rs!k = (\text{take } (\text{length } rs) \text{ } ss)!k$ by *simp*

moreover from *assms(5)* *ss(2)* have $k < \text{length } (\text{take } (\text{length } rs) \text{ } ss)$ by *simp*

ultimately show *?thesis* using *in-set-conv-nth* by *force*

qed

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

$\text{set } ms \subseteq M \implies \neg R\text{-lin-independent } ms$

$\implies \exists rs. \text{set } rs \subseteq R \wedge \text{set } rs \neq 0 \wedge \text{length } rs = \text{length } ms \wedge rs \cdot ms = 0$

proof (*induct ms*)

case (*Cons m ms*) show *?case*

proof (*cases R-lin-independent ms*)

case *True*

with *Cons(3)*

have $\neg (\forall r rs. (\text{set } (r\#rs) \subseteq R \wedge (r\#rs) \cdot (m\#ms) = 0) \longrightarrow r = 0)$

by *simp*

from *this* obtain *r rs*

where *r-rs*: $\text{set } (r\#rs) \subseteq R \text{ } (r\#rs) \cdot (m\#ms) = 0 \text{ } r \neq 0$

by *fast*

from *r-rs(1)* *Cons(2)* obtain *ss*

where *ss*: $\text{set } ss \subseteq R \text{ length } ss = \text{length } ms \text{ } rs \cdot ms = ss \cdot ms$

using *lincomb-obtain-same-length-Rcoeffs[of rs ms]*

by *auto*

from *ss r-rs* have $\text{set } (r\#ss) \subseteq R \wedge \text{set } (r\#ss) \neq 0$

$\wedge \text{length } (r\#ss) = \text{length } (m\#ms) \wedge (r\#ss) \cdot (m\#ms) = 0$

```

    using lincomb-Cons
    by simp
  thus ?thesis by fast
next
case False
with Cons(1,2) obtain rs
  where rs: set rs  $\subseteq$  R set rs  $\neq$  0 length rs = length ms rs  $\cdot$  ms = 0
  by fastforce
from False rs Cons(2)
  have set (0#rs)  $\subseteq$  R  $\wedge$  set (0#rs)  $\neq$  0  $\wedge$  length (0#rs) = length (m#ms)
     $\wedge$  (0#rs)  $\cdot$  (m#ms) = 0
  using Ring1.zero-closed[OF Ring1] lincomb-Cons[of 0 rs m ms]
    zero-smult[of m] empty-set-diff-single[of set rs]
  by fastforce
  thus ?thesis by fast
qed
qed simp

```

```

lemma R-lin-independent-imp-distinct :
  set ms  $\subseteq$  M  $\implies$  R-lin-independent ms  $\implies$  distinct ms
proof (induct ms)
case (Cons m ms)
  have  $\bigwedge n. n \in \text{set } ms \implies m \neq n$ 
  proof
    fix n assume n: n  $\in$  set ms m = n
    from n(1) obtain xs ys where ms = xs @ n # ys using split-list by fast
    with Cons(2) n(2)
      have (1 # replicate (length xs) 0 @ [-1])  $\cdot$  (m # ms) = 0
      using lincomb-Cons lincomb-append lincomb-replicate0-left lincomb-Nil neg-eq-neg1-smult
      by simp
    with Cons(3) have 1 = 0
      using R-scalars.zero-closed one-closed R-scalars.neg-closed by force
    thus False using one-neq-zero by fast
  qed
  with Cons show ?case by auto
qed simp

```

```

lemma R-lin-independent-imp-independent-take :
  set ms  $\subseteq$  M  $\implies$  R-lin-independent ms  $\implies$  R-lin-independent (take n ms)
proof (induct ms arbitrary: n)
case (Cons m ms) show ?case
  proof (cases n)
  case (Suc k)
    hence take n (m#ms) = m # take k ms by simp
    moreover have R-lin-independent (m # take k ms)
    proof (rule R-lin-independent-ConsI)
    from Cons show R-lin-independent (take k ms) by simp
    next
    fix r rs assume r-rs: set (r#rs)  $\subseteq$  R (r#rs)  $\cdot$  (m # take k ms) = 0

```

from $r\text{-rs}(1)$ $\text{Cons}(2)$ **obtain** ss
where ss : $\text{set } ss \subseteq R$ $\text{length } ss = \text{length } (\text{take } k \text{ } ms)$
 $rs \cdot \text{take } k \text{ } ms = ss \cdot \text{take } k \text{ } ms$
using $\text{set-take-subset}[\text{of } k \text{ } ms]$ $\text{lincomb-obtain-same-length-Rcoeffs}$
by force
from $r\text{-rs}(1)$ $ss(1)$ **have** $\text{set } (r\#ss) \subseteq R$ **by** simp
moreover from $r\text{-rs}(2)$ ss **have** $(r\#ss) \cdot (m\#ms) = 0$
using lincomb-Cons lincomb-Nil
 $\text{lincomb-append-right}[\text{of } ss \text{ take } k \text{ } ms \text{ drop } k \text{ } ms]$
by simp
ultimately show $r = 0$ **using** $\text{Cons}(3)$ **by** auto
qed
ultimately show $?thesis$ **by** simp
qed simp
qed simp

lemma $R\text{-lin-independent-Cons-imp-independent-RSpans}$:
assumes $m \in M$ $R\text{-lin-independent } (m\#ms)$
shows $\text{add-independentS } [R\text{Span } [m], R\text{Span } ms]$
proof ($\text{rule add-independentS-doubleI}$)
fix $x \ y$ **assume** xy : $x \in R\text{Span } [m]$ $y \in R\text{Span } ms$ $x + y = 0$
from $xy(1,2)$ **obtain** $r \ rs$ **where** $r\text{-rs}$: $r \in R$ $x = r \cdot m$ $\text{set } rs \subseteq R$ $y = rs \cdot ms$
using $R\text{Span-single}$ $R\text{SpanD-lincomb}$ **by** fast
with $xy(3)$ **have** $\text{set } (r\#rs) \subseteq R$ $(r\#rs) \cdot (m\#ms) = 0$
using lincomb-Cons **by** auto
with $\text{assms } r\text{-rs}(2)$ **show** $x = 0$ **using** zero-smult **by** auto
qed

lemma $hd0\text{-imp-}R\text{-lin-dependent}$: $\neg R\text{-lin-independent } (0\#ms)$
using $\text{lincomb-Cons}[\text{of } 1 \ [] \ 0 \ ms]$ $\text{lincomb-Nil}[\text{of } [] \ ms]$ $\text{smult-zero one-closed}$
 $R\text{-lin-independent-Cons}$
by fastforce

lemma $R\text{-lin-independent-imp-hd-n0}$: $R\text{-lin-independent } (m\#ms) \implies m \neq 0$
using $hd0\text{-imp-}R\text{-lin-dependent}$ **by** fast

lemma $R\text{-lin-independent-imp-hd-independent-from-RSpan}$:
assumes $m \in M$ $\text{set } ms \subseteq M$ $R\text{-lin-independent } (m\#ms)$
shows $m \notin R\text{Span } ms$
proof
assume m : $m \in R\text{Span } ms$
with $\text{assms}(2)$ **have** $(-1) \cdot m \in R\text{Span } ms$
using $R\text{Submodule-RSpan}[\text{of } ms]$
 $R\text{Module.smult-closed}[\text{of } R \ \text{smult } R\text{Span } ms \ -1 \ m]$
 $\text{one-closed } R\text{-scalars.neg-closed}[\text{of } 1]$
by simp
moreover from $\text{assms}(1)$ **have** $m + (-1) \cdot m = 0$
using neg-eq-neg1-smult **by** simp
ultimately show False

using *RSpan-contains-spanset-single* *assms* *R-lin-independent-Cons-imp-independent-RSpans*
add-independentS-doubleD *R-lin-independent-imp-hd-n0*
by *fast*
qed

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 (*induct* *ms*)

case (*Cons* *m* *ms*)

moreover **have** $\bigwedge r\ rs.\ set\ (r\ \#\ rs) \subseteq R \implies (r\ \#\ rs) \cdot (m\ \#\ ms\ @\ ns) = 0$
 $\implies r = 0$

proof–

fix *r* *rs* **assume** *r-rs*: $set\ (r\ \#\ rs) \subseteq R\ (r\ \#\ rs) \cdot (m\ \#\ ms\ @\ ns) = 0$

from *Cons*(2) *r-rs*(1) **obtain** *ss*

where *ss*: $set\ ss \subseteq R\ length\ ss = length\ ms\ rs \cdot ms = ss \cdot ms$

using *lincomb-obtain-same-length-Rcoeffs*[*of* *rs* *ms*]

by *auto*

from *assms* *ss*(2,3) *r-rs*(2)

have $(r\ \#\ ss\ @\ 0\ \#\ drop\ (length\ ms)\ rs) \cdot (m\ \#\ ms\ @\ n\ \#\ ns) = 0$

using *lincomb-Cons*

lincomb-append-right *add.assoc*[*of* *r*·*m* *rs*·*ms* (*drop* (*length* *ms*) *rs*)·*ns*]
zero-smult *lincomb-append*

by *simp*

moreover **from** *r-rs*(1) *ss*(1)

have $set\ (r\ \#\ ss\ @\ 0\ \#\ drop\ (length\ ms)\ rs) \subseteq R$

using *R-scalars.zero-closed* *set-drop-subset*[*of* - *rs*]

by *auto*

ultimately **show** $r = 0$

using *Cons*(3)

R-lin-independent-ConsD2[*of* *m* - *r* *ss* @ 0 # *drop* (*length* *ms*) *rs*]

by *simp*

qed

ultimately **show** $R\text{-lin-independent}\ ((m\ \#\ ms)\ @\ ns)$ **by** *auto*

qed *simp*

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–

define *k* **where** $k = length\ rs$

hence $(rs\ @\ s\ \#\ ss)!k = s$ **by** *simp*

moreover **from** *k-def* *assms*(4) **have** $k < \min\ (length\ (rs\ @\ s\ \#\ ss))\ (length\ (ms\ @\ n\ \#\ ns))$

by *simp*

ultimately **show** *?thesis*

using *assms*(1,2,3,5) *R-lin-independentD-all-scalars-nth*[*of* *rs*@*s*#*ss* *ms*@*n*#*ns*]

by simp
qed

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

set $ms \subseteq M \implies R\text{-lin-independent } (ms@ns)$
 $\implies \text{add-independentS } [R\text{Span } ms, R\text{Span } ns]$

proof (induct ms)

case (Cons m ms)

show ?case

proof (rule add-independentS-doubleI)

fix x y assume xy: $y \in R\text{Span } ns$ $x \in R\text{Span } (m\#ms)$ $x + y = 0$

from xy(2) obtain x1 x2

where x1-x2: $x1 \in R\text{Span } [m]$ $x2 \in R\text{Span } ms$ $x = x1 + x2$

using RSpan-Cons set-plus-def[of RSpan [m]]

by auto

from x1-x2(1,2) xy(1) obtain r rs ss

where r-rs-ss: set $(r\#(rs@ss)) \subseteq R$ $\text{length } rs = \text{length } ms$ $x1 = r \cdot m$
 $x2 = rs \cdot ms$ $y = ss \cdot ns$

using RSpan-single in-RSpan-obtain-same-length-coeffs[of x2 ms]

RSpanD-lincomb[of ns]

by auto

have x1-0: $x1 = 0$

proof–

from xy(3) x1-x2(3) r-rs-ss(2–5) have $(r\#(rs@ss)) \cdot (m\#(ms@ns)) = 0$

using lincomb-append lincomb-Cons by (simp add: algebra-simps)

with r-rs-ss(1,3) Cons(2,3) show ?thesis

using R-lin-independent-ConsD2[of m ms@ns r rs@ss] zero-smult by simp

qed

moreover have x2 = 0

proof–

from x1-0 xy(3) x1-x2(3) have $x2 + y = 0$ by simp

with xy(1) x1-x2(2) Cons show ?thesis

using add-independentS-doubleD by simp

qed

ultimately show $x = 0$ using x1-x2(3) by simp

qed

qed simp

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-scalar-mult.R-lin-independent-ConsI [OF R-scalar-mult, of smult]
lemmas lin-independent-ConsD1
      = R-scalar-mult.R-lin-independent-ConsD1 [OF R-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 = ( $\exists n. \forall ms. \text{set } ms \subseteq M$ 
     $\wedge$  R-lin-independent ms  $\longrightarrow$  length ms  $\leq$  n)

lemma R-finrankI :
  ( $\bigwedge ms. \text{set } ms \subseteq M \Longrightarrow$  R-lin-independent ms  $\Longrightarrow$  length ms  $\leq$  n)
   $\Longrightarrow$  R-finrank M
  unfolding R-finrank-def by blast

lemma R-finrankD :
  R-finrank M  $\Longrightarrow$   $\exists n. \forall ms. \text{set } ms \subseteq M \wedge$  R-lin-independent ms
   $\longrightarrow$  length ms  $\leq$  n
  unfolding R-finrank-def by fast

lemma submodule-R-finrank : R-finrank M  $\Longrightarrow$  N  $\subseteq$  M  $\Longrightarrow$  R-finrank N
  unfolding R-finrank-def by blast

end

context scalar-mult
begin

```

```

abbreviation finrank :: 'm set  $\Rightarrow$  bool
  where finrank  $\equiv$  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  $\Rightarrow$  'm::ab-group-add  $\Rightarrow$  'm (infixr  $\cdot$  70)
  and M      :: 'm set
  and smult' :: 'r  $\Rightarrow$  'n::ab-group-add  $\Rightarrow$  'n (infixr  $\star$  70)
  and T      :: 'm  $\Rightarrow$  'n
+ assumes R-map:  $\bigwedge r m. r \in R \Rightarrow m \in M \Rightarrow T (r \cdot m) = r \star T m$ 

abbreviation (in RModuleHom) lincomb' :: 'r list  $\Rightarrow$  'n list  $\Rightarrow$  'n (infix  $\star$  70)
  where lincomb'  $\equiv$  Codomain.lincomb

```

```

lemma (in RModule) RModuleHomI :
  assumes GroupHom M T
   $\bigwedge r m. r \in R \Rightarrow m \in M \Rightarrow T (r \cdot m) = smult' r (T m)$ 
  shows RModuleHom R smult M smult' T
  by (
    rule RModuleHom.intro, rule RModule-axioms, rule assms(1), un-
    fold-locales,
    rule assms(2)
  )

```

```

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

```

```

locale ModuleHom = RModuleHom UNIV smult M smult' T
  for smult  :: 'r::ring-1  $\Rightarrow$  'm::ab-group-add  $\Rightarrow$  'm (infixr  $\cdot$  70)
  and M      :: 'm set
  and smult' :: 'r  $\Rightarrow$  'n::ab-group-add  $\Rightarrow$  'n (infixr  $\star$  70)
  and T      :: 'm  $\Rightarrow$  '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 ⊆ M

locale RModuleIso = 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 smult' :: 'r ⇒ 'n::ab-group-add ⇒ 'n (infixr ★ 70)
  and T    :: 'm ⇒ '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 (rule RModuleIso.intro)
  from assms show RModuleHom R (·) M smult' T
    using GroupIso.axioms(1) RModuleHomI by fastforce
  from assms(1) show RModuleIso-axioms M T N
    using GroupIso.bijjective by unfold-locales
qed

3.3.2 Basic facts

lemma (in RModule) trivial-RModuleHom :
   $\forall r \in R. smult' r 0 = 0 \implies RModuleHom R smult M smult' 0$ 
  using trivial-GroupHom RModuleHomI by fastforce

lemma (in RModule) RModHom-idhom : RModuleHom R smult M smult (id ↓ M)
  using RModule-axioms GroupHom-idhom
proof (rule RModuleHom.intro)
  show RModuleHom-axioms R (·) M (·) (id ↓ M)
    using smult-closed by unfold-locales simp
qed

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 ..*

lemma *codomain-smult-zero* : $r \in R \implies r \star 0 = 0$
using *im-zero smult-zero zero-closed R-map[of r 0]* **by** *simp*

lemma *RSubmodule-Ker* : *Domain.RSubmodule Ker*
proof (*rule Domain.RSubmoduleI, rule conjI, rule Group-Ker*)
fix $r\ m$ **assume** $r: r \in R$ **and** $m: m \in Ker$
thus $r \cdot m \in Ker$
using *R-map[of r m] kerD[of m T] codomain-smult-zero kerI Domain.smult-closed*
by *simp*
qed *fast*

lemma *RModule-Im* : *RModule R smult' ImG*
using *Ring1 Group-Im*
proof (*rule RModuleI, unfold-locales*)
show $\bigwedge n. n \in T \text{ ' } M \implies 1 \star n = n$ **using** *one-closed R-map[of 1]* **by** *auto*
next
fix $r\ s\ m\ n$ **assume** $r: r \in R$ **and** $s: s \in R$ **and** $m: m \in T \text{ ' } M$
and $n: n \in T \text{ ' } M$
from $m\ n$ **obtain** $m'\ n'$
where $m': m' \in M\ m = T\ m'$ **and** $n': n' \in M\ n = T\ n'$
by *fast*
from $m'\ r$ *R-map* **have** $r \star m = T\ (r \cdot m')$ **by** *simp*
with $r\ m'(1)$ **show** $r \star m \in T \text{ ' } M$ **using** *smult-closed* **by** *fast*
from $r\ m'\ n'$ **show** $r \star (m + n) = r \star m + r \star n$
using *hom add-closed R-map[of r m'+n'] smult-closed R-map[of r]* **by** *simp*
from $r\ s\ m'$ **show** $(r + s) \star m = r \star m + s \star m$
using *R-scalars.add-closed R-map[of r+s] smult-closed hom R-map* **by** *simp*
from $r\ s\ m'$ **show** $r \star s \star m = (r \star s) \star m$
using *smult-closed R-map[of s] R-map[of r s \cdot m'] mult-closed R-map[of r*s]*
by *simp*
qed

lemma *im-submodule* :
assumes *RSubmodule N*
shows *RModule.RSubmodule R smult' ImG (T ' N)*
proof (*rule RModule.RSubmoduleI, rule RModule-Im*)
from *assms* **show** *Group.Subgroup (T ' M) (T ' N)*
using *im-subgroup Subgroup-RSubmodule* **by** *fast*
from *assms R-map* **show** $\bigwedge r\ n. r \in R \implies n \in T \text{ ' } N \implies r \star n \in T \text{ ' } N$
using *RModule.smult-closed* **by** *force*
qed

lemma *RModHom-composite-left* :
assumes $T \text{ ' } M \subseteq N$ *RModuleHom R smult' N smult'' S*

shows $RModuleHom\ R\ smult\ M\ smult''\ (S\ \circ\ T)$
proof (*rule* $RModule.RModuleHomI$, *rule* $RModule-axioms$)
from $assms(1)$ **show** $GroupHom\ M\ (S\ \circ\ T)$
using $RModuleHom.GroupHom[OF\ assms(2)]$ $GroupHom-composite-left$
by *auto*
from $assms(1)$
show $\bigwedge r\ m. r \in R \implies m \in M \implies (S\ \circ\ T)\ (r \cdot m) = smult''\ r\ ((S\ \circ\ T)\ m)$
using $R-map\ RModuleHom.R-map[OF\ assms(2)]$
by *auto*
qed

lemma $RModuleHom-restrict0-submodule$:
assumes $RSubmodule\ N$
shows $RModuleHom\ R\ smult\ N\ smult'\ (T\ \downarrow\ N)$
proof (*rule* $RModuleHom.intro$)
from $assms$ **show** $RModule\ R\ (\cdot)\ N$ **by** *fast*
from $assms$ **show** $GroupHom\ N\ (T\ \downarrow\ N)$
using $RModule.Group\ GroupHom-restrict0-subgroup$ **by** *fast*
show $RModuleHom-axioms\ R\ (\cdot)\ N\ (\star)\ (T\ \downarrow\ N)$
proof
fix $r\ m$ **assume** $r \in R\ m \in N$
with $assms$ **show** $(T\ \downarrow\ N)\ (r \cdot m) = r \star (T\ \downarrow\ N)\ m$
using $RModule.smult-closed\ R-map$ **by** *fastforce*
qed
qed

lemma $distrib-lincomb$:
 $set\ rs \subseteq R \implies set\ ms \subseteq M \implies T\ (rs\ \cdot\ ms) = rs\ \star\ map\ T\ ms$
using $Domain.lincomb-Nil\ im-zero\ Codomain.lincomb-Nil\ R-map\ Domain.lincomb-Cons$
 $Domain.smult-closed\ Domain.lincomb-closed\ additive\ Codomain.lincomb-Cons$
by (*induct* $rs\ ms$ *rule*: $list-induct2'$) *auto*

lemma $same-image-on-RSpanset-imp-same-hom$:
assumes $RModuleHom\ R\ smult\ M\ smult'\ S\ set\ ms \subseteq M$
 $M = Domain.R-scalars.RSpan\ ms\ \forall m \in set\ ms. S\ m = T\ m$
shows $S = T$
proof
fix m **show** $S\ m = T\ m$
proof (*cases* $m \in M$)
case *True*
with $assms(2,3)$ **obtain** rs **where** $rs: set\ rs \subseteq R\ m = rs\ \cdot\ ms$
using $Domain.RSpanD-lincomb-arb-len-coeffs$ **by** *fast*
from $rs(1)\ assms(2)$ **have** $S\ (rs\ \cdot\ ms) = rs\ \star\ (map\ S\ ms)$
using $RModuleHom.distrib-lincomb[OF\ assms(1)]$ **by** *simp*
moreover from $rs(1)\ assms(2)$ **have** $T\ (rs\ \cdot\ ms) = rs\ \star\ (map\ T\ ms)$
using $distrib-lincomb$ **by** *simp*
ultimately show *?thesis* **using** $assms(4)\ map-ext[of\ ms\ S\ T]\ rs(2)$ **by** *auto*
next
case *False* **with** $assms(1)$ *supp* **show** *?thesis*

```

    using RModuleHom.supp suppI-contr[of - S] suppI-contr[of - T] by fastforce
  qed
qed
end

```

```

lemma RSubmodule-eigenspace :
  fixes smult :: 'r::ring-1  $\Rightarrow$  'm::ab-group-add  $\Rightarrow$  'm (infixr  $\cdot$  70)
  assumes RModHom: RModuleHom R smult M smult T
  and r:  $r \in R \wedge s m. s \in R \implies m \in M \implies s \cdot r \cdot m = r \cdot s \cdot m$ 
  defines E:  $E \equiv \{m \in M. T m = r \cdot m\}$ 
  shows RModule.RSubmodule R smult M E
proof (rule RModule.RSubmoduleI)
  from RModHom show rmod: RModule R smult M
    using RModuleHom.axioms(1) by fast
  have Group E
  proof
    from r(1) E show  $E \neq \{\}$ 
      using RModule.zero-closed[OF rmod] RModuleHom.im-zero[OF RModHom]
        RModule.smult-zero[OF rmod]
      by auto
    next
      fix m n assume  $m \in E n \in E$ 
      with r(1) E show  $m - n \in E$ 
        using RModule.diff-closed[OF rmod] RModuleHom.im-diff[OF RModHom]
          RModule.smult-distrib-left-diff[OF rmod]
        by simp
    qed
  with E show Group.Subgroup M E by fast
  show  $\wedge s m. s \in R \implies m \in E \implies s \cdot m \in E$ 
  proof-
    fix s m assume  $s \in R m \in E$ 
    with E r RModuleHom.R-map[OF RModHom] show  $s \cdot m \in E$ 
      using RModule.smult-closed[OF rmod] by simp
  qed
qed

```

3.3.3 Basic facts about endomorphisms

```

lemma (in RModule) Rmap-endorph-is-RModuleEnd :
  assumes grpend: GroupEnd M T
  and Rmap :  $\wedge r m. r \in R \implies m \in M \implies T (r \cdot m) = r \cdot (T m)$ 
  shows RModuleEnd R smult M T
proof (rule RModuleEnd.intro, rule RModuleHomI)
  from grpend show GroupHom M T using GroupEnd.axioms(1) by fast
  from grpend show RModuleEnd-axioms M T
    using GroupEnd.endomorph by unfold-locales
  qed (rule Rmap)

```

```

lemma (in RModuleEnd) GroupEnd : GroupEnd M T
proof (rule GroupEnd.intro)
  from endomorph show GroupEnd-axioms M T by unfold-locales
qed (unfold-locales)

lemmas (in RModuleEnd) proj-decomp = GroupEnd.proj-decomp[OF GroupEnd]

lemma (in ModuleEnd) RModuleEnd : RModuleEnd UNIV smult M T
  using endomorph RModuleEnd.intro by unfold-locales

lemmas (in ModuleEnd) GroupEnd = RModuleEnd.GroupEnd[OF RModuleEnd]

lemma RModuleEnd-over-UNIV-is-ModuleEnd :
  RModuleEnd UNIV rsmult M T  $\implies$  ModuleEnd rsmult M T
proof (rule ModuleEnd.intro, rule ModuleHom.intro)
  assume endo: RModuleEnd UNIV rsmult M T
  thus RModuleHom UNIV rsmult M rsmult T
  using RModuleEnd.axioms(1) by fast
  from endo show ModuleEnd-axioms M T
  using RModuleEnd.endomorph by unfold-locales
qed

```

3.3.4 Basic facts about isomorphisms

```

context RModuleIso
begin

```

```

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

```

```

lemma GroupIso : GroupIso M T N
proof (rule GroupIso.intro)
  show GroupHom M T ..
  from bijective show GroupIso-axioms M T N by unfold-locales
qed

```

```

lemmas ImG = GroupIso.ImG [OF GroupIso]
lemmas GroupHom-inv = GroupIso.inv [OF GroupIso]
lemmas invT-into = GroupIso.invT-into [OF GroupIso]
lemmas T-invT = GroupIso.T-invT [OF GroupIso]
lemmas invT-eq = GroupIso.invT-eq [OF GroupIso]

```

```

lemma RModuleN : RModule R smult' N using RModule-Im ImG by fast

```

```

lemma inv : RModuleIso R smult' N smult invT M
  using RModuleN GroupHom-inv
proof (rule RModule.RModuleIsoI)
  fix r n assume rn: r  $\in$  R n  $\in$  N
  thus invT (r  $\star$  n) = r  $\cdot$  invT n
  using invT-into smult-closed R-map T-invT invT-eq by simp

```


qed

end

3.4 Inner direct sums of RModules

lemma (in *RModule*) *RModule-inner-dirsum-el-decomp-Rsmult* :
 assumes $\forall N \in \text{set } Ns. \text{RSubmodule } N \text{ add-independentS } Ns \ r \in R$
 $x \in (\bigoplus N \leftarrow Ns. N)$
 shows $(\bigoplus Ns \leftarrow (r \cdot x)) = [r \cdot m. m \leftarrow (\bigoplus Ns \leftarrow x)]$
proof –
 define *xs* **where** $xs = (\bigoplus Ns \leftarrow x)$
 with *assms* **have** $x: xs \in \text{listset } Ns \ x = \text{sum-list } xs$
 using *RModule.AbGroup*[of *R*] *AbGroup-inner-dirsum-el-decompI*[of *Ns* *x*]
 by *auto*
 from *assms*(1,2,4) *xs-def* **have** $xs \subseteq M$
 using *Subgroup-RSubmodule*
 AbGroup.abSubgroup-inner-dirsum-el-decomp-set[OF *AbGroup*]
 by *fast*
 from *assms*(1,3) *x*(1) **have** $[r \cdot m. m \leftarrow xs] \in \text{listset } Ns$
 using *listset-RModule-Rsmult-closed* **by** *fast*
 moreover from *x* *assms*(3) *xs-M* **have** $r \cdot x = \text{sum-list } [r \cdot m. m \leftarrow xs]$
 using *smult-sum-list-distrib* **by** *fast*
 moreover from *assms*(1,3,4) **have** $r \cdot x \in (\bigoplus M \leftarrow Ns. M)$
 using *RModule-inner-dirsum* *RModule.smult-closed* **by** *fast*
 ultimately show $(\bigoplus Ns \leftarrow (r \cdot x)) = [r \cdot m. m \leftarrow xs]$
 using *assms*(1,2) *RModule.AbGroup* *AbGroup-inner-dirsum-el-decomp-eq*
 by *fast*
qed

lemma (in *RModule*) *RModuleEnd-inner-dirsum-el-decomp-nth* :
 assumes $\forall N \in \text{set } Ns. \text{RSubmodule } N \text{ add-independentS } Ns \ n < \text{length } Ns$
 shows $\text{RModuleEnd } R \ \text{smult } (\bigoplus N \leftarrow Ns. N) \ (\bigoplus Ns \downarrow n)$
proof (*rule* *RModule.Rmap-endomorph-is-RModuleEnd*)
 from *assms*(1) **show** $\text{RModule } R \ \text{smult } (\bigoplus N \leftarrow Ns. N)$
 using *RSubmodule-inner-dirsum* **by** *fast*
 from *assms* **show** $\text{GroupEnd } (\bigoplus N \leftarrow Ns. N) \ (\bigoplus Ns \downarrow n)$
 using *RModule.AbGroup* *GroupEnd-inner-dirsum-el-decomp-nth*[of *Ns*] **by** *fast*
 show $\bigwedge r \ m. r \in R \implies m \in (\bigoplus N \leftarrow Ns. N)$
 $\implies (\bigoplus Ns \downarrow n) (r \cdot m) = r \cdot ((\bigoplus Ns \downarrow n) m)$
proof –
 fix *r* *m* **assume** $r \in R \ m \in (\bigoplus N \leftarrow Ns. N)$
 moreover with *assms*(1) **have** $r \cdot m \in (\bigoplus M \leftarrow Ns. M)$
 using *RModule-inner-dirsum* *RModule.smult-closed* **by** *fast*
 ultimately show $(\bigoplus Ns \downarrow n) (r \cdot m) = r \cdot (\bigoplus Ns \downarrow n) m$
 using *assms* *RModule.AbGroup*[of *R* *smult*]
 AbGroup-length-inner-dirsum-el-decomp[of *Ns*]
 RModule-inner-dirsum-el-decomp-Rsmult
 by *simp*

qed
qed

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-group-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- qed
```

```
locale FinDimVectorSpace = VectorSpace
+ assumes findim: findim V
```

```
lemma (in VectorSpace) FinDimVectorSpaceI :
  findim V  $\Longrightarrow$  FinDimVectorSpace ( $\cdot$ ) V
  by unfold-locales fast
```

```
context VectorSpace
begin
```

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

```
lemma SubspaceD1 : Subspace U  $\Longrightarrow$  VectorSpace smult U
  using VectorSpace.intro Module.intro by fast
```

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*: $\llbracket a \neq 0; u \in V; v \in V; a \cdot u = a \cdot v \rrbracket \implies u = v$
using *smult-assoc*[of 1/a a u] **by** *simp*

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* :
 $\llbracket v \in V; \text{set } vs \subseteq V; a \neq 0; (a \# as) \cdot (v \# vs) = 0 \rrbracket \implies v \in \text{Span } vs$
using *lincomb-Cons eq-neg-iff-add-eq-0*[of a · v as · vs]
neg-lincomb smult-assoc[of 1/a a v] *smult-lincomb SpanD-lincomb-arb-len-coeffs*
by *auto*

lemma *lin-independent-Cons-conditions* :
 $\llbracket v \in V; \text{set } vs \subseteq V; v \notin \text{Span } vs; \text{lin-independent } vs \rrbracket$
 $\implies \text{lin-independent } (v \# vs)$
using *lincomb-Cons-0-imp-in-Span lin-independent-ConsI* **by** *fast*

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 $v \in \text{Span } (w \# us @ vs)$
proof –
define x **where** $x = (1 \# [- c. c \leftarrow as @ bs]) \cdot (w \# us @ vs)$

```

from assms(1,4-6) have  $v = (1/b) \cdot (w + - ((as@bs) \cdot (us@vs)))$ 
  using lincomb-append lincomb-Cons by simp
moreover from assms(1,2,3,6) have  $w: w \in V$  using lincomb-closed by simp
ultimately have  $v = (1/b) \cdot x$ 
  using x-def assms(2,3) neg-lincomb[of - us@vs] lincomb-Cons[of 1 - w] by simp
with x-def w assms(2,3) show ?thesis
  using SpanD-lincomb-arb-len-coeffs[of  $w \# us @ vs$ ]
    Span-smult-closed[of  $1/b w \# us @ vs x$ ]
  by auto
qed

```

lemma *lin-independent-replace1-by-lincomb* :

```

assumes  $set\ us \subseteq V\ v \in V\ set\ vs \subseteq V\ lin-independent\ (us\ @\ v\ \# \ vs)$ 
   $length\ as = length\ us\ b \neq 0$ 
shows  $lin-independent\ (((as\ @\ b\ \# \ bs) \cdot (us\ @\ v\ \# \ vs)) \# \ us\ @\ vs)$ 
proof -

```

```

define w where  $w = (as\ @\ b\ \# \ bs) \cdot (us\ @\ v\ \# \ vs)$ 

```

```

from assms(1,2,4) have  $lin-independent\ (us\ @\ vs)$ 

```

```

  using lin-independent-reduce by fast

```

```

hence  $lin-independent\ (w\ \# \ us\ @\ vs)$ 

```

```

proof (rule lin-independent-ConsI)

```

```

  fix c cs assume  $A: (c\ \# \ cs) \cdot (w\ \# \ us\ @\ vs) = 0$ 

```

```

  from assms(1,3) obtain ds es fs

```

```

    where dsesfs:  $length\ ds = length\ vs\ bs \cdot vs = ds \cdot vs$ 

```

```

       $length\ es = length\ vs\ (drop\ (length\ us)\ cs) \cdot vs = es \cdot vs$ 

```

```

       $length\ fs = length\ us\ cs \cdot us = fs \cdot us$ 

```

```

    using lincomb-obtain-same-length-coeffs[of bs vs]

```

```

      lincomb-obtain-same-length-coeffs[of  $drop\ (length\ us)\ cs\ vs$ ]

```

```

      lincomb-obtain-same-length-coeffs[of cs us]

```

```

    by auto

```

```

define xs ys

```

```

  where  $xs = [x+y.\ (x,y)\ \leftarrow\ zip\ [c*a.\ a\ \leftarrow\ as]\ fs]$ 

```

```

  and  $ys = [x+y.\ (x,y)\ \leftarrow\ zip\ es\ [c*d.\ d\ \leftarrow\ ds]]$ 

```

```

with assms(5) dsesfs(5) have  $len\ xs: length\ xs = length\ us$ 

```

```

  using length-concat-map-split-zip[of - [c*a.\ a\ \leftarrow\ as] fs] by simp

```

```

from  $A$  w-def assms(1-3,5) dsesfs(2,4,6)

```

```

  have  $0 = c \cdot as \cdot us + fs \cdot us + (c * b) \cdot v + es \cdot vs + c \cdot ds \cdot vs$ 

```

```

using lincomb-Cons lincomb-append-right lincomb-append add-closed smult-closed
  lincomb-closed

```

```

  by (simp add: algebra-simps)

```

```

also from assms(1,3,5) dsesfs(1,3,5) xs-def ys-def len-xs

```

```

  have  $\dots = (xs\ @\ (c * b)\ \# \ ys) \cdot (us\ @\ v\ \# \ vs)$ 

```

```

  using smult-lincomb lincomb-sum lincomb-Cons lincomb-append by simp

```

```

finally have  $(xs\ @\ (c * b)\ \# \ ys) \cdot (us\ @\ v\ \# \ vs) = 0$  by simp

```

```

with assms(1-3,4,6) len-xs show  $c = 0$ 

```

```

  using lin-independent-vs-lincomb0 by fastforce

```

```

qed

```

```

with w-def show ?thesis by fast

```

```

qed

```

```

lemma build-lin-independent-seq :
  assumes us-V: set us  $\subseteq V$ 
  and indep-us: lin-independent us
  shows  $\exists ws. \text{set } ws \subseteq V \wedge \text{lin-independent } (ws @ us) \wedge (\text{Span } (ws @ us) = V$ 
     $\vee \text{length } ws = n)$ 
proof (induct n)
  case 0 from indep-us show ?case by force
next
  case (Suc m)
  from this obtain ws
    where ws: set ws  $\subseteq V$  lin-independent (ws @ us)
      Span (ws@us) =  $V \vee \text{length } ws = m$ 
    by auto
  show ?case
  proof (cases V = Span (ws@us))
    case True with ws show ?thesis by fast
  next
    case False
    moreover from ws(1) us-V have ws-us-V: set (ws @ us)  $\subseteq V$  by simp
    ultimately have Span (ws@us)  $\subset V$  using Span-closed by fast
    from this obtain w where w:  $w \in V \wedge w \notin \text{Span } (ws@us)$  by fast
    define vs where vs = w # ws
    with w ws-us-V ws(2,3)
      have set (vs @ us)  $\subseteq V$  lin-independent (vs @ us) length vs = Suc m
      using lin-independent-Cons-conditions[of w ws@us]
      by auto
    thus ?thesis by auto
  qed
qed

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

```

```

context VectorSpace
begin

```

```

lemma spanset-contains-basis :
  set vs  $\subseteq V \implies \exists us. \text{set } us \subseteq \text{set } vs \wedge \text{basis-for } (\text{Span } vs) us$ 
proof (induct vs)
  case Nil show ?case using lin-independent-Nil by simp
next
  case (Cons v vs)
  from this obtain us where us: set us  $\subseteq \text{set } vs$  basis-for (Span vs) us by auto
  show ?case

```

```

proof (cases v ∈ Span vs)
  case True
    with Cons(2) ws(2) have basis-for (Span (v#vs)) ws
      using spanset-reduce-Cons by force
    with ws(1) show ?thesis by auto
  next
    case False
    from Cons(2) ws
      have set (v#ws) ⊆ set (v#vs) set (v#ws) ⊆ Span (v#vs)
        Span (v#vs) = Span (v#ws)
      using Span-contains-spanset[of v#vs]
        Span-contains-Spans-Cons-right[of v vs] Span-Cons
      by auto
    moreover have lin-independent (v#ws)
    proof (rule lin-independent-Cons-conditions)
      from Cons(2) ws(1) show v ∈ V set ws ⊆ V by auto
      from ws(2) False show v ∉ Span ws lin-independent ws by auto
    qed
    ultimately show ?thesis by blast
  qed
qed

```

```

lemma basis-for-Span-ex : set vs ⊆ V ⇒ ∃ us. basis-for (Span vs) us
  using spanset-contains-basis by fastforce

```

```

lemma replace-basis-one-step :

```

```

  assumes closed: set vs ⊆ V set us ⊆ V and indep: lin-independent (us@vs)
  and new-w: w ∈ Span (us@vs) − Span us
  shows ∃ xs y ys. vs = xs @ y # ys
    ∧ basis-for (Span (us@vs)) (w # us @ xs @ ys)

```

```

proof −

```

```

  from new-w obtain u v where uv: u ∈ Span us v ∈ Span vs w = u + v
    using Span-append set-plus-def[of Span us] by auto
  from uv(1,3) new-w have v-n0: v ≠ 0 by auto
  from uv(1,2) obtain as bs
    where as-bs: length as = length us u = as .. us length bs = length vs
      v = bs .. vs
    using in-Span-obtain-same-length-coeffs
    by blast
  from v-n0 as-bs(4) closed(1) obtain b where b: b ∈ set bs b ≠ 0
    using lincomb-0coeffs[of vs] by auto
  from b(1) obtain cs ds where cs-ds: bs = cs @ b # ds using split-list by fast
  define n where n = length cs
  define fvs where fvs = take n vs
  define y where y = vs!n
  define bvs where bvs = drop (Suc n) vs
  define ufvs where ufvs = us @ fvs
  define acs where acs = as @ cs
  from as-bs(1,3) cs-ds n-def acs-def ufvs-def fvs-def

```

```

    have  $n\text{-len-vs}$ :  $n < \text{length } vs$  and  $\text{len-acs}$ :  $\text{length } acs = \text{length } ufvs$ 
  by auto
  from  $n\text{-len-vs}$   $fvs\text{-def}$   $y\text{-def}$   $bvs\text{-def}$  have  $vs\text{-decomp}$ :  $vs = fvs @ y \# bvs$ 
  using  $id\text{-take-nth-drop}$  by simp
  with  $w(3)$   $as\text{-bs}(1,2,4)$   $cs\text{-ds}$   $acs\text{-def}$   $ufvs\text{-def}$ 
  have  $w\text{-decomp}$ :  $w = (acs @ b \# ds) \cdot (ufvs @ y \# bvs)$ 
  using  $lincomb\text{-append}$ 
  by simp
  from  $closed(1)$   $vs\text{-decomp}$ 
  have  $y\text{-V}$ :  $y \in V$  and  $fvs\text{-V}$ :  $\text{set } fvs \subseteq V$  and  $bvs\text{-V}$ :  $\text{set } bvs \subseteq V$ 
  by auto
  from  $ufvs\text{-def}$   $fvs\text{-V}$   $closed(2)$  have  $ufvs\text{-V}$ :  $\text{set } ufvs \subseteq V$  by simp
  from  $w\text{-decomp}$   $ufvs\text{-V}$   $y\text{-V}$   $bvs\text{-V}$  have  $w\text{-V}$ :  $w \in V$ 
  using  $lincomb\text{-closed}$  by simp
  have  $\text{Span } (us@vs) = \text{Span } (w \# ufvs @ bvs)$ 
  proof
    from  $vs\text{-decomp}$   $ufvs\text{-def}$  have 1:  $\text{Span } (us@vs) = \text{Span } (y \# ufvs @ bvs)$ 
    using  $\text{Span-append}$   $\text{Span-Cons}[of y bvs]$   $\text{Span-Cons}[of y ufvs]$ 
       $\text{Span-append}[of y\#ufvs bvs]$ 
    by (simp add: algebra-simps)
    with  $new\text{-w}$   $y\text{-V}$   $ufvs\text{-V}$   $bvs\text{-V}$  show  $\text{Span } (w \# ufvs @ bvs) \subseteq \text{Span } (us@vs)$ 
    using  $\text{Span-replace-hd}$  by simp
    from  $\text{len-acs}$   $w\text{-decomp}$   $y\text{-V}$   $ufvs\text{-V}$   $bvs\text{-V}$  have  $y \in \text{Span } (w \# ufvs @ bvs)$ 
    using  $b(2)$   $\text{coeff-n0-imp-in-Span-others}$  by simp
    with  $w\text{-V}$   $ufvs\text{-V}$   $bvs\text{-V}$  have  $\text{Span } (y \# ufvs @ bvs) \subseteq \text{Span } (w \# ufvs @ bvs)$ 
    using  $\text{Span-replace-hd}$  by simp
    with 1 show  $\text{Span } (us@vs) \subseteq \text{Span } (w \# ufvs @ bvs)$  by fast
  qed
  moreover from  $ufvs\text{-V}$   $y\text{-V}$   $bvs\text{-V}$   $ufvs\text{-def}$   $\text{indep}$   $vs\text{-decomp}$   $w\text{-decomp}$   $\text{len-acs}$ 
   $b(2)$ 
  have  $\text{lin-independent } (w \# ufvs @ bvs)$ 
  using  $\text{lin-independent-replace1-by-lincomb}[of ufvs y bvs acs b ds]$ 
  by simp
  moreover have  $\text{set } (w \# (us@fvs) @ bvs) \subseteq \text{Span } (us@vs)$ 
  proof-
    from  $new\text{-w}$  have  $w \in \text{Span } (us@vs)$  by fast
    moreover from  $closed$  have  $\text{set } us \subseteq \text{Span } (us@vs)$ 
    using  $\text{Span-contains-spanset-append-left}$  by fast
    moreover from  $closed$   $fvs\text{-def}$  have  $\text{set } fvs \subseteq \text{Span } (us@vs)$ 
    using  $\text{Span-contains-spanset-append-right}[of us]$   $\text{set-take-subset}$  by fastforce
    moreover from  $closed$   $bvs\text{-def}$  have  $\text{set } bvs \subseteq \text{Span } (us@vs)$ 
    using  $\text{Span-contains-spanset-append-right}[of us]$   $\text{set-drop-subset}$  by fastforce
    ultimately show  $?thesis$  by simp
  qed
  ultimately show  $?thesis$  using  $ufvs\text{-def}$   $vs\text{-decomp}$  by auto
  qed

```

lemma replace-basis :

assumes $closed$: $\text{set } vs \subseteq V$ and indep-vs : $\text{lin-independent } vs$

shows $\llbracket \text{length } us \leq \text{length } vs; \text{set } us \subseteq \text{Span } vs; \text{lin-independent } us \rrbracket$
 $\implies \exists pvs. \text{length } pvs = \text{length } vs \wedge \text{set } pvs = \text{set } vs$
 $\wedge \text{basis-for } (\text{Span } vs) (\text{take } (\text{length } vs) (us @ pvs))$

proof (*induct us*)
case *Nil* **from** *closed indep-vs* **show** *?case*
using *Span-contains-spanset[of vs]* **by** *fastforce*

next
case (*Cons u us*)
from *this* **obtain** *ppvs*
where *ppvs*: $\text{length } ppvs = \text{length } vs$ $\text{set } ppvs = \text{set } vs$
 $\text{basis-for } (\text{Span } vs) (\text{take } (\text{length } vs) (us @ ppvs))$
using *lin-independent-ConsD1*[*of u us*]
by *auto*

define *fppvs bppvs*
where *fppvs* = $\text{take } (\text{length } vs - \text{length } us) ppvs$
and *bppvs* = $\text{drop } (\text{length } vs - \text{length } us) ppvs$

with *ppvs(1)* *Cons(2)*
have *ppvs-decomp*: $ppvs = fppvs @ bppvs$
and *len-fppvs* : $\text{length } fppvs = \text{length } vs - \text{length } us$
by *auto*

from *closed Cons(3)* **have** *uus-V*: $u \in V$ $\text{set } us \subseteq V$
using *Span-closed* **by** *auto*

from *closed ppvs(2)* **have** $\text{set } ppvs \subseteq V$ **by** *fast*

with *fppvs-def* **have** *fppvs-V*: $\text{set } fppvs \subseteq V$ **using** *set-take-subset[of - ppvs]* **by**
fast

from *fppvs-def Cons(2)*
have *prev-basis-decomp*: $\text{take } (\text{length } vs) (us @ ppvs) = us @ fppvs$
by *auto*

with *Cons(3,4)* *ppvs(3)* *fppvs-V* *uus-V* **obtain** *xs y ys*
where *xs-y-ys*: $fppvs = xs @ y \# ys$ $\text{basis-for } (\text{Span } vs) (u \# us @ xs @ ys)$
using *lin-independent-imp-hd-independent-from-Span*
 $\text{replace-basis-one-step}[of fppvs us u]$
by *auto*

define *pvs* **where** $pvs = xs @ ys @ y \# bppvs$

with *xs-y-ys* *len-fppvs* *ppvs-decomp* *ppvs(1,2)*
have $\text{length } pvs = \text{length } vs$ $\text{set } pvs = \text{set } vs$
 $\text{basis-for } (\text{Span } vs) (\text{take } (\text{length } vs) ((u \# us) @ pvs))$
using *take-append*[*of length vs u \# us @ xs @ ys*]
by *auto*

thus *?case* **by** *fast*

qed

lemma *replace-basis-completely* :
 $\llbracket \text{set } vs \subseteq V; \text{lin-independent } vs; \text{length } us = \text{length } vs;$
 $\text{set } us \subseteq \text{Span } vs; \text{lin-independent } us \rrbracket \implies \text{basis-for } (\text{Span } vs) us$
using *replace-basis*[*of vs us*] **by** *auto*

lemma *basis-for-obtain-unique-scalars* :
 $\text{basis-for } V vs \implies v \in V \implies \exists! as. \text{length } as = \text{length } vs \wedge v = as \cdot \cdot vs$

using *lin-independent-obtain-unique-scalars* by *fast*

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 –

from *vs v v' as bs*

have *as'*: $length\ as = length\ vs \wedge v = as \cdot vs$

and *bs'*: $length\ bs = length\ vs \wedge v' = bs \cdot vs$

using *basis-for-obtain-unique-scalars theI'*[

of $\lambda ds.\ length\ ds = length\ vs \wedge v = ds \cdot vs$

]

theI'[*of* $\lambda ds.\ length\ ds = length\ vs \wedge v' = ds \cdot vs$]

by *auto*

have $length\ [a+b.\ (a,b)\leftarrow zip\ as\ bs] = length\ (zip\ as\ bs)$

by (*induct as bs rule: list-induct2'*) *auto*

with *vs as' bs'*

have $length\ [a+b.\ (a,b)\leftarrow zip\ as\ bs]$

$= length\ vs \wedge v + v' = [a + b.\ (a,b)\leftarrow zip\ as\ bs] \cdot vs$

using *lincomb-sum*

by *auto*

moreover from *vs v v' have* $\exists!\ ds.\ length\ ds = length\ vs \wedge v+v' = ds \cdot vs$

using *add-closed basis-for-obtain-unique-scalars* by *force*

ultimately show *?thesis* **using** *cs the1-equality* by *fast*

qed

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

shows $bs = map\ ((*)\ a)\ as$

proof –

from *vs v as* **have** $length\ as = length\ vs \wedge v = as \cdot vs$

using *basis-for-obtain-unique-scalars theI'*[

of $\lambda cs.\ length\ cs = length\ vs \wedge v = cs \cdot vs$

]

by *auto*

with *vs* **have** $length\ (map\ ((*)\ a)\ as)$

$= length\ vs \wedge a \cdot v = (map\ ((*)\ a)\ as) \cdot vs$

using *smult-lincomb* by *auto*

moreover from *vs v* **have** $\exists!\ cs.\ length\ cs = length\ vs \wedge a \cdot v = cs \cdot vs$

using *smult-closed basis-for-obtain-unique-scalars* by *fast*

ultimately show *?thesis* **using** *bs the1-equality* by *fast*

qed

lemma *max-lin-independent-set-in-Span* :
assumes $set\ vs \subseteq V$ $set\ us \subseteq Span\ vs$ *lin-independent us*
shows $length\ us \leq length\ vs$
proof (*cases us*)
case (*Cons x xs*)
from *assms(1)* *spanset-contains-basis[of vs]* **obtain** *bvs*
where *bvs: set bvs* $\subseteq set\ vs$ *basis-for (Span vs) bvs*
by *auto*
with *assms(1)* **have** *len-bvs: length bvs* $\leq length\ vs$
using *lin-independent-imp-distinct[of bvs]* *distinct-card finite-set*
card-mono[of set vs set bvs] *card-length[of vs]*
by *fastforce*
moreover **have** $length\ (x\#\ xs) > length\ bvs \implies \neg\ lin\text{-}independent\ (x\#\ xs)$
proof
assume *A: length (x#xs) > length bvs lin-independent (x#xs)*
define *ws* **where** $ws = take\ (length\ bvs)\ xs$
from *Cons assms(1,2)* **have** *xxs-V: x* $\in V$ *set xs* $\subseteq V$
using *Span-closed* **by** *auto*
from *ws-def A(1)* **have** $length\ ws = length\ bvs$ **by** *simp*
moreover **from** *Cons assms(2)* *bvs(2)* *ws-def* **have** $set\ ws \subseteq Span\ bvs$
using *set-take-subset* **by** *fastforce*
ultimately **have** *basis-for (Span vs) ws*
using *A(2)* *ws-def assms(1)* *bvs* *xxs-V* *lin-independent-ConsD1*
lin-independent-imp-independent-take *replace-basis-completely[of bvs ws]*
by *force*
with *Cons assms(2)* *ws-def A(2)* *xxs-V* **show** *False*
using *Span-contains-Span-take[of xs]*
lin-independent-imp-hd-independent-from-Span[of x xs]
by *auto*
qed
ultimately **show** *?thesis* **using** *Cons assms(3)* **by** *fastforce*
qed *simp*

lemma *finrank-Span* : $set\ vs \subseteq V \implies finrank\ (Span\ vs)$
using *max-lin-independent-set-in-Span* *finrankI* **by** *blast*

end

4.3 Finite dimensional spaces

context *VectorSpace*

begin

lemma *dim-eq-size-basis* : *basis-for V vs* $\implies length\ vs = dim\ V$
using *max-lin-independent-set-in-Span*
Least-equality
of $\lambda n::nat. \exists us. length\ us = n \wedge set\ us \subseteq V \wedge RSpan\ us = V$ *length vs*
]
unfolding *dim-R-def* **by** *fastforce*

lemma *finrank-imp-findim* :
assumes *finrank V*
shows *findim V*
proof –
from *assms* **obtain** *n*
where $\forall vs. \text{set } vs \subseteq V \wedge \text{lin-independent } vs \longrightarrow \text{length } vs \leq n$
using *finrankD*
by *fastforce*
moreover from *build-lin-independent-seq*[of []] **obtain** *ws*
where $\text{set } ws \subseteq V \text{ lin-independent } ws \text{ Span } ws = V \vee \text{length } ws = \text{Suc } n$
by *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *subspace-Span-is-findim* :
 $\llbracket \text{set } vs \subseteq V; \text{Subspace } W; W \subseteq \text{Span } vs \rrbracket \Longrightarrow \text{findim } W$
using *finrank-Span* *subspace-finrank*[of *Span vs W*] *SubspaceD1*[of *W*]
VectorSpace.finrank-imp-findim
by *auto*

end

context *FinDimVectorSpace*
begin

lemma *Subspace-is-findim* : *Subspace U* \Longrightarrow *findim U*
using *findim* *subspace-Span-is-findim* **by** *fast*

lemma *basis-ex* : $\exists vs. \text{basis-for } V \text{ } vs$
using *findim* *basis-for-Span-ex* **by** *auto*

lemma *lin-independent-length-le-dim* :
 $\text{set } us \subseteq V \Longrightarrow \text{lin-independent } us \Longrightarrow \text{length } us \leq \text{dim } V$
using *basis-ex* *max-lin-independent-set-in-Span* *dim-eq-size-basis*
by *force*

lemma *too-long-lin-dependent* :
 $\text{set } us \subseteq V \Longrightarrow \text{length } us > \text{dim } V \Longrightarrow \neg \text{lin-independent } us$
using *lin-independent-length-le-dim* **by** *fastforce*

lemma *extend-lin-independent-to-basis* :
assumes $\text{set } us \subseteq V \text{ lin-independent } us$
shows $\exists vs. \text{basis-for } V (vs @ us)$
proof –
define *n* **where** $n = \text{Suc } (\text{dim } V - \text{length } us)$
from *assms* **obtain** *vs*
where $vs: \text{set } vs \subseteq V \text{ lin-independent } (vs @ us)$
 $\text{Span } (vs @ us) = V \vee \text{length } vs = n$

using *build-lin-independent-seq*[of *us n*]
by *fast*
with *assms n-def* **show** *?thesis*
using *set-append lin-independent-length-le-dim*[of *vs @ us*] **by** *auto*
qed

lemma *extend-Subspace-basis* :
 $U \subseteq V \implies \text{basis-for } U \text{ } us \implies \exists \text{ } vs. \text{ basis-for } V \text{ } (vs@us)$
using *Span-contains-spanset extend-lin-independent-to-basis* **by** *fast*

lemma *Subspace-dim-le* :
assumes *Subspace U*
shows $\dim U \leq \dim V$
using *assms findim*
proof –
from *assms* **obtain** *us* **where** *basis-for U us*
using *Subspace-is-findim SubspaceD1*
 $VectorSpace.FinDimVectorSpaceI$ [of (\cdot) *U*]
 $FinDimVectorSpace.basis-ex$ [of (\cdot) *U*]
by *auto*
with *assms* **show** *?thesis*
using *RSpan-contains-spanset*[of *us*] *lin-independent-length-le-dim*[of *us*]
 $SubspaceD1$ $VectorSpace.dim-eq-size-basis$ [of (\cdot) *U us*]
by *auto*
qed

lemma *Subspace-eqdim-imp-equal* :
assumes *Subspace U dim U = dim V*
shows $U = V$
proof –
from *assms(1)* **obtain** *us* **where** *us: basis-for U us*
using *Subspace-is-findim SubspaceD1*
 $VectorSpace.FinDimVectorSpaceI$ [of (\cdot) *U*]
 $FinDimVectorSpace.basis-ex$ [of (\cdot) *U*]
by *auto*
with *assms(1)* **obtain** *vs* **where** *vs: basis-for V (vs@us)*
using *extend-Subspace-basis*[of *U us*] **by** *fast*
from *assms us vs* **show** *?thesis*
using $SubspaceD1$ $VectorSpace.dim-eq-size-basis$ [of *smult U*]
 $dim-eq-size-basis$ [of *vs@us*]
by *auto*
qed

lemma *Subspace-dim-lt* : $Subspace U \implies U \neq V \implies \dim U < \dim V$
using *Subspace-dim-le Subspace-eqdim-imp-equal* **by** *fastforce*

lemma *semisimple* :
assumes *Subspace U*
shows $\exists W. Subspace W \wedge (V = W \oplus U)$

```

proof –
  from assms obtain us where us: basis-for U us
    using SubspaceD1 Subspace-is-findim VectorSpace.FinDim VectorSpaceI
      FinDim VectorSpace.basis-ex[of - U]
    by fastforce
  with assms obtain ws where basis: basis-for V (ws@us)
    using extend-Subspace-basis by fastforce
  hence ws-V: set ws ⊆ V and ind-ws-us: lin-independent (ws@us)
    and V-eq: V = Span (ws@us)
    by auto
  have V = Span ws ⊕ Span us
  proof (rule inner-dirsum-doubleI)
    from V-eq show V = Span ws + Span us using Span-append by fast
    from ws-V ind-ws-us show add-independentS [Span ws, Span us]
      using lin-independent-append-imp-independent-Spans by auto
  qed
  with us ws-V have Subspace (Span ws) ∧ V = (Span ws) ⊕ U
    using Subspace-Span by auto
  thus ?thesis by fast
qed

end

```

4.4 Vector space homomorphisms

4.4.1 Locales

```

locale VectorSpaceHom = ModuleHom smult V smult' T
  for smult :: 'f::field ⇒ 'v::ab-group-add ⇒ 'v (infixr · 70)
  and V :: 'v set
  and smult' :: 'f ⇒ 'w::ab-group-add ⇒ 'w (infixr ★ 70)
  and T :: 'v ⇒ 'w

```

```

sublocale VectorSpaceHom < VectorSpace ..

```

```

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
  using assms
  by unfold-locales

```

```

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

shows *VectorSpaceEnd* $smult\ V\ T$

by (rule *VectorSpaceEnd.intro*, rule *assms(1)*, *unfold-locales*, rule *assms(2)*)

lemma (in *VectorSpaceEnd*) *VectorSpaceHom*: *VectorSpaceHom* $smult\ V\ smult\ T$

..

lemma (in *VectorSpaceEnd*) *ModuleEnd* : *ModuleEnd* $smult\ V\ T$

using *endomorph* *ModuleEnd.intro* **by** *unfold-locales*

locale *VectorSpaceIso* = *VectorSpaceHom* $smult\ V\ smult'\ T$

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

and $V :: 'v\ set$

and $smult' :: 'f \Rightarrow 'w::ab-group-add \Rightarrow 'w$ (**infixr** \star 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* $smult'\ W \equiv (\exists T. VectorSpaceIso\ smult\ V\ smult'\ T\ W)$

4.4.2 Basic facts

lemma (in *VectorSpace*) *trivial-VectorSpaceHom* :

$(\bigwedge a. smult'\ a\ 0 = 0) \implies VectorSpaceHom\ smult\ V\ smult'\ 0$

using *trivial-RModuleHom*[of *smult'*] *ModuleHom.intro* *VectorSpaceHom.intro*

by *fast*

lemma (in *VectorSpace*) *VectorSpaceHom-idhom* :

VectorSpaceHom $smult\ V\ smult\ (id \downarrow V)$

using *smult-zero* *RModHom-idhom* *ModuleHom.intro* *VectorSpaceHom.intro*

by *fast*

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*
 using *RModule-Im VectorSpace.intro Module.intro* **by** *fast*

lemma *VectorSpaceHom-scalar-mul* :
VectorSpaceHom smult V smult' (λv. a ★ T v)
proof–
show $\bigwedge v v'. v \in V \implies v' \in V \implies a \star T (v + v') = a \star T v + a \star T v'$
 using *additive VectorSpace.smult-distrib-left*[*OF VectorSpace-Im*] **by** *simp*
have $\bigwedge v. v \notin V \implies v \notin \text{supp } (\lambda v. a \star T v)$
proof–
fix *v* **assume** $v \notin V$
hence $a \star T v = 0$
 using *supp suppI-contr*[*of - T*] *codomain-smult-zero* **by** *fastforce*
thus $v \notin \text{supp } (\lambda v. a \star T v)$ **using** *suppD-contr* **by** *fast*
qed
thus $\text{supp } (\lambda v. a \star T v) \subseteq V$ **by** *fast*
show $\bigwedge c v. v \in V \implies a \star T (c \cdot v) = c \star a \star T v$
 using *f-map VectorSpace.smult-assoc*[*OF VectorSpace-Im*] **by** (*simp add: field-simps*)
qed

lemma *VectorSpaceHom-composite-left* :
assumes $\text{Im}G \subseteq W$ *VectorSpaceHom smult' W smult'' S*
shows *VectorSpaceHom smult V smult'' (S ○ T)*
proof–
have *RModuleHom UNIV smult' W smult'' S*
 using *VectorSpaceHom.axioms(1)*[*OF assms(2)*] *ModuleHom.axioms(1)*
by *fast*
with *assms(1)* **have** *RModuleHom UNIV smult V smult'' (S ○ T)*
 using *RModHom-composite-left*[*of W*] **by** *fast*
thus *?thesis* **using** *ModuleHom.intro VectorSpaceHom.intro* **by** *fast*
qed

lemma *findim-domain-findim-image* :
assumes *findim V*
shows *fscalar-mult.findim smult' ImG*
proof–
from *assms* **obtain** *vs* **where** $vs: \text{set } vs \subseteq V$ *scalar-mult.Span smult vs = V*
by *fast*
define *ws* **where** $ws = \text{map } T \text{ vs}$
with *vs(1)* **have** $1: \text{set } ws \subseteq \text{Im}G$ **by** *auto*
moreover **have** $\text{Span } ws = \text{Im}G$

```

proof
  show  $\text{Span } ws \subseteq \text{Im}G$ 
    using 1 VectorSpace.Span-closed[OF VectorSpace-Im] by fast
  from vs ws-def show  $\text{Span } ws \supseteq \text{Im}G$ 
    using 1 SpanD-lincomb-arb-len-coeffs distrib-lincomb
      VectorSpace.SpanD-lincomb-arb-len-coeffs[OF VectorSpace-Im]
    by auto
  qed
  ultimately show ?thesis by fast
qed

end

lemma (in VectorSpace) basis-im-defines-hom :
  fixes smult' :: 'f  $\Rightarrow$  'w::ab-group-add  $\Rightarrow$  'w (infixr  $\star$  70)
  and lincomb' :: 'f list  $\Rightarrow$  'w list  $\Rightarrow$  'w (infixr  $\cdot\star$  70)
  defines lincomb' : lincomb'  $\equiv$  scalar-mult.lincomb smult'
  assumes VSpW : VectorSpace smult' W
  and basisV : basis-for V vs
  and basisV-im : set ws  $\subseteq$  W length ws = length vs
  shows  $\exists!$  T. VectorSpaceHom smult V smult' T  $\wedge$  map T vs = ws
proof (rule ex-ex1I)
  define T where T = restrict0 ( $\lambda v$ . (THE as. length as = length vs  $\wedge$  v = as  $\cdot\cdot$ 
vs)  $\cdot\star$  ws) V
  have VectorSpaceHom ( $\cdot$ ) V smult' T
  proof
    fix v v' assume vv': v  $\in$  V v'  $\in$  V
    with T-def lincomb' basisV basisV-im(1) show T (v + v') = T v + T v'
    using basis-for-obtain-unique-scalars theI'[
      of  $\lambda ds$ . length ds = length vs  $\wedge$  v = ds  $\cdot\cdot$  vs
    ]
    theI'[of  $\lambda ds$ . length ds = length vs  $\wedge$  v' = ds  $\cdot\cdot$  vs] add-closed
    add-unique-scalars VectorSpace.lincomb-sum[OF VSpW]
    by auto
  next
  from T-def show supp T  $\subseteq$  V using supp-restrict0 by fast
  next
  fix a v assume v: v  $\in$  V
  with basisV basisV-im(1) T-def lincomb' show T (a  $\cdot$  v) = a  $\star$  T v
  using smult-closed smult-unique-scalars VectorSpace.smult-lincomb[OF VSpW]
by auto
  qed
  moreover have map T vs = ws
  proof (rule nth-equalityI)
  from basisV-im(2) show length (map T vs) = length ws by simp
  have  $\bigwedge i$ .  $i < \text{length} (\text{map } T \text{ vs}) \implies \text{map } T \text{ vs} ! i = \text{ws} ! i$ 
  proof–
  fix i assume i:  $i < \text{length} (\text{map } T \text{ vs})$ 
  define zs where zs = (replicate (length vs) (0::'f))[i:=1]

```


with *basisV* *i* **have** $\text{length } zs = \text{length } vs \wedge vs!i = zs \cdot vs$
using *delta-scalars-lincomb-eq-nth* **by** *auto*
moreover from *basisV* *i* **have** $vs!i \in V$ **by** *auto*
ultimately show $(\text{map } T \text{ vs})!i = ws!i$
using *basisV* *basisV-im* *T-def* *lincomb'* *zs-def* *i*
basis-for-obtain-unique-scalars[*of vs vs!i*]
the1-equality[*of $\lambda zs. \text{length } zs = \text{length } vs \wedge vs!i = zs \cdot vs$*]
VectorSpace.delta-scalars-lincomb-eq-nth[*OF VSpW, of ws*]
by *force*
qed
thus $\bigwedge i. i < \text{length } (\text{map } T \text{ vs}) \implies \text{map } T \text{ vs } ! i = ws ! i$ **by** *fast*
qed
ultimately have $\text{VectorSpaceHom } (\cdot) V \text{ smult}' T \wedge \text{map } T \text{ vs} = ws$ **by** *fast*
thus $\exists T. \text{VectorSpaceHom } (\cdot) V \text{ smult}' T \wedge \text{map } T \text{ vs} = ws$ **by** *fast*
next
fix *S* *T* **assume**
 $\text{VectorSpaceHom } (\cdot) V \text{ smult}' S \wedge \text{map } S \text{ vs} = ws$
 $\text{VectorSpaceHom } (\cdot) V \text{ smult}' T \wedge \text{map } T \text{ vs} = ws$
with *basisV* **show** $S = T$
using *VectorSpaceHom.same-image-on-spanset-imp-same-hom* *map-eq-conv*
by *fastforce*
qed

4.4.3 Hom-sets

definition *VectorSpaceHomSet* ::
 $(f :: \text{field} \implies 'v :: \text{ab-group-add} \implies 'v) \text{ set} \implies (f \implies 'w :: \text{ab-group-add} \implies 'w) \implies 'w \text{ set} \implies ('v \implies 'w) \text{ set}$
where $\text{VectorSpaceHomSet } fsmult \ V \ fsmult' \ W$
 $\equiv \{T. \text{VectorSpaceHom } fsmult \ V \ fsmult' \ T\} \cap \{T. T \text{ ' } V \subseteq W\}$

abbreviation (in *VectorSpace*) $\text{VectorSpaceEndSet} \equiv \{S. V \text{End } S\}$

lemma *VectorSpaceHomSetI* :
 $\text{VectorSpaceHom } fsmult \ V \ fsmult' \ T \implies T \text{ ' } V \subseteq W$
 $\implies T \in \text{VectorSpaceHomSet } fsmult \ V \ fsmult' \ W$
unfolding *VectorSpaceHomSet-def* **by** *fast*

lemma *VectorSpaceHomSetD-VectorSpaceHom* :
 $T \in \text{VectorSpaceHomSet } fsmult \ V \ fsmult' \ N$
 $\implies \text{VectorSpaceHom } fsmult \ V \ fsmult' \ T$
unfolding *VectorSpaceHomSet-def* **by** *fast*

lemma *VectorSpaceHomSetD-Im* :
 $T \in \text{VectorSpaceHomSet } fsmult \ V \ fsmult' \ W \implies T \text{ ' } V \subseteq W$
unfolding *VectorSpaceHomSet-def* **by** *fast*

context *VectorSpace*
begin

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

fixes *smult'* :: '*f* ⇒ '*w*::*ab-group-add* ⇒ '*w* (**infixr** ★ 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

show *VectorSpaceHomSet smult V smult' W*
 $\subseteq (\text{GroupHomSet } V \ W) \cap \{T. \forall a. \forall v \in V. T (a \cdot v) = a \star (T v)\}$

using *VectorSpaceHomSetD-VectorSpaceHom[of - smult]*
VectorSpaceHomSetD-Im[of - smult]
VectorSpaceHom.GroupHom[of smult] GroupHomSetI[of V - W]
VectorSpaceHom.f-map[of smult]

by *fastforce*

show *VectorSpaceHomSet smult V smult' W*
 $\supseteq (\text{GroupHomSet } V \ W) \cap \{T. \forall a. \forall v \in V. T (a \cdot v) = a \star (T v)\}$

proof

fix *T* **assume** *T*: *T* ∈ (*GroupHomSet V W*)
 $\cap \{T. \forall a. \forall v \in V. T (a \cdot v) = a \star (T v)\}$

have *VectorSpaceHom smult V smult' T*

proof (*rule VectorSpaceHom.intro, rule ModuleHom.intro, rule RModuleHom.intro*)

show *RModule UNIV (·) V ..*

from *T* **show** *GroupHom V T* **using** *GroupHomSetD-GroupHom* **by** *fast*

from *T* **show** *RModuleHom-axioms UNIV smult V smult' T*
by *unfold-locales fast*

qed

with *T* **show** *T* ∈ *VectorSpaceHomSet smult V smult' W*
using *GroupHomSetD-Im[of T] VectorSpaceHomSetI* **by** *fastforce*

qed

qed

lemma *Group-VectorSpaceHomSet* :

fixes *smult'* :: '*f* ⇒ '*w*::*ab-group-add* ⇒ '*w* (**infixr** ★ 70)

assumes *VectorSpace smult' W*

shows *Group (VectorSpaceHomSet smult V smult' W)*

proof

show *VectorSpaceHomSet smult V smult' W ≠ {}*
using *VectorSpace.smult-zero[OF assms] VectorSpace.zero-closed[OF assms]*
trivial-VectorSpaceHom[of smult'] VectorSpaceHomSetI

by *fastforce*

next

fix *S T*

assume *S*: *S* ∈ *VectorSpaceHomSet smult V smult' W*
and *T*: *T* ∈ *VectorSpaceHomSet smult V smult' W*

from *S T*

have *ST*: *S* ∈ (*GroupHomSet V W*)
 $\cap \{T. \forall a. \forall v \in V. T (a \cdot v) = a \star (T v)\}$
 $T \in (\text{GroupHomSet } V \ W) \cap \{T. \forall a. \forall v \in V. T (a \cdot v) = a \star (T v)\}$

using *VectorSpaceHomSet-is-fmaps-in-GroupHomSet*
by *auto*

hence $S - T \in \text{GroupHomSet } V \ W$
using $\text{VectorSpace.AbGroup}[OF \ \text{assms}] \ \text{Group-GroupHomSet} \ \text{Group.diff-closed}$
by fast
moreover have $\bigwedge a \ v. \ v \in V \implies (S - T) (a \cdot v) = a \star ((S - T) v)$
proof –
fix $a \ v$ **assume** $v \in V$
with ST **show** $(S - T) (a \cdot v) = a \star ((S - T) v)$
using GroupHomSetD-Im
 $\text{VectorSpace.smult-distrib-left-diff}[OF \ \text{assms}, \ \text{of } a \ S \ v \ T \ v]$
by fastforce
qed
ultimately show $S - T \in \text{VectorSpaceHomSet } (\cdot) \ V \ (\star) \ W$
using $\text{VectorSpaceHomSet-is-fmaps-in-GroupHomSet}[of \ \text{smult}' \ W]$ **by** fast
qed

lemma $\text{VectorSpace-VectorSpaceHomSet}$:
fixes $\text{smult}' \ :: 'f \Rightarrow 'w :: \text{ab-group-add} \Rightarrow 'w$ (**infixr** \star 70)
and $\text{hom-smult} \ :: 'f \Rightarrow ('v \Rightarrow 'w) \Rightarrow ('v \Rightarrow 'w)$ (**infixr** \star 70)
defines $\text{hom-smult}: \text{hom-smult} \equiv \lambda a \ T \ v. \ a \star T \ v$
assumes $VSpW: \text{VectorSpace} \ \text{smult}' \ W$
shows $\text{VectorSpace} \ \text{hom-smult} \ (\text{VectorSpaceHomSet} \ \text{smult}' \ V \ \text{smult}' \ W)$
proof ($\text{rule } \text{VectorSpace.intro}, \ \text{rule } \text{Module.intro}, \ \text{rule } \text{RModule.intro}, \ \text{rule } \text{R-scalar-mult}$)

from $VSpW$ **show** $\text{Group} \ (\text{VectorSpaceHomSet} \ (\cdot) \ V \ (\star) \ W)$
using $\text{Group-VectorSpaceHomSet}$ **by** fast

show $\text{RModule-axioms} \ \text{UNIV} \ \text{hom-smult} \ (\text{VectorSpaceHomSet} \ (\cdot) \ V \ (\star) \ W)$
proof
fix $a \ b \ S \ T$
assume $S: S \in \text{VectorSpaceHomSet} \ (\cdot) \ V \ (\star) \ W$
and $T: T \in \text{VectorSpaceHomSet} \ (\cdot) \ V \ (\star) \ W$
show $a \star T \in \text{VectorSpaceHomSet} \ (\cdot) \ V \ (\star) \ W$
proof ($\text{rule } \text{VectorSpaceHomSetI}$)
from $\text{assms } T$ **show** $\text{VectorSpaceHom} \ (\cdot) \ V \ (\star) \ (a \star T)$
using $\text{VectorSpaceHomSetD-VectorSpaceHom} \ \text{VectorSpaceHomSetD-Im}$
 $\text{VectorSpaceHom.VectorSpaceHom-scalar-mul}$
by fast
from hom-smult **show** $(a \star T) ' V \subseteq W$
using $\text{VectorSpaceHomSetD-Im}[OF \ T] \ \text{VectorSpace.smult-closed}[OF \ VSpW]$
by auto
qed

show $a \star (S + T) = a \star S + a \star T$
proof
fix v **from** assms **show** $(a \star (S + T)) v = (a \star S + a \star T) v$
using $\text{VectorSpaceHomSetD-Im}[OF \ S] \ \text{VectorSpaceHomSetD-Im}[OF \ T]$
 $\text{VectorSpace.smult-distrib-left}[OF \ VSpW, \ \text{of } a \ S \ v \ T \ v]$
 $\text{VectorSpaceHomSetD-VectorSpaceHom}[OF \ S]$
 $\text{VectorSpaceHomSetD-VectorSpaceHom}[OF \ S]$

```

      VectorSpaceHom.supp suppI-contr[of v S] suppI-contr[of v T]
      VectorSpace.smult-zero
    by fastforce
  qed

show (a + b) ★ T = a ★ T + b ★ T
proof
  fix v from assms show ((a + b) ★ T) v = (a ★ T + b ★ T) v
  using VectorSpaceHom.SetD-Im[OF T] VectorSpace.smult-distrib-right
        VectorSpaceHom.SetD-VectorSpaceHom[OF T] VectorSpaceHom.supp
        suppI-contr[of v] VectorSpace.smult-zero
  by fastforce
  qed

show a ★ b ★ T = (a * b) ★ T
proof
  fix v from assms show (a ★ b ★ T) v = ((a * b) ★ T) v
  using VectorSpaceHom.SetD-Im[OF T] VectorSpace.smult-assoc
        VectorSpaceHom.SetD-VectorSpaceHom[OF T]
        VectorSpaceHom.supp suppI-contr[of v]
        VectorSpace.smult-zero[OF VSpW, of b]
        VectorSpace.smult-zero[OF VSpW, of a]
        VectorSpace.smult-zero[OF VSpW, of a * b]
  by fastforce
  qed

show 1 ★ T = T
proof
  fix v from assms T show (1 ★ T) v = T v
  using VectorSpaceHom.SetD-Im VectorSpace.one-smult
        VectorSpaceHom.SetD-VectorSpaceHom VectorSpaceHom.supp
        suppI-contr[of v] VectorSpace.smult-zero
  by fastforce
  qed

qed
qed

end

```

4.4.4 Basic facts about endomorphisms

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

fixes *smult* :: 'f::field ⇒ 'v::ab-group-add ⇒ 'v

assumes *ModuleEnd smult V T*

shows *VectorSpaceEnd smult V T*

proof (*rule VectorSpaceEndI, rule VectorSpaceHom.intro*)

from *assms* **show** *ModuleHom smult V smult T*

using *ModuleEnd.axioms(1)* **by** *fast*

from *assms* **show** $T \subseteq V \subseteq V$ **using** *ModuleEnd.endomorph* **by** *fast*
qed

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* :: $'f \Rightarrow ('v \Rightarrow 'v) \Rightarrow ('v \Rightarrow 'v)$ (**infixr** \cdot 70)
where $a \cdot T \equiv (\lambda v. a \cdot T v)$

abbreviation *end-lincomb*
 :: $'f \text{ list} \Rightarrow (('v \Rightarrow 'v) \text{ list}) \Rightarrow ('v \Rightarrow 'v)$ (**infixr** \dots 70)
where *end-lincomb* \equiv *scalar-mult.lincomb end-smult*

lemma *end-smult0*: $a \cdot 0 = 0$
using *smult-zero* **by** *auto*

lemma *end-0smult*: $\text{range } T \subseteq V \implies 0 \cdot T = 0$
using *zero-smult* **by** *fastforce*

lemma *end-smult-distrib-left* :
assumes $\text{range } S \subseteq V \text{ range } T \subseteq V$
shows $a \cdot (S + T) = a \cdot S + a \cdot T$
proof
fix v **from** *assms* **show** $(a \cdot (S + T)) v = (a \cdot S + a \cdot T) v$
using *smult-distrib-left*[*of a S v T v*] **by** *fastforce*
qed

lemma *end-smult-distrib-right* :
assumes $\text{range } T \subseteq V$
shows $(a+b) \cdot T = a \cdot T + b \cdot T$
proof
fix v **from** *assms* **show** $((a+b) \cdot T) v = (a \cdot T + b \cdot T) v$
using *smult-distrib-right*[*of a b T v*] **by** *fastforce*
qed

lemma *end-smult-assoc* :
assumes $\text{range } T \subseteq V$
shows $a \cdot b \cdot T = (a * b) \cdot T$
proof
fix v **from** *assms* **show** $(a \cdot b \cdot T) v = ((a * b) \cdot T) v$
using *smult-assoc*[*of a b T v*] **by** *fastforce*
qed

lemma *end-smult-comp-comm-left* : $(a \cdot T) \circ S = a \cdot (T \circ S)$
by *auto*

lemma *end-idhom* : $V\text{End}(id \downarrow V)$
by (*rule VectorSpaceEnd.intro, rule VectorSpaceHom-idhom, unfold-locales*) *auto*

lemma *VectorSpaceEndSet-is-VectorSpaceHomSet* :
 $VectorSpaceHomSet\ smult\ V\ smult\ V = \{T. V\text{End}\ T\}$

proof

show $VectorSpaceHomSet\ smult\ V\ smult\ V \subseteq \{T. V\text{End}\ T\}$
using *VectorSpaceHomSetD-VectorSpaceHom VectorSpaceHomSetD-Im*
VectorSpaceEndI

by *fast*

show $VectorSpaceHomSet\ smult\ V\ smult\ V \supseteq \{T. V\text{End}\ T\}$
using *VectorSpaceEnd.VectorSpaceHom[of smult V]*
VectorSpaceEnd.endomorph[of smult V]
VectorSpaceHomSetI[of smult V smult - V]

by *fast*

qed

lemma *VectorSpace-VectorSpaceEndSet* : $VectorSpace\ end-smult\ VectorSpaceEndSet$

using *VectorSpace-axioms VectorSpace-VectorSpaceHomSet*
VectorSpaceEndSet-is-VectorSpaceHomSet

by *fastforce*

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* : $T \in VectorSpaceEndSet$
using *VectorSpaceEnd-axioms* **by** *fast*

lemma *end-smult-comp-comm-right* :
 $range\ S \subseteq V \implies T \circ (a \cdot S) = a \cdot (T \circ S)$
using *f-map* **by** *fastforce*

```

lemma VEnd-end-smult-VEnd : VEnd (a · T)
  using in-VEndSet VectorSpace.smult-closed[OF VectorSpace-VectorSpaceEndSet]
  by fast

lemma VEnd-composite-left :
  assumes VEnd S
  shows VEnd (S ∘ T)
  using endomorph VectorSpaceEnd.axioms(1)[OF assms] VectorSpaceHom-composite-left
    VectorSpaceEnd.endomorph[OF assms] VectorSpaceEndI[of smult V S ∘ T]
  by fastforce

lemma VEnd-composite-right : VEnd S  $\implies$  VEnd (T ∘ S)
  using VectorSpaceEnd-axioms VectorSpaceEnd.VEnd-composite-left by fast

end

lemma (in VectorSpace) inj-comp-end :
  assumes VEnd S inj-on S V VEnd T inj-on T V
  shows inj-on (S ∘ T) V
proof –
  have ker (S ∘ T) ∩ V ⊆ 0
  proof
    fix v assume v ∈ ker (S ∘ T) ∩ V
    moreover hence T v = 0 using kerD[of v S ∘ T]
    using VectorSpaceEnd.endomorph[OF assms(3)] kerI[of S]
      VectorSpaceEnd.inj-on-imp-Ker0[OF assms(1,2)]
    by auto
    ultimately show v ∈ 0
    using kerI[of T] VectorSpaceEnd.inj-on-imp-Ker0[OF assms(3,4)] by auto
  qed
  with assms(1,3) show ?thesis
    using VectorSpaceEnd.VEnd-composite-right VectorSpaceEnd.Ker0-imp-inj-on
    by fast
qed

lemma (in VectorSpace) n-inj-comp-end :
  [ VEnd S; VEnd T; ¬ inj-on (S ∘ T) V ]  $\implies$   $\neg inj-on S V \vee \neg inj-on T V$ 
  using inj-comp-end by fast

```

4.4.5 Polynomials of endomorphisms

```

context VectorSpaceEnd
begin

```

```

primrec endpow :: nat  $\Rightarrow$  ('v  $\Rightarrow$  'v)
  where endpow0: endpow 0 = id ↓ V
  | endpowSuc: endpow (Suc n) = T ∘ (endpow n)

```

```

definition polymap :: 'f poly  $\Rightarrow$  ('v  $\Rightarrow$  'v)

```

where $\text{polymap } p \equiv (\text{coeffs } p) \cdots (\text{map } \text{endpow } [0..<\text{Suc } (\text{degree } p)])$

lemma $\text{VEnd-endpow} : \text{VEnd } (\text{endpow } n)$
proof $(\text{induct } n)$
 case 0 **show** $?case$ **using** end-idhom **by** simp
next
 case $(\text{Suc } k)$
moreover have $\text{VEnd } T \dots$
ultimately have $\text{VEnd } (T \circ (\text{endpow } k))$ **using** $\text{VEnd-composite-right}$ **by** fast
moreover have $\text{endpow } (\text{Suc } k) = T \circ (\text{endpow } k)$ **by** simp
ultimately show $\text{VEnd } (\text{endpow } (\text{Suc } k))$ **by** simp
qed

lemma $\text{endpow-list-apply-closed} :$
 $v \in V \implies \text{set } (\text{map } (\lambda S. S v) (\text{map } \text{endpow } [0..<k])) \subseteq V$
using $\text{VEnd-endpow VectorSpaceEnd.endomorph}$ **by** fastforce

lemma $\text{map-endpow-Suc} :$
 $\text{map } \text{endpow } [0..<\text{Suc } n] = (\text{id} \downarrow V) \# \text{map } ((\circ) T) (\text{map } \text{endpow } [0..<n])$
proof $(\text{induct } n)$
 case $(\text{Suc } k)$
hence $\text{map } \text{endpow } [0..<\text{Suc } (\text{Suc } k)] = \text{id} \downarrow V$
 $\# \text{map } ((\circ) T) (\text{map } \text{endpow } [0..<k]) @ \text{map } ((\circ) T) [\text{endpow } k]$
 by auto
also have $\dots = \text{id} \downarrow V \# \text{map } ((\circ) T) (\text{map } \text{endpow } ([0..<\text{Suc } k]))$ **by** simp
finally show $?case$ **by** fast
qed simp

lemma $T\text{-endpow-list-apply-commute} :$
 $\text{map } T (\text{map } (\lambda S. S v) (\text{map } \text{endpow } [0..<n]))$
 $= \text{map } (\lambda S. S v) (\text{map } ((\circ) T) (\text{map } \text{endpow } [0..<n]))$
by $(\text{induct } n) \text{ auto}$

lemma $\text{polymap0} : \text{polymap } 0 = 0$
using $\text{polymap-def scalar-mult.lincomb-Nil}$ **by** force

lemma $\text{VEnd-polymap} : \text{VEnd } (\text{polymap } p)$
proof –
have $\text{set } (\text{map } \text{endpow } [0..<\text{Suc } (\text{degree } p)]) \subseteq \{S. \text{VEnd } S\}$
 using VEnd-endpow **by** auto
thus $?thesis$
 using $\text{polymap-def VectorSpace-VectorSpaceEndSet VectorSpace.lincomb-closed}$
 by fastforce
qed

lemma $\text{polymap-pCons} : \text{polymap } (p\text{Cons } a p) = a \cdot (\text{id} \downarrow V) + (T \circ (\text{polymap } p))$
proof cases
assume $p: p = 0$
show $?thesis$


```

proof cases
  assume  $a = 0$  with  $p$  show ?thesis
  using polymap0 VectorSpace-VectorSpaceEndSet VectorSpace.zero-smult end-idhom
    comp-zero
  by fastforce
next
  assume  $a: a \neq 0$ 
  define  $zmap$  where  $zmap = (0::'v \Rightarrow 'v)$ 
  from  $a$   $p$  have  $polymap (pCons a p) = a \cdot (endpow 0)$ 
    using polymap-def scalar-mult.lincomb-singles by simp
  moreover have  $a \cdot (id \downarrow V) + (T \circ (polymap p)) = a \cdot (id \downarrow V)$ 
    using  $p$  polymap0 comp-zero by simp
  ultimately show ?thesis by simp
qed
next
  assume  $p \neq 0$ 
  hence  $polymap (pCons a p)$ 
     $= (a \# (coeffs p)) \cdots (map endpow [0..<Suc (Suc (degree p))])$ 
    using polymap-def by simp
  also have  $\dots = (a \# (coeffs p))$ 
     $\cdots ((id \downarrow V) \# map ((\circ) T) (map endpow [0..<Suc (degree p)]))$ 
    using map-endpow-Suc[of Suc (degree p)] by fastforce
  also have  $\dots = a \cdot (id \downarrow V) + (coeffs p)$ 
     $\cdots (map ((\circ) T) (map endpow [0..<Suc (degree p)]))$ 
    using scalar-mult.lincomb-Cons by simp
  also have  $\dots = a \cdot (id \downarrow V) + (\sum (c,S)$ 
     $\leftarrow zip (coeffs p) (map ((\circ) T) (map endpow [0..<Suc (degree p)]))$ 
     $c \cdot S)$ 
    using scalar-mult.lincomb-def by simp
  finally have calc:
     $polymap (pCons a p) = a \cdot (id \downarrow V)$ 
     $+ (\sum (c,k) \leftarrow zip (coeffs p) [0..<Suc (degree p)]. c \cdot (T \circ (endpow k)))$ 
    using sum-list-prod-map2[
      of  $\lambda c S. c \cdot S$   $coeffs p (\circ) T$   $map endpow [0..<Suc (degree p)]$ 
    ]
    sum-list-prod-map2[
      of  $\lambda c S. c \cdot (T \circ S)$   $coeffs p endpow [0..<Suc (degree p)]$ 
    ]
    by simp
  show ?thesis
proof
  fix  $v$  show  $polymap (pCons a p) v = ((a \cdot (id \downarrow V)) + (T \circ (polymap p))) v$ 
  proof (cases v \in V)
    case True
    with calc
      have  $polymap (pCons a p) v = a \cdot v + (\sum (c,k)$ 
         $\leftarrow zip (coeffs p) [0..<Suc (degree p)]. c \cdot T (endpow k v))$ 
        using sum-list-prod-fun-apply[of \lambda c k. c \cdot (T \circ (endpow k))] by simp
      hence  $polymap (pCons a p) v = a \cdot v + (coeffs p) \cdot (map T$ 

```

```

      (map (λS. S v) (map endpow [0..<Suc (degree p)]))
using sum-list-prod-map2[
  of λc S. c · T (S v) coeffs p endpow [0..<Suc (degree p)]
]
sum-list-prod-map2[
  of λc u. c · T u coeffs p λS. S v map endpow [0..<Suc (degree p)]
]
sum-list-prod-map2[
  of λc u. c · u coeffs p T
  map (λS. S v) (map endpow [0..<Suc (degree p)])
]
lincomb-def
by simp
also from True
have ... = a · v + T ( coeffs p
  · (map (λS. S v) (map endpow [0..<Suc (degree p)])) )
using endpow-list-apply-closed[of v Suc (degree p)] distrib-lincomb
by simp
finally show ?thesis
using True lincomb-def
sum-list-prod-map2[
  of λc u. c · u coeffs p λS. S v map endpow [0..<Suc (degree p)]
]
sum-list-prod-fun-apply[of λc S. c · S] polymap-def
scalar-mult.lincomb-def[of end-smult]
by simp
next
case False
hence polymap (pCons a p) v = 0
using VEnd-polymap VectorSpaceEnd.supp suppI-contr by fast
moreover from False have ((a · (id↓V)) + (T ∘ (polymap p))) v = 0
using smult-zero VEnd-polymap[of p] VectorSpaceEnd.supp suppI-contr
im-zero
by fastforce
ultimately show ?thesis by simp
qed
qed
qed

```

```

lemma polymap-plus : polymap (p + q) = polymap p + polymap q
proof (induct p q rule: pCons-induct2)
case 00 show ?case using polymap0 by simp
case lpCons show ?case using polymap0 by simp
case rpCons show ?case using polymap0 by simp
next
case (pCons2 a p b q)
have polymap (pCons a p + pCons b q) = a · (id↓V) + b · (id↓V)
  + (T ∘ (polymap (p+q)))
using polymap-pCons end-idhom end-smult-distrib-right[OF VectorSpaceEnd.range]

```

```

    by simp
  also from pCons2(3)
    have ... = a .. (id↓V) + b .. (id↓V) + (T ∘ (polymap p + polymap q))
    by auto
  finally show ?case
    using pCons2(3) distrib-comp-sum-left[of polymap p polymap q] VEnd-polymap
      VectorSpaceEnd.range polymap-pCons
    by fastforce
qed

```

lemma *polymap-polysmult* : $\text{polymap } (\text{Polynomial.smult } a \ p) = a \cdot \text{polymap } p$

proof (*induct p*)

case 0 **show** $\text{polymap } (\text{Polynomial.smult } a \ 0) = a \cdot \text{polymap } 0$

using *polymap0 end-smult0* **by** *simp*

next

case (*pCons b p*)

hence $\text{polymap } (\text{Polynomial.smult } a \ (\text{pCons } b \ p))$

$= a \cdot b \cdot (\text{id}\downarrow V) + a \cdot (T \circ \text{polymap } p)$

using *polymap-pCons VectorSpaceEnd.range[OF VEnd-polymap]*

end-smult-comp-comm-right VectorSpaceEnd.range[OF end-idhom] end-smult-assoc

by *simp*

thus $\text{polymap } (\text{Polynomial.smult } a \ (\text{pCons } b \ p)) = a \cdot (\text{polymap } (\text{pCons } b \ p))$

using *VectorSpaceEnd.VEnd-end-smult-VEnd[OF end-idhom, of b]*

VEnd-composite-right[OF VEnd-polymap, of p]

end-smult-distrib-left[

OF VectorSpaceEnd.range VectorSpaceEnd.range,

of smult - smult T ∘ polymap p

]

polymap-pCons

by *simp*

qed

lemma *polymap-times* : $\text{polymap } (p * q) = (\text{polymap } p) \circ (\text{polymap } q)$

proof (*induct p*)

case 0 **show** ?case using *polymap0* **by** *auto*

next

case (*pCons a p*)

have $\text{polymap } (\text{pCons } a \ p * q) = a \cdot \text{polymap } q + (T \circ (\text{polymap } (p * q)))$

using *polymap-plus polymap-polysmult polymap-pCons end-idhom*

end-0smult[OF VectorSpaceEnd.range]

by *simp*

also from *pCons(2)*

have $\dots = a \cdot ((\text{id}\downarrow V) \circ \text{polymap } q) + (T \circ \text{polymap } p \circ \text{polymap } q)$

using *VectorSpaceEnd.endomorph[OF VEnd-polymap]*

VectorSpaceEnd.idhom-left[OF VEnd-polymap]

by *auto*

finally show $\text{polymap } (\text{pCons } a \ p * q) = (\text{polymap } (\text{pCons } a \ p)) \circ (\text{polymap } q)$

using *end-smult-comp-comm-left*

distrib-comp-sum-right[of a .. id ↓ V - polymap q]

```

      polymap-pCons
    by simp
  qed

lemma polymap-apply :
  assumes  $v \in V$ 
  shows  $\text{polymap } p \ v = (\text{coeffs } p) \cdot (\text{map } \text{endpow } [0..<\text{Suc } (\text{degree } p)])$ 
proof (induct p)
  case 0 show ?case
    using lincomb-Nil scalar-mult.lincomb-Nil[of - - end-smult] polymap-def
    by simp
  next
  case (pCons a p)
  show ?case
  proof (cases p = 0)
    case True
    moreover with pCons(1) have  $\text{polymap } (pCons \ a \ p) = a \cdot \text{id} \downarrow V$ 
      using polymap-pCons polymap0 comp-zero by simp
    ultimately show ?thesis using assms pCons(1) lincomb-singles by simp
  next
  case False
  from assms pCons(2)
  have  $\text{polymap } (pCons \ a \ p) \ v = a \cdot v + T (\text{coeffs } p \cdot \text{map } (\lambda S. S \ v) (\text{map } \text{endpow } [0..<\text{Suc } (\text{degree } p)]))$ 
    using polymap-pCons by simp
  with assms pCons(1)
  have 1:  $\text{polymap } (pCons \ a \ p) \ v = (\text{coeffs } (pCons \ a \ p)) \cdot (v \# \text{map } T (\text{map } (\lambda S. S \ v) (\text{map } \text{endpow } [0..<\text{Suc } (\text{degree } p)])))$ 
    using endpow-list-apply-closed[of v Suc (degree p)] distrib-lincomb lincomb-Cons
    by auto
  have 2:  $\text{map } T (\text{map } (\lambda S. S \ v) (\text{map } \text{endpow } [0..<\text{Suc } (\text{degree } p)])) = \text{map } (\lambda S. S \ v) (\text{map } ((\circ) \ T) (\text{map } \text{endpow } [0..<\text{Suc } (\text{degree } p)]))$ 
    using T-endpow-list-apply-commute[of v Suc (degree p)] by simp
  from 1 2
  have  $\text{polymap } (pCons \ a \ p) \ v = (\text{coeffs } (pCons \ a \ p)) \cdot (v \# \text{map } (\lambda S. S \ v) (\text{map } ((\circ) \ T) (\text{map } \text{endpow } [0..<\text{Suc } (\text{degree } p)])))$ 
    using subst[
      OF 2, of  $\lambda x. \text{polymap } (pCons \ a \ p) \ v = (\text{coeffs } (pCons \ a \ p)) \cdot (v \# x)$ 
    ]
    by simp
  with assms
  have 3:  $\text{polymap } (pCons \ a \ p) \ v = (\text{coeffs } (pCons \ a \ p)) \cdot (\text{map } (\lambda S. S \ v) (\text{id} \downarrow V \# \text{map } ((\circ) \ T) (\text{map } \text{endpow } [0..<\text{Suc } (\text{degree } p)])))$ 
    by simp
  from False pCons(1)
  have 4:  $\text{id} \downarrow V \# \text{map } ((\circ) \ T) (\text{map } \text{endpow } [0..<\text{Suc } (\text{degree } p)]) = \text{map } \text{endpow } [0..<\text{Suc } (\text{degree } (pCons \ a \ p))]$ 

```

```

    using map-endpow-Suc[of Suc (degree p), THEN sym]
  by simp
from 3 show ?thesis
  using subst[
    OF 4,
    of  $\lambda x. \text{polymap } (p\text{Cons } a \ p) \ v$ 
      = (coeffs (pCons a p))  $\cdot$  (map ( $\lambda S. S \ v$ ) x)
  ]
  by simp
qed
qed

```

lemma *polymap-apply-linear* : $v \in V \implies \text{polymap } [:-c, 1:] \ v = T \ v - c \cdot v$
 using *polymap-apply lincomb-def neg-smult endomorph* **by** *auto*

lemma *polymap-const-inj* :
 assumes *degree p = 0 p \neq 0*
 shows *inj-on (polymap p) V*
proof (*rule inj-onI*)
 fix $u \ v$ **assume** $uv: u \in V \ v \in V \ \text{polymap } p \ u = \text{polymap } p \ v$
 from *assms* **have** $p: \text{coeffs } p = [\text{coeff } p \ 0]$ **unfolding** *coeffs-def* **by** *simp*
 from *uv assms* **have** $(\text{coeff } p \ 0) \cdot u = (\text{coeff } p \ 0) \cdot v$
 using *polymap-apply lincomb-singles* **unfolding** *coeffs-def* **by** *simp*
 with *assms uv(1,2)* **show** $u = v$
 using *const-poly-nonzero-coeff cancel-scalar* **by** *auto*
qed

lemma *n-inj-polymap-times* :
 $\neg \text{inj-on } (\text{polymap } (p * q)) \ V$
 $\implies \neg \text{inj-on } (\text{polymap } p) \ V \vee \neg \text{inj-on } (\text{polymap } q) \ V$
 using *polymap-times VEnd-polymap n-inj-comp-end* **by** *fastforce*

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 (*induct n \equiv degree p arbitrary: p*)
 case ($0 \ p$) **thus** *?case* **using** *polymap-const-inj* **by** *simp*
next
 case (*Suc n p*)
 from *Suc(2) alg-closed* **obtain** c **where** $c: \text{poly } p \ c = 0$ **by** *fastforce*
define q **where** $q = \text{synthetic-div } p \ c$
 with c **have** *p-decomp: $p = [:-c, 1:] * q$*
 using *synthetic-div-correct'[of c p]* **by** *simp*
show *?case*
proof (*cases inj-on (polymap q) V*)
 case *True* with *Suc(4)* **show** *?thesis*
 using *p-decomp n-inj-polymap-times* **by** *fast*

```

next
  case False
  then have  $n = \text{degree } q$ 
    using degree-synthetic-div [of p c] q-def <Suc n = degree p>
    by auto
  moreover have  $q \neq 0$ 
    using <p ≠ 0> p-decomp
    by auto
  ultimately show ?thesis
    using False
    by (rule Suc.hyps)
qed
qed
end

```

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

```

lemma (in FinDim VectorSpace) endomorph-has-eigenvector :
  assumes alg-closed:  $\bigwedge p::'a \text{ poly. degree } p > 0 \implies \exists c. \text{poly } p \ c = 0$ 
  and dim :  $\text{dim } V > 0$ 
  and endo : VectorSpaceEnd smult V T
  shows  $\exists c \ u. u \in V \wedge u \neq 0 \wedge T \ u = c \cdot u$ 
proof -
  define Tpolymap where Tpolymap = VectorSpaceEnd.polymap smult V T
  from dim obtain v where  $v \in V \ v \neq 0$ 
    using dim-nonzero nonempty by auto
  define Tpows where Tpows = map (VectorSpaceEnd.endpow V T) [0..<Suc (dim V)]
  define Tpows-v where Tpows-v = map (λS. S v) Tpows
  with endo Tpows-def v(1) have Tpows-v-V:  $\text{set } Tpows-v \subseteq V$ 
    using VectorSpaceEnd.endpow-list-apply-closed by fast
  moreover from Tpows-v-def Tpows-def Tpows-v-V have  $\neg \text{lin-independent } Tpows-v$ 
    using too-long-lin-dependent by simp
  ultimately obtain as
    where as:  $\text{set } as \neq \{0\} \ \text{length } as = \text{length } Tpows-v \ \text{as} \cdot Tpows-v = 0$ 
    using lin-dependent-dependence-relation
    by fast
  define p where p = Poly as
  with dim Tpows-def Tpows-v-def as(1,2) have p-n0:  $p \neq 0$ 
    using nonzero-coeffs-nonzero-poly[of as] by fastforce
  define Tpows' where Tpows' = map (VectorSpaceEnd.endpow V T) [0..<Suc (degree p)]
  define Tpows-v' where Tpows-v' = map (λS. S v) Tpows'
  have Tpows-v' = take (Suc (degree p)) Tpows
proof -
  from Tpows-def
  have 1:  $\text{take } (Suc \ (\text{degree } p)) \ Tpows = \text{map } (VectorSpaceEnd.endpow \ V \ T)$ 

```

```

      (take (Suc (degree p)) [0..Suc (dim V)])
    using take-map[of - - [0..Suc (dim V)]]
    by simp
  from p-n0 p-def as(2) Tpows-v-def Tpows-def
  have 2: take (Suc (degree p)) [0..Suc (dim V)] = [0..Suc (degree p)]
  using length-coeffs-degree[of p] length-strip-while-le[of (=) 0 as]
    take-upt[of 0 Suc (degree p) Suc (dim V)]
  by simp
  from 1 Tpows'-def have take (Suc (degree p)) Tpows = Tpows'
  using subst[OF 2] by fast
  thus ?thesis by simp
qed
with Tpows-v-def Tpows-v'-def have Tpows-v' = take (Suc (degree p)) Tpows-v
  using take-map[of -  $\lambda S. S v Tpows$ ] by simp
moreover from p-def Tpows-v-V as(3) Tpows-v'-def have (coeffs p) · Tpows-v
= 0
  using lincomb-strip-while-0coeffs by simp
ultimately have (coeffs p) · Tpows-v' = 0
  using p-n0 lincomb-conv-take-right[of coeffs p] length-coeffs-degree[of p] by simp
with Tpolymap-def v(1) Tpows-v'-def Tpows'-def have Tpolymap p v = 0
  using VectorSpaceEnd.polymap-apply[OF endo] by simp
with alg-closed Tpolymap-def v endo p-n0 obtain c
  where  $\neg$  inj-on (Tpolymap [-c, 1:]) V
  using VectorSpaceEnd.VEnd-polymap VectorSpaceEnd.nonzero-Ker-el-imp-n-inj
    VectorSpaceEnd.n-inj-polymap-findlinear[OF endo]
  by fastforce
with Tpolymap-def have (GroupHom.Ker V (Tpolymap [-c, 1:])) - 0  $\neq$  {}
  using VectorSpaceEnd.VEnd-polymap[OF endo] VectorSpaceEnd.Ker0-imp-inj-on
  by fast
from this obtain u where u  $\in$  V Tpolymap [-c, 1:] u = 0 u  $\neq$  0
  using kerD by fastforce
with Tpolymap-def show ?thesis
  using VectorSpaceEnd.polymap-apply-linear[OF endo] by auto
qed

```

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 · 70)

begin

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

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

where *as* $\#$ · *vs* \equiv scalar-mult.lincomb fsmult *as vs*

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

where f-lin-independent \equiv scalar-mult.lin-independent fsmult

abbreviation $fSpan :: 'v list \Rightarrow 'v set$ **where** $fSpan \equiv scalar-mult.Span fsmult$
definition $Gmult :: 'g \Rightarrow 'v \Rightarrow 'v$ (**infixr** $*$ 70) **where** $g * v \equiv (1 \ \delta\delta \ g) \cdot v$

lemmas $R-scalar-mult = R-scalar-mult$

lemma $fsmultD : a \# \cdot v = (a \ \delta\delta \ 0) \cdot v$
unfolding $fsmult-def$ **by** $fast$

lemma $GmultD : g * v = (1 \ \delta\delta \ g) \cdot v$
unfolding $Gmult-def$ **by** $fast$

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

lemma $negGorbit-Cons :$
 $negGorbit-list \ (g\#gs) \ T \ as$
 $= (map \ (Gmult \ (-g) \circ T) \ as) \# negGorbit-list \ gs \ T \ as$
using $negGorbit-list-def[of \ - \ T \ as]$ **by** $simp$

lemma $length-negGorbit-list : length \ (negGorbit-list \ gs \ T \ as) = length \ gs$
using $negGorbit-list-def[of \ gs \ T]$ **by** $simp$

lemma $length-negGorbit-list-sublist :$
 $fs \in set \ (negGorbit-list \ gs \ T \ as) \Longrightarrow length \ fs = length \ as$
using $negGorbit-list-def[of \ gs \ T]$ **by** $auto$

lemma $length-concat-negGorbit-list :$
 $length \ (concat \ (negGorbit-list \ gs \ T \ as)) = (length \ gs) * (length \ as)$
using $length-concat[of \ negGorbit-list \ gs \ T \ as]$
 $length-negGorbit-list-sublist[of \ - \ gs \ T \ as]$
 $const-sum-list[of \ negGorbit-list \ gs \ T \ as \ length \ length \ as] \ length-negGorbit-list$
by $auto$

lemma $negGorbit-list-nth :$
 $\bigwedge i. i < length \ gs \Longrightarrow (negGorbit-list \ gs \ T \ as)!i = map \ (Gmult \ (-gs!i) \circ T) \ as$
proof ($induct \ gs$)
case ($Cons \ g \ gs$) **thus** $?case$ **using** $negGorbit-Cons[of \ - \ - \ T]$ **by** ($cases \ i$) $auto$
qed $simp$

end

5.2 Locale and basic facts

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

sublocale $FGModule < aezfun\text{-}scalar\text{-}mult$ **proof**– **qed**

lemma (in $Group$) $trivial\text{-}FGModule$:
fixes $smult :: ('f::field, 'g) aezfun \Rightarrow 'v::ab\text{-}group\text{-}add \Rightarrow 'v$
assumes $smult\text{-}zero: \forall a \in group\text{-}ring. smult\ a\ (0::'v) = 0$
shows $FGModule\ G\ smult\ (0::'v\ set)$
proof (rule $FGModule.intro$)
from $assms$ **show** $RModule\ group\text{-}ring\ smult\ 0$
using $Ring1\text{-}RG\ trivial\text{-}RModule$ **by** $fast$
qed ($unfold\text{-}locales$)

context $FGModule$
begin

abbreviation $FG :: ('f, 'g) aezfun\ set$ **where** $FG \equiv ActingGroup.group\text{-}ring$
abbreviation $FGSubmodule \equiv RSubmodule$
abbreviation $FG\text{-}proj \equiv ActingGroup.RG\text{-}proj$

lemma $GroupG: Group\ G ..$

lemmas $zero\text{-}closed = zero\text{-}closed$
lemmas $neg\text{-}closed = neg\text{-}closed$
lemmas $diff\text{-}closed = diff\text{-}closed$
lemmas $zero\text{-}smult = zero\text{-}smult$
lemmas $smult\text{-}zero = smult\text{-}zero$
lemmas $AbGroup = AbGroup$
lemmas $sum\text{-}closed = AbGroup.sum\text{-}closed[OF\ AbGroup]$
lemmas $FGSubmoduleI = RSubmoduleI$
lemmas $FG\text{-}proj\text{-}mult\text{-}leftdelta = ActingGroup.RG\text{-}proj\text{-}mult\text{-}leftdelta$
lemmas $FG\text{-}proj\text{-}mult\text{-}right = ActingGroup.RG\text{-}proj\text{-}mult\text{-}right$
lemmas $FG\text{-}el\text{-}decomp = ActingGroup.RG\text{-}el\text{-}decomp\text{-}aezdeltafun$

lemma $FG\text{-}n0: FG \neq 0$ **using** $ActingGroup.RG\text{-}n0$ **by** $fast$

lemma $FG\text{-}proj\text{-}in\text{-}FG : FG\text{-}proj\ x \in FG$
using $ActingGroup.RG\text{-}proj\text{-}in\text{-}RG$ **by** $fast$

lemma $FG\text{-}fddg\text{-}closed : g \in G \Longrightarrow a\ \delta\delta\ g \in FG$
using $ActingGroup.RG\text{-}aezdeltafun\text{-}closed$ **by** $fast$

lemma $FG\text{-}fdd0\text{-}closed : a\ \delta\delta\ 0 \in FG$
using $ActingGroup.RG\text{-}aezdelta0fun\text{-}closed$ **by** $fast$

lemma $Gmult\text{-}closed : g \in G \Longrightarrow v \in V \Longrightarrow g * \cdot v \in V$
using $FG\text{-}fddg\text{-}closed\ smult\text{-}closed\ GmultD$ **by** $simp$

lemma $map\text{-}Gmult\text{-}closed :$
 $g \in G \Longrightarrow set\ vs \subseteq V \Longrightarrow set\ (map\ ((*\cdot)\ g)\ vs) \subseteq V$
using $Gmult\text{-}def\ FG\text{-}fddg\text{-}closed\ map\text{-}smult\text{-}closed[of\ 1\ \delta\delta\ g\ vs]$ **by** $auto$

lemma *Gmult0* :

assumes $v \in V$

shows $0 * v = v$

proof –

have $0 * v = (1 \delta\delta 0) \cdot v$ **using** *GmultD* **by** *fast*

moreover have $1 \delta\delta 0 = (1::('f,'g) \text{ aezfun})$ **using** *one-aezfun-transfer* **by** *fast*

ultimately have $0 * v = (1::('f,'g) \text{ aezfun}) \cdot v$ **by** *simp*

with *assms* **show** *?thesis* **using** *one-smult* **by** *simp*

qed

lemma *Gmult-assoc* :

assumes $g \in G \ h \in G \ v \in V$

shows $g * h * v = (g + h) * v$

proof –

define n **where** $n = (1::'f)$

with *assms* **have** $g * h * v = ((n \delta\delta g) * (n \delta\delta h)) \cdot v$

using *FG-fddg-closed* *GmultD* **by** *simp*

moreover from *n-def* **have** $n \delta\delta g * (n \delta\delta h) = n \delta\delta (g + h)$

using *times-aezdeltafun-aezdeltafun[of n g n h]* **by** *simp*

ultimately show *?thesis* **using** *n-def* *GmultD* **by** *simp*

qed

lemma *Gmult-distrib-left* :

$\llbracket g \in G; v \in V; v' \in V \rrbracket \implies g * (v + v') = g * v + g * v'$

using *GmultD* *FG-fddg-closed* **by** *simp*

lemma *neg-Gmult* : $g \in G \implies v \in V \implies g * (-v) = -(g * v)$

using *GmultD* *FG-fddg-closed* *smult-neg* **by** *simp*

lemma *Gmult-neg-left* : $g \in G \implies v \in V \implies (-g) * g * v = v$

using *ActingGroup.neg-closed* *Gmult-assoc*[of $-g$ g] *Gmult0* **by** *simp*

lemma *fddg-smult-decomp* : $g \in G \implies v \in V \implies (f \delta\delta g) \cdot v = f \sharp \cdot g * v$

using *aezdeltafun-decomp*[of f g] *FG-fddg-closed* *FG-fdd0-closed* *GmultD*
fsmult-def

by *simp*

lemma *sum-list-aezdeltafun-smult-distrib* :

assumes $v \in V$ *set* $(\text{map } \text{snd } \text{fgs}) \subseteq G$

shows $(\sum (f,g) \leftarrow \text{fgs}. f \delta\delta g) \cdot v = (\sum (f,g) \leftarrow \text{fgs}. f \sharp \cdot g * v)$

proof –

from *assms*(2) **have** *set* $(\text{map } (\text{case-prod } \text{aezdeltafun}) \text{ fgs}) \subseteq FG$

using *FG-fddg-closed* **by** *auto*

with *assms*(1) **have** $(\sum (f,g) \leftarrow \text{fgs}. f \delta\delta g) \cdot v = (\sum (f,g) \leftarrow \text{fgs}. (f \delta\delta g) \cdot v)$

using *sum-list-prod-map-smult-distrib* **by** *auto*

also have $\dots = (\sum (f,g) \leftarrow \text{fgs}. f \sharp \cdot g * v)$

using *assms* *fddg-smult-decomp*

sum-list-prod-cong[of fgs $\lambda f g. (f \delta\delta g) \cdot v$ $\lambda f g. f \sharp \cdot g * v$]

by *fastforce*
 finally show *?thesis* by *fast*
 qed

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*
 using *FGModule.axioms(2)* by *fast*

lemma *GSubspace-is-FGModule* :
 assumes *GSubspace U*
 shows *FGModule G smult U*
 proof (rule *FGModule.intro*, rule *GroupG*)
 from *assms* show *RModule FG* (\cdot) *U* by *fast*
 qed (*unfold-locales*)

lemma *restriction-to-subgroup-is-module* :
 fixes *H* :: 'g set
 assumes *subgrp*: *Group.Subgroup G H*
 shows *FGModule H smult V*
 proof (rule *FGModule.intro*)
 from *subgrp* show *Group H* by *fast*
 from *assms* show *RModule* (*Group.group-ring H*) (\cdot) *V*
 using *ActingGroup.Subgroup-imp-Subring SModule-restrict-scalars* by *fast*
 qed

lemma *negGorbit-list-V* :
 assumes *set gs* \subseteq *G* *T* ' (*set as*) \subseteq *V*
 shows *set* (*concat* (*negGorbit-list gs T as*)) \subseteq *V*
 proof –
 from *assms(2)*
 have *set* (*concat* (*negGorbit-list gs T as*)) \subseteq ($\bigcup_{g \in \text{set } gs. (Gmult (-g))$ ' *V*)
 using *set-concat negGorbit-list-def*[*of gs T as*]
 by *force*
 moreover from *assms(1)* have $\bigwedge g. g \in \text{set } gs \implies (Gmult (-g))$ ' *V* \subseteq *V*
 using *ActingGroup.neg-closed Gmult-closed* by *fast*
 ultimately show *?thesis* by *fast*
 qed

lemma *negGorbit-list-Cons0* :
 T ' (*set as*) \subseteq *V*
 \implies *negGorbit-list* (*0* # *gs*) *T as* = (*map T as*) # (*negGorbit-list gs T as*)
 using *negGorbit-Cons*[*of 0 gs T as*] *Gmult0* by *auto*

end

5.3 Modules over a group ring as a vector spaces

context *FGModule*

begin

lemma *fVectorSpace* : *VectorSpace fsmult V*

proof (rule *VectorSpaceI, unfold-locales*)

fix *a* :: '*f* show $\bigwedge v. v \in V \implies a \# \cdot v \in V$

using *fsmult-def smult-closed FG-fdd0-closed* by *simp*

next

fix *a* :: '*f* show $\bigwedge u v. u \in V \implies v \in V \implies a \# \cdot (u + v) = a \# \cdot u + a \# \cdot v$

using *fsmult-def FG-fdd0-closed* by *simp*

next

fix *a b* :: '*f* and *v* :: '*v* assume *v*: *v* ∈ *V*

have $(a+b) \# \cdot v = (a \ \delta\delta \ 0 + b \ \delta\delta \ 0) \cdot v$

using *aezdeltafun-plus[of a b 0] arg-cong[of - - λr. r · v] fsmult-def* by *fastforce*

with *v* show $(a+b) \# \cdot v = a \# \cdot v + b \# \cdot v$

using *fsmult-def FG-fdd0-closed* by *simp*

next

fix *a b* :: '*f* show $\bigwedge v. v \in V \implies a \# \cdot (b \# \cdot v) = (a * b) \# \cdot v$

using *times-aezdeltafun-aezdeltafun[of a 0 b 0] arg-cong fsmult-def FG-fdd0-closed*

by *simp*

next

fix *v* :: '*v* assume *v* ∈ *V* thus $1 \# \cdot v = v$

using *one-aezfun-transfer arg-cong[of 1 δδ 0 1 λa. a · v] fsmult-def* by *fastforce*

qed

abbreviation *fSubspace* ≡ *VectorSpace.Subspace fsmult V*

abbreviation *fbasis-for* ≡ *fscalar-mult.basis-for fsmult*

abbreviation *fdim* ≡ *scalar-mult.dim fsmult V*

lemma *VectorSpace-fSubspace* : *fSubspace W* ⇒ *VectorSpace fsmult W*

using *Module.intro VectorSpace.intro* by *fast*

lemma *fsmult-closed* : *v* ∈ *V* ⇒ *a* # · *v* ∈ *V*

using *FG-fdd0-closed smult-closed fsmult-def* by *simp*

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* : $r \in FG \implies v \in V \implies a \cdot r \cdot v = r \cdot a \cdot v$
using *fsmultD FG-fdd0-closed smult-assoc aezdeta0fun-commutes*[*of r*] **by** *simp*

lemma *fsmult-Gmult-comm* : $g \in G \implies v \in V \implies a \cdot g \cdot v = g \cdot a \cdot v$
using *aezdeltafun-decomp*[*of a g*] *aezdeltafun-decomp'*[*of a g*] *FG-fddg-closed*
FG-fdd0-closed fsmult-def GmultD
by *simp*

lemma *Gmult-flincomb-comm* :
assumes $g \in G$ *set vs* $\subseteq V$
shows $g \cdot \text{as} \cdot \text{vs} = \text{as} \cdot (Gmult\ g)\ \text{vs}$

proof –

have $g \cdot \text{as} \cdot \text{vs} = (1\ \delta\delta\ g) \cdot (\sum (a,v) \leftarrow \text{zip as vs}. a \cdot v)$
using *Gmult-def scalar-mult.lincomb-def*[*of fsmult*] **by** *simp*

with *assms* **have** $g \cdot \text{as} \cdot \text{vs}$

= *sum-list* (*map* ($(\cdot)\ (1\ \delta\delta\ g) \circ (\lambda(x,y). x \cdot y)$) (*zip as vs*))

using *set-zip-rightD fsmult-closed FG-fddg-closed*[*of g 1::'f*]
smult-sum-list-distrib[*of 1 \delta\delta g map (case-prod (\cdot)) (zip as vs)*]
map-map[*of (\cdot)\ (1\ \delta\delta\ g) case-prod (\cdot) zip as vs*]

by *fastforce*

moreover **have** $(\cdot)\ (1\ \delta\delta\ g) \circ (\lambda(x,y). x \cdot y) = (\lambda(x,y). (1\ \delta\delta\ g) \cdot (x \cdot y))$

by *auto*

ultimately **have** $g \cdot \text{as} \cdot \text{vs} = \text{sum-list}\ (\text{map}\ (\lambda(x,y). g \cdot x \cdot y)\ (\text{zip as vs}))$

using *Gmult-def* **by** *simp*

moreover **from** *assms* **have** $\forall (x,y) \in \text{set}\ (\text{zip as vs}). g \cdot x \cdot y = x \cdot g \cdot y$

using *set-zip-rightD fsmult-Gmult-comm* **by** *fastforce*

ultimately **have** $g \cdot \text{as} \cdot \text{vs}$

= *sum-list* (*map* ($\lambda(x,y). x \cdot y$) (*zip as (map (Gmult g) vs)*))

using *sum-list-prod-cong sum-list-prod-map2*[*of \lambda x y. x \cdot y as Gmult g*]

by *force*

thus *?thesis* **using** *scalar-mult.lincomb-def*[*of fsmult*] **by** *simp*

qed

lemma *GSubspace-is-Subspace* :

GSubspace U \implies *VectorSpace.Subspace fsmult V U*

using *GSubspace-is-FGModule FGModule.fVectorSpace VectorSpace.axioms*
Module.axioms

by *fast*

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 => 'v::ab-group-add => 'v (infixr · 70)
and V      :: 'v set
and smult' :: ('f, 'g) aezfun => 'w::ab-group-add => 'w (infixr ★ 70)
and T      :: 'v => 'w

```

sublocale *FGModuleHom* < *FGModule* ..

```

lemma (in FGModule) FGModuleHomI-fromaxioms :
  assumes  $\bigwedge v v'. v \in V \implies v' \in V \implies T (v + v') = T v + T v'$ 
          $\text{supp } T \subseteq V \bigwedge r m. r \in FG \implies m \in V \implies T (r \cdot m) = \text{smult}' r (T m)$ 
  shows  FGModuleHom G smult V smult' T
  using  assms
  by     unfold-locales

```

```

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 => 'v::ab-group-add => 'v (infixr · 70)
  and V      :: 'v set
  and T      :: 'v => '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 => 'v::ab-group-add => 'v (infixr · 70)
  and V      :: 'v set
  and smult' :: ('f, 'g) aezfun => 'w::ab-group-add => 'w (infixr ★ 70)
  and T      :: 'v => 'w

```

+ **fixes** *W* :: 'w set
assumes *bijjective*: *bij-betw* T V W

```

abbreviation (in FGModule) isomorphic ::
  (('f, 'g) aezfun => 'w::ab-group-add => 'w) => 'w set => 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  $\bigwedge r. r \in FG \implies \text{smult}' r 0 = 0$ 
  shows  FGModuleHom G smult V smult' 0
proof (rule FGModuleHom.intro)
  from  assms show RModuleHom FG (·) V smult' 0
        using trivial-RModuleHom by auto
qed (unfold-locales)

```

lemma *FGModHom-idhom* : *FGModuleHom* *G smult V smult* (*id*↓*V*)
proof (*rule FGModuleHom.intro*)
 show *RModuleHom FG smult V smult* (*id*↓*V*) **using** *RModHom-idhom* **by fast**
qed (*unfold-locales*)

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

show $\bigwedge v v'. v \in V \implies v' \in V \implies T (v + v') = T v + T v'$
using *VectorSpaceHom.GroupHom[OF hom]* *GroupHom.hom* **by auto**

from *hom* **show** *supp T* ⊆ *V* **using** *VectorSpaceHom.supp* **by fast**

show $\bigwedge r v. r \in FG \implies v \in V \implies T (r \cdot v) = r * T v$

proof–

fix *r v* **assume** *r*: *r* ∈ *FG* **and** *v*: *v* ∈ *V*

from *r* **obtain** *fgs*

where *fgs*: *set* (*map snd fgs*) ⊆ *G* *r* = ($\sum (f,g) \leftarrow fgs. f \delta \delta g$)
using *FG-el-decomp*

by fast

from *fgs v* **have** $r \cdot v = (\sum (f,g) \leftarrow fgs. f \# \cdot g * \cdot v)$

using *sum-list-aezdeltafun-smult-distrib* **by simp**

moreover from *v fgs(1)* **have** *set* (*map* ($\lambda(f,g). f \# \cdot g * \cdot v$) *fgs*) ⊆ *V*

using *Gmult-closed fsmult-closed* **by auto**

ultimately have $T (r \cdot v) = (\sum (f,g) \leftarrow fgs. T (f \# \cdot g * \cdot v))$

using *hom VectorSpaceHom.im-sum-list-prod* **by auto**

moreover from *hom G-map fgs(1) v*

have $\forall (f,g) \in \text{set } fgs. T (f \# \cdot g * \cdot v) = f \# * g ** T v$

using *Gmult-closed VectorSpaceHom.f-map[of fsmult V fsmult' T]*

by auto

ultimately have $T (r \cdot v) = (\sum (f,g) \leftarrow fgs. f \# * g ** T v)$

using *sum-list-prod-cong* **by simp**

with *v fgs fsmult' Gmult' Im-W(2)* **show** $T (r \cdot v) = r * (T v)$

using *FGModule.sum-list-aezdeltafun-smult-distrib[OF Im-W(1)]* **by auto**

qed

qed

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 : ⋀ g v. g ∈ G ⇒ v ∈ set vs ⇒ T (g *· v) = g ** (T v)
shows FGModuleHom G smult V smult' T
proof (rule VecHom-GMap-is-FGModuleHom)
  from fsmult' hom
    show VectorSpaceHom (#·) V (aezfun-scalar-mult.fsmult (★)) T
  by fast
next
fix g v assume g: g ∈ G and v: v ∈ V
from v fbasis obtain cs where cs: v = cs ·#· vs
  using VectorSpace.in-Span-obtain-same-length-coeffs[OF fVectorSpace] by fast
with g(1) fbasis fsmult' flincomb'
  have T (g *· v) = cs ·#* (map (T ∘ (Gmult g)) vs)
  using Gmult-flincomb-comm map-Gmult-closed
    VectorSpaceHom.distrib-lincomb[OF hom]
  by auto
moreover have T ∘ (Gmult g) = (λv. T (g *· v)) by auto
ultimately have T (g *· v) = cs ·#* (map (λv. g ** (T v)) vs)
  using fbasis g(1) G-map map-cong[of vs vs λv. T (g *· v)]
  by simp
moreover have (λv. g ** (T v)) = (Gmult' g) ∘ T by auto
ultimately have T (g *· v) = g ** cs ·#* (map T vs)
  using g(1) fbasis Im-W(2) Gmult' flincomb'
    FGModule.Gmult-flincomb-comm[OF Im-W(1), of g map T vs]
  by fastforce
thus T (g *· v) = aezfun-scalar-mult.Gmult (★) g (T v)
  using fbasis fsmult' Gmult' flincomb' cs
    VectorSpaceHom.distrib-lincomb[OF hom]
  by auto
qed (rule Im-W(1), rule Im-W(2))

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-imp-imp-diff-in-Ker* = *eq-imp-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* : $v \in V \implies T (a \# \cdot v) = a \# \star T v$
using *fsmultD ActingGroup.RG-aezdelta0fun-closed[of a] FG-map fsmultD'*
by *simp*

lemma *G-map* : $g \in G \implies v \in V \implies T (g * \cdot v) = g * \star T v$
using *GmultD ActingGroup.RG-aezdeltafun-closed[of g 1] FG-map GmultD'*
by *simp*

lemma *VectorSpaceHom* : *VectorSpaceHom fsmult V fsmult' T*
by (
rule VectorSpace.VectorSpaceHomI, rule fVectorSpace, unfold-locales,
rule f-map
)

lemmas *distrib-flincomb* = *VectorSpaceHom.distrib-lincomb[OF VectorSpaceHom]*

lemma *FGModule-Im* : *FGModule G smult' ImG*
by (*rule FGModule.intro, rule GroupG, rule RModule-Im, unfold-locales*)

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 (*rule FGModuleHom.intro*)
from *assms(2) show RModuleHom FG smult V smult'' (S \circ T)*
using *FGModuleHom.axioms(2)[OF assms(1)] RModHom-composite-left[of W]*
by *fast*
qed (*rule GroupG, unfold-locales*)

lemma *restriction-to-subgroup-is-hom* :
fixes *H :: 'g set*
assumes *subgrp: Group.Subgroup G H*
shows *FGModuleHom H smult V smult' T*
proof (*rule FGModule.FGModuleHomI-fromaxioms*)
have *FGModule G smult V ..*
with *assms show FGModule H (\cdot) V*
using *FGModule.restriction-to-subgroup-is-module by fast*

```

from supp show supp  $T \subseteq V$  by fast
from assms
  show  $\bigwedge r m. \llbracket r \in (\text{Group.group-ring } H); m \in V \rrbracket \implies T (r \cdot m) = r \star T m$ 
  using FG-map ActingGroup.Subgroup-imp-Subring by fast
qed (rule hom)

```

```

lemma FGModuleHom-restrict0-GSubspace :
  assumes GSubspace U
  shows FGModuleHom G smult U smult' (T ↓ U)
proof (rule FGModuleHom.intro)
  from assms show RModuleHom FG (·) U (★) (T ↓ U)
  using RModuleHom-restrict0-submodule by fast
qed (unfold-locales)

```

```

lemma FGModuleHom-fscalar-mul :
  FGModuleHom G smult V smult' (λv. a #★ T v)
proof
  have vsphom: VectorSpaceHom fsmult V fsmult' (λv. a #★ T v)
  using VectorSpaceHom.VectorSpaceHom-scalar-mul[OF VectorSpaceHom]
  by fast
  thus  $\bigwedge v v'. v \in V \implies v' \in V \implies a \#★ T (v + v') = a \#★ T v + a \#★ T v'$ 
  using VectorSpaceHom.additive[of fsmult V] by auto
  from vsphom show supp (λv. a #★ T v) ⊆ V
  using VectorSpaceHom.supp by fast
next
  fix r v assume rv: r ∈ FG v ∈ V
  thus  $a \#★ T (r \cdot v) = r \star a \#★ T v$ 
  using FG-map FGModule.fsmult-smult-comm[OF FGModule-Im]
  by auto
qed

```

end

```

lemma GSubspace-eigenspace :
  fixes e :: 'f::field
  and E :: 'v::ab-group-add set
  and smult :: ('f::field, 'g::group-add) aefun  $\Rightarrow$  'v  $\Rightarrow$  'v (infixr · 70)
  assumes FGModHom: FGModuleHom G smult V smult T
  defines E :  $E \equiv \{v \in V. T v = aefun\text{-scalar-mult.fsmult } smult e v\}$ 
  shows FGModule.GSubspace G smult V E
proof –
  have FGModule.GSubspace G smult V {v ∈ V. T v = (e δδ 0) · v}
  using FGModuleHom.axioms(2)[OF FGModHom]
  proof (rule RSubmodule-eigenspace)
  show  $e \delta\delta 0 \in \text{FGModule.FG } G$ 
  using FGModuleHom.FG-fdd0-closed[OF FGModHom] by fast
  show  $\bigwedge s v. s \in \text{FGModule.FG } G \implies v \in V \implies s \cdot (e \delta\delta 0) \cdot v = (e \delta\delta 0) \cdot s \cdot v$ 
  using FGModuleHom.fsmult-smult-domain-comm[OF FGModHom]

```

by *aezfun-scalar-mult.fsmultD[of smult]*
 by *simp*
 qed
 with *E* show *?thesis* using *aezfun-scalar-mult.fsmultD[of smult]* by *simp*
 qed

5.4.3 Basic facts about endomorphisms

lemma *RModuleEnd-over-group-ring-is-FGModuleEnd* :
 fixes *G* :: '*g*::group-add set
 and *smult* :: ('*f*::field, '*g*) *aezfun* \Rightarrow '*v*::ab-group-add \Rightarrow '*v*
 assumes *G* : *Group* *G* and *endo*: *RModuleEnd* (*Group.group-ring* *G*) *smult* *V* *T*
 shows *FGModuleEnd* *G* *smult* *V* *T*
proof (*rule* *FGModuleEnd.intro*, *rule* *FGModuleHom.intro*, *rule* *G*)
 from *endo* show *RModuleHom* (*Group.group-ring* *G*) *smult* *V* *smult* *T*
 using *RModuleEnd.axioms(1)* by *fast*
 from *endo* show *FGModuleEnd-axioms* *V* *T*
 using *RModuleEnd.endomorph* by *unfold-locales*
 qed

lemma (in *FGModule*) *VecEnd-GMap-is-FGModuleEnd* :
 assumes *endo* : *VectorSpaceEnd fsmult* *V* *T*
 and *G-map*: $\bigwedge g v. g \in G \Longrightarrow v \in V \Longrightarrow T (g * v) = g * (T v)$
 shows *FGModuleEnd* *G* *smult* *V* *T*
proof (*rule* *FGModuleEnd.intro*, *rule* *VecHom-GMap-is-FGModuleHom*)
 from *endo* show *VectorSpaceHom* ($\#$.) *V* ($\#$.) *T*
 using *VectorSpaceEnd.axioms(1)* by *fast*
 from *endo* show *T* '*V* \subseteq *V* using *VectorSpaceEnd.endomorph* by *fast*
 from *endo* show *FGModuleEnd-axioms* *V* *T*
 using *VectorSpaceEnd.endomorph* by *unfold-locales*
 qed (*unfold-locales*, *rule* *G-map*)

lemma (in *FGModule*) *GEnd-inner-dirsum-el-decomp-nth* :
 $\llbracket \forall U \in \text{set } Us. G\text{Subspace } U; \text{add-independent } S \text{ } Us; n < \text{length } Us \rrbracket$
 $\Longrightarrow FGModuleEnd G smult (\bigoplus U \leftarrow Us. U) (\bigoplus Us \downarrow n)$
 using *GroupG RModuleEnd-inner-dirsum-el-decomp-nth*
RModuleEnd-over-group-ring-is-FGModuleEnd
 by *fast*

context *FGModuleEnd*
begin

lemma *RModuleEnd* : *RModuleEnd* *ActingGroup.group-ring* *smult* *V* *T*
 using *endomorph* by *unfold-locales*

lemma *VectorSpaceEnd* : *VectorSpaceEnd fsmult* *V* *T*
 by (
rule *VectorSpaceEnd.intro*, *rule* *VectorSpaceHom*, *unfold-locales*,
rule *endomorph*

)

lemmas *proj-decomp* = *RModuleEnd.proj-decomp*[*OF RModuleEnd*]
lemmas *GSubspace-Ker* = *GSubspace-Ker*
lemmas *FGModuleHom-restrict0-GSubspace* = *FGModuleHom-restrict0-GSubspace*

end

5.4.4 Basic facts about isomorphisms

context *FGModuleIso*
begin

lemmas *VectorSpaceHom* = *VectorSpaceHom*

abbreviation *invT* \equiv (*the-inv-into V T*) \downarrow *W*

lemma *RModuleIso* : *RModuleIso FG smult V smult' T W*

proof (*rule RModuleIso.intro*)

show *RModuleHom FG* (\cdot) *V* (\star) *T*

using *FGModuleIso-axioms FGModuleIso.axioms*(1) *FGModuleHom.axioms*(2)

by *fast*

qed (*unfold-locales, rule bijective*)

lemmas *ImG* = *RModuleIso.ImG*[*OF RModuleIso*]

lemma *FGModuleIso-restrict0-GSubspace* :

assumes *GSubspace U*

shows *FGModuleIso G smult U smult' (T \downarrow U) (T ' U)*

proof (*rule FGModuleIso.intro*)

from *assms* **show** *FGModuleHom G* (\cdot) *U* (\star) (*T \downarrow U*)

using *FGModuleHom-restrict0-GSubspace* **by** *fast*

show *FGModuleIso-axioms U (T \downarrow U) (T ' U)*

proof

from *assms bijective* **have** *bij-betw T U (T ' U)*

using *subset-inj-on unfolding bij-betw-def* **by** *auto*

thus *bij-betw (T \downarrow U) U (T ' U)* **unfolding** *bij-betw-def inj-on-def* **by** *auto*

qed

qed

lemma *inv* : *FGModuleIso G smult' W smult invT V*

proof (*rule FGModuleIso.intro, rule FGModuleHom.intro*)

show *RModuleHom FG* (\star) *W* (\cdot) *invT*

using *RModuleIso.inv*[*OF RModuleIso*] *RModuleIso.axioms*(1) **by** *fast*

show *FGModuleIso-axioms W invT V*

using *RModuleIso.inv*[*OF RModuleIso*] *RModuleIso.bijective* **by** *unfold-locales*

qed (*unfold-locales*)

lemma *FGModIso-composite-left* :

assumes $FGModuleIso\ G\ smult'\ W\ smult''\ S\ X$
shows $FGModuleIso\ G\ smult\ V\ smult''\ (S\ \circ\ T)\ X$
proof (rule $FGModuleIso.intro$)
from $assms$ **show** $FGModuleHom\ G\ (\cdot)\ V\ smult''\ (S\ \circ\ T)$
using $FGModuleIso.axioms(1)\ ImG\ FGModHom-composite-left$ **by** *fast*
show $FGModuleIso.axioms\ V\ (S\ \circ\ T)\ X$
using $bijjective\ FGModuleIso.bijjective[OF\ assms]$ $bij-betw-trans$ **by** *unfold-locales*
qed

lemma $isomorphic-sym : FGModule.isomorphic\ G\ smult'\ W\ smult\ V$
using inv **by** *fast*

lemma $isomorphic-trans :$
 $FGModule.isomorphic\ G\ smult'\ W\ smult''\ X$
 $\implies FGModule.isomorphic\ G\ smult\ V\ smult''\ X$
using $FGModIso-composite-left$ **by** *fast*

lemma $isomorphic-to-zero-left : V = 0 \implies W = 0$
using $bijjective\ bij-betw-imp-surj-on\ im-zero$ **by** *fastforce*

lemma $isomorphic-to-zero-right : W = 0 \implies V = 0$
using $isomorphic-sym\ FGModuleIso.isomorphic-to-zero-left$ **by** *fast*

lemma $isomorphic-to-irr-right' :$
assumes $\bigwedge U. FGModule.GSubspace\ G\ smult'\ W\ U \implies U = 0 \vee U = W$
shows $\bigwedge U. GSubspace\ U \implies U = 0 \vee U = V$
proof–
fix U **assume** $U : GSubspace\ U$
have $U \neq V \implies U = 0$
proof–
assume $UV : U \neq V$
from U $bijjective$ **have** $T\ 'U = T\ 'V \implies U = V$
using $bij-betw-imp-inj-on[of\ T\ V\ W]\ inj-onD[of\ T\ V]$ **by** *fast*
with UV $bijjective$ **have** $T\ 'U \neq W$ **using** $bij-betw-imp-surj-on$ **by** *fast*
moreover **from** U **have** $FGModule.GSubspace\ G\ smult'\ W\ (T\ 'U)$
using $ImG\ im-submodule$ **by** *fast*
ultimately **show** $U = 0$
using $assms\ U\ FGModuleIso-restrict0-GSubspace$
 $FGModuleIso.isomorphic-to-zero-right$
by *fast*
qed
thus $U = 0 \vee U = V$ **by** *fast*
qed

end

context $FGModule$
begin

lemma *isomorphic-sym* :
isomorphic smult' W \implies *FGModule.isomorphic G smult' W smult V*
using *FGModuleIso.inv* **by fast**

lemma *isomorphic-trans* :
isomorphic smult' W \implies *FGModule.isomorphic G smult' W smult'' X*
 \implies *isomorphic smult'' X*
using *FGModuleIso.FGModIso-composite-left* **by fast**

lemma *isomorphic-to-zero-left* : *V = 0* \implies *isomorphic smult' W* \implies *W = 0*
using *FGModuleIso.isomorphic-to-zero-left* **by fast**

lemma *isomorphic-to-zero-right* : *isomorphic smult' 0* \implies *V = 0*
using *FGModuleIso.isomorphic-to-zero-right* **by fast**

lemma *FGModIso-idhom* : *FGModuleIso G smult V smult (id↓V)* *V*
using *FGModHom-idhom*
proof (*rule FGModuleIso.intro*)
show *FGModuleIso-axioms V (id↓V)* *V*
using *bij-betw-id bij-betw-restrict0* **by unfold-locales fast**
qed

lemma *isomorphic-reft* : *isomorphic smult V* **using** *FGModIso-idhom* **by fast**

end

5.4.5 Hom-sets

definition *FGModuleHomSet* ::
'g::group-add set \implies (*'f::field,'g*) *aezfun* \implies *'v::ab-group-add* \implies *'w*) \implies *'v set*
 \implies (*'f,'g*) *aezfun* \implies *'w::ab-group-add* \implies *'w*) \implies *'w set*
 \implies (*'v* \implies *'w*) *set*
where *FGModuleHomSet G fgsmult V fgsmult' W*
 $\equiv \{T. \text{FGModuleHom } G \text{ fgsmult } V \text{ fgsmult' } T\} \cap \{T. T \text{ ' } V \subseteq W\}$

lemma *FGModuleHomSetI* :
FGModuleHom G fgsmult V fgsmult' T \implies *T ' V* \subseteq *W*
 \implies *T* \in *FGModuleHomSet G fgsmult V fgsmult' W*
unfolding *FGModuleHomSet-def* **by fast**

lemma *FGModuleHomSetD-FGModuleHom* :
T \in *FGModuleHomSet G fgsmult V fgsmult' W*
 \implies *FGModuleHom G fgsmult V fgsmult' T*
unfolding *FGModuleHomSet-def* **by fast**

lemma *FGModuleHomSetD-Im* :
T \in *FGModuleHomSet G fgsmult V fgsmult' W* \implies *T ' V* \subseteq *W*
unfolding *FGModuleHomSet-def* **by fast**

context *FGModule*
begin

lemma *FGModuleHomSet-is-Gmaps-in-VectorSpaceHomSet* :

fixes *smult'* :: ('f, 'g) aezfun \Rightarrow 'w::ab-group-add \Rightarrow '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-scalar-mult.fsmult *smult'*

and *Gmult'* : *Gmult'* \equiv aezfun-scalar-mult.Gmult *smult'*

assumes *FGModW* : *FGModule* *G smult' W*

shows *FGModuleHomSet* *G smult V smult' W*
 $=$ (*VectorSpaceHomSet* *fsmult V fsmult' W*)
 $\cap \{T. \forall g \in G. \forall v \in V. T (g * \cdot v) = g ** (T v)\}$

proof

from *fsmult' Gmult'*

show *FGModuleHomSet* *G smult V smult' W*
 \subseteq (*VectorSpaceHomSet* *fsmult V fsmult' W*)
 $\cap \{T. \forall g \in G. \forall v \in V. T (g * \cdot v) = g ** T v\}$

using *FGModuleHomSetD-FGModuleHom*[of - *G smult V smult'*]
FGModuleHom.VectorSpaceHom[of *G smult V smult'*]
FGModuleHomSetD-Im[of - *G smult V smult'*]
VectorSpaceHomSetI[of *fsmult V fsmult'*]
FGModuleHom.G-map[of *G smult V smult'*]

by *auto*

show *FGModuleHomSet* *G smult V smult' W*
 \supseteq (*VectorSpaceHomSet* *fsmult V fsmult' W*)
 $\cap \{T. \forall g \in G. \forall v \in V. T (g * \cdot v) = g ** T v\}$

proof

fix *T*

assume *T*: *T* \in (*VectorSpaceHomSet* *fsmult V fsmult' W*)
 $\cap \{T. \forall g \in G. \forall v \in V. T (g * \cdot v) = g ** T v\}$

show *T* \in *FGModuleHomSet* *G smult V smult' W*

proof (*rule* *FGModuleHomSetI*, *rule* *VecHom-GMap-is-FGModuleHom*)

from *T fsmult'*

show *VectorSpaceHom* (# \cdot) *V (aezfun-scalar-mult.fsmult smult')* *T*

using *VectorSpaceHomSetD-VectorSpaceHom*

by *fast*

from *T* **show** *T ' V* \subseteq *W* **using** *VectorSpaceHomSetD-Im* **by** *fast*

from *T Gmult'*

show $\bigwedge g v. g \in G \implies v \in V$
 $\implies T (g * \cdot v) =$ aezfun-scalar-mult.Gmult (\star) *g* (*T v*)

by *fast*

from *T* **show** *T ' V* \subseteq *W* **using** *VectorSpaceHomSetD-Im* **by** *fast*

qed (*rule* *FGModW*)

qed

qed

lemma *Group-FGModuleHomSet* :

fixes *smult'* :: ('f, 'g) aezfun \Rightarrow 'w::ab-group-add \Rightarrow '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 Group (FGModuleHomSet G smult V smult' W)
proof
from FGModW show FGModuleHomSet G (·) V smult' W ≠ {}
using FGModule.smult-zero trivial-FGModuleHom[of smult'] FGModule.zero-closed
FGModuleHomSetI
by fastforce
next
fix S T
assume S: S ∈ FGModuleHomSet G (·) V smult' W
and T: T ∈ FGModuleHomSet G (·) V smult' W
with assms
have ST: S ∈ (VectorSpaceHomSet fsmult V fsmult' W)
∩ {T. ∀ g ∈ G. ∀ v ∈ V. T (g *· v) = g ** T v}
T ∈ (VectorSpaceHomSet fsmult V fsmult' W)
∩ {T. ∀ g ∈ G. ∀ v ∈ V. T (g *· v) = g ** T v}
using FGModuleHomSet-is-Gmaps-in-VectorSpaceHomSet
by auto
with fsmult' have S - T ∈ VectorSpaceHomSet fsmult V fsmult' W
using FGModule.fVectorSpace[OF FGModW]
VectorSpace.Group-VectorSpaceHomSet[OF fVectorSpace] Group.diff-closed
by fast
moreover have ∧ g v. g ∈ G ⇒ v ∈ V ⇒ (S - T) (g *· v) = g ** ((S - T) v)
proof -
fix g v assume g ∈ G v ∈ V
moreover with ST have S v ∈ W T v ∈ W - T v ∈ W
using VectorSpaceHomSetD-Im[of S fsmult V fsmult']
VectorSpaceHomSetD-Im[of T fsmult V fsmult']
FGModule.neg-closed[OF FGModW]
by auto
ultimately show (S - T) (g *· v) = g ** ((S - T) v)
using ST Gmult' FGModule.neg-Gmult[OF FGModW]
FGModule.Gmult-distrib-left[OF FGModW, of g S v - T v]
by auto
qed
ultimately show S - T ∈ FGModuleHomSet G (·) V smult' W
using fsmult' Gmult'
FGModuleHomSet-is-Gmaps-in-VectorSpaceHomSet[OF FGModW]
by fast
qed
lemma Subspace-FGModuleHomSet :
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   hom-fsmult :: 'f ⇒ ('v ⇒ 'w) ⇒ ('v ⇒ 'w) (infixr #★ 70)
defines fsmult'   : fsmult' ≡ aezfun-scalar-mult.fsmult smult'
and   Gmult'     : Gmult' ≡ aezfun-scalar-mult.Gmult smult'
defines hom-fsmult : hom-fsmult ≡ λa T v. a #★ T v
assumes FGModW   : FGModule G smult' W
shows  VectorSpace.Subspace hom-fsmult
        (VectorSpaceHomSet fsmult V fsmult' W)
        (FGModuleHomSet G smult V smult' W)
proof (rule VectorSpace.SubspaceI)
from hom-fsmult fsmult'
  show VectorSpace (#★.) (VectorSpaceHomSet (#.) V (#★) W)
  using FGModule.fVectorSpace[OF FGModW]
        VectorSpace.VectorSpace-VectorSpaceHomSet[OF fVectorSpace]
  by fast
from fsmult' Gmult' FGModW
  show Group (FGModuleHomSet G (.) V (★) W)
        ∧ FGModuleHomSet G (.) V (★) W
        ⊆ VectorSpaceHomSet (#.) V (#★) W
  using Group-FGModuleHomSet FGModuleHomSet-is-Gmaps-in-VectorSpaceHomSet
  by fast
next
fix a T assume T: T ∈ FGModuleHomSet G (.) V (★) W
from hom-fsmult fsmult' have FGModuleHom G smult V smult' (a #★ T)
  using FGModuleHomSetD-FGModuleHom[OF T]
        FGModuleHomSetD-Im[OF T]
        FGModuleHom.FGModuleHom-fscalar-mul
  by simp
moreover from hom-fsmult fsmult' have (a #★ T) ' V ⊆ W
  using FGModuleHomSetD-Im[OF T] FGModule.fsmult-closed[OF FGModW]
  by auto
ultimately show a #★ T ∈ FGModuleHomSet G (.) V (★) W
  using FGModuleHomSetI by fastforce
qed

```

```

lemma VectorSpace-FGModuleHomSet :
fixes smult'     :: ('f, 'g) aezfun ⇒ 'w::ab-group-add ⇒ 'w (infixr ★ 70)
and   fsmult'   :: 'f ⇒ 'w ⇒ 'w (infixr #★ 70)
and   hom-fsmult :: 'f ⇒ ('v ⇒ 'w) ⇒ ('v ⇒ 'w) (infixr #★ 70)
defines fsmult' ≡ aezfun-scalar-mult.fsmult smult'
defines hom-fsmult ≡ λa T v. a #★ T v
assumes FGModule G smult' W
shows  VectorSpace hom-fsmult (FGModuleHomSet G smult V smult' W)
using assms Subspace-FGModuleHomSet Module.intro VectorSpace.intro
by fast

```

end

5.5 Induced modules

5.5.1 Additive function spaces

definition *addfunset* ::

'a::monoid-add set \Rightarrow *'m::monoid-add set* \Rightarrow (*'a* \Rightarrow *'m*) *set*
where *addfunset* *A M* \equiv $\{f. \text{supp } f \subseteq A \wedge \text{range } f \subseteq M$
 $\wedge (\forall x \in A. \forall y \in A. f (x+y) = f x + f y) \}$

lemma *addfunsetI* :

$\llbracket \text{supp } f \subseteq A; \text{range } f \subseteq M; \forall x \in A. \forall y \in A. f (x+y) = f x + f y \rrbracket$
 $\implies f \in \text{addfunset } A M$

unfolding *addfunset-def* **by** *fast*

lemma *addfunsetD-supp* : $f \in \text{addfunset } A M \implies \text{supp } f \subseteq A$

unfolding *addfunset-def* **by** *fast*

lemma *addfunsetD-range* : $f \in \text{addfunset } A M \implies \text{range } f \subseteq M$

unfolding *addfunset-def* **by** *fast*

lemma *addfunsetD-range'* : $f \in \text{addfunset } A M \implies f x \in M$

using *addfunsetD-range* **by** *fast*

lemma *addfunsetD-add* :

$\llbracket f \in \text{addfunset } A M; x \in A; y \in A \rrbracket \implies f (x+y) = f x + f y$

unfolding *addfunset-def* **by** *fast*

lemma *addfunset0* : $\text{addfunset } A (0::'m::\text{monoid-add set}) = 0$

proof

show $\text{addfunset } A 0 \subseteq 0$ **using** *addfunsetD-range'* **by** *fastforce*

have $(0::'a \Rightarrow 'm) \in \text{addfunset } A 0$

using *supp-zerofun-subset-any* **by** (*rule addfunsetI*) *auto*

thus $\text{addfunset } A (0::'m::\text{monoid-add set}) \supseteq 0$ **by** *simp*

qed

lemma *Group-addfunset* :

fixes *M::'g::ab-group-add set*

assumes *Group M*

shows *Group (addfunset R M)*

proof

from *assms* **show** $\text{addfunset } R M \neq \{\}$

using *addfunsetI[of 0 R M]* *supp-zerofun-subset-any* *Group.zero-closed*

by *fastforce*

next

fix *g h* **assume** *gh: g* \in *addfunset R M* *h* \in *addfunset R M*

show $g - h \in \text{addfunset } R M$

proof (*rule addfunsetI*)

from *gh* **show** $\text{supp } (g - h) \subseteq R$

using *addfunsetD-supp* *supp-diff-subset-union-supp* **by** *fast*

from *gh* **show** $\text{range } (g - h) \subseteq M$

using *addfunsetD-range Group.diff-closed [OF assms]*
by (*simp add: addfunsetD-range' image-subsetI*)
show $\forall x \in R. \forall y \in R. (g - h) (x + y) = (g - h) x + (g - h) y$
using *addfunsetD-add[OF gh(1)] addfunsetD-add[OF gh(2)] by simp*
qed
qed

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 $\text{smultfunset } G \text{ } FH \equiv \{f. (\forall a::'r. \forall g \in G. \forall x \in FH. f (a \ \delta\delta \ g \ * \ x) = (a \ \delta\delta \ g) \cdot (f \ x))\}$

lemma *smultfunsetD* :
 $[f \in \text{smultfunset } G \text{ } FH; g \in G; x \in FH] \Longrightarrow f (a \ \delta\delta \ g \ * \ x) = (a \ \delta\delta \ g) \cdot (f \ x)$
unfolding *smultfunset-def* **by** *fast*

lemma *smultfunsetI* :
 $\forall a::'r. \forall g \in G. \forall x \in FH. f (a \ \delta\delta \ g \ * \ x) = (a \ \delta\delta \ g) \cdot (f \ x)$
 $\Longrightarrow f \in \text{smultfunset } G \text{ } FH$
unfolding *smultfunset-def* **by** *fast*

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 $\text{indspace } G \text{ } FH \text{ } V = \text{addfunset } FH \text{ } V \cap \text{smultfunset } G \text{ } FH$

lemma *indspaceD* :
 $f \in \text{indspace } G \text{ } FH \text{ } V \Longrightarrow f \in \text{addfunset } FH \text{ } V \cap \text{smultfunset } G \text{ } FH$
using *indspace-def* **by** *fast*

lemma *indspaceD-supp* : $f \in \text{indspace } G \text{ } FH \text{ } V \Longrightarrow \text{supp } f \subseteq FH$
using *indspace-def addfunsetD-supp* **by** *fast*

lemma *indspaceD-supp'* : $f \in \text{indspace } G \text{ } FH \text{ } V \Longrightarrow x \notin FH \Longrightarrow f \ x = 0$
using *indspaceD-supp suppI-contr* **by** *fast*

lemma *indspaceD-range* : $f \in \text{indspace } G \text{ } FH \text{ } V \Longrightarrow \text{range } f \subseteq V$
using *indspace-def addfunsetD-range* **by** *fast*

lemma *indspaceD-range'*: $f \in \text{indspace } G \text{ FH } V \implies f x \in V$
using *indspaceD-range* **by** *fast*

lemma *indspaceD-add* :
 $\llbracket f \in \text{indspace } G \text{ FH } V; x \in \text{FH}; y \in \text{FH} \rrbracket \implies f (x+y) = f x + f y$
using *indspace-def addfunsetD-add* **by** *auto*

lemma *indspaceD-transform* :
 $\llbracket f \in \text{indspace } G \text{ FH } V; g \in G; x \in \text{FH} \rrbracket \implies f (a \delta\delta g * x) = (a \delta\delta g) \cdot (f x)$
using *indspace-def smultfunsetD* **by** *auto*

lemma *indspaceI* :
 $f \in \text{addfunset } \text{FH } V \implies f \in \text{smultfunset } G \text{ FH} \implies f \in \text{indspace } G \text{ FH } V$
using *indspace-def* **by** *fast*

lemma *indspaceI'* :
 $\llbracket \text{supp } f \subseteq \text{FH}; \text{range } f \subseteq V; \forall x \in \text{FH}. \forall y \in \text{FH}. f (x+y) = f x + f y;$
 $\forall a::'r. \forall g \in G. \forall x \in \text{FH}. f (a \delta\delta g * x) = (a \delta\delta g) \cdot (f x) \rrbracket$
 $\implies f \in \text{indspace } G \text{ FH } V$
using *smultfunsetI addfunsetI[of f] indspaceI* **by** *simp*

lemma *mono-indspace* : *mono (indspace G FH)*

proof (*rule monoI*)

fix $U V :: 'v \text{ set}$ **assume** $U \subseteq V$

show $\text{indspace } G \text{ FH } U \subseteq \text{indspace } G \text{ FH } V$

proof

fix f **assume** $f: f \in \text{indspace } G \text{ FH } U$

show $f \in \text{indspace } G \text{ FH } V$ **using** *indspaceD-supp[OF f]*

proof (*rule indspaceI'*)

from $f \text{ U-V}$ **show** $\text{range } f \subseteq V$ **using** *indspaceD-range[of f G FH]* **by** *auto*

from f **show** $\forall x \in \text{FH}. \forall y \in \text{FH}. f (x+y) = f x + f y$

using *indspaceD-add* **by** *auto*

from f **show** $\forall a::'r. \forall g \in G. \forall x \in \text{FH}. f (a \delta\delta g * x) = (a \delta\delta g) \cdot (f x)$

using *indspaceD-transform* **by** *auto*

qed

qed

qed

end

context *FGModule*

begin

lemma *zero-transforms* : $0 \in \text{smultfunset } G \text{ FH}$
using *smultfunsetI FG-fddg-closed smult-zero* **by** *simp*

lemma *indspace0* : $\text{indspace } G \text{ FH } 0 = 0$
using *zero-transforms addfunset0 indspace-def* **by** *auto*

```

lemma Group-indspace :
  assumes Ring1 FH
  shows Group (indspace G FH V)
proof
  from zero-closed have  $0 \subseteq V$  by simp
  with mono-indspace [of G FH]
  have indspace G FH 0  $\subseteq$  indspace G FH V
    by (auto dest!: monoD [of - 0 V])
  then show indspace G FH V  $\neq$   $\{\}$ 
    using indspace0 [of FH] by auto
next
  fix f1 f2 assume ff: f1  $\in$  indspace G FH V f2  $\in$  indspace G FH V
  hence f1 - f2  $\in$  addfunset FH V
    using assms indspaceD indspaceD Group Group-addfunset Group.diff-closed
    by fast
  moreover from ff have f1 - f2  $\in$  smultfunset G FH
    using indspaceD-transform FG-fddg-closed indspaceD-range' smult-distrib-left-diff
      smultfunsetI
    by simp
  ultimately show f1 - f2  $\in$  indspace G FH V using indspaceI by fast
qed

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 :  $x \in R \Longrightarrow (r \bowtie f) x = f (x * r)$ 
  unfolding rightreg-scalar-mult-def by simp

```

```

lemma rightreg-scalar-multD2 :  $x \notin R \Longrightarrow (r \bowtie f) x = 0$ 
  unfolding rightreg-scalar-mult-def by simp

```

```

lemma rrsmult-supp : supp (r  $\bowtie$  f)  $\subseteq R$ 
  using rightreg-scalar-multD2 suppD-contr by force

```

```

lemma rrsmult-range : range (r  $\bowtie$  f)  $\subseteq$   $\{0\} \cup$  range f

```

```

proof (rule image-subsetI)
  fix x show  $(r \bowtie f) x \in \{0\} \cup$  range f
    using rightreg-scalar-multD1 [of x r f] image-eqI
      rightreg-scalar-multD2 [of x r f]
    by (cases x  $\in R$ ) auto
qed

```

```

lemma rrsmult-distrib-left :  $r \bowtie (f + g) = r \bowtie f + r \bowtie g$ 
proof
  fix  $x$  show  $(r \bowtie (f + g)) x = (r \bowtie f + r \bowtie g) x$ 
    unfolding rightreg-scalar-mult-def by (cases  $x \in R$ ) auto
qed

lemma rrsmult-distrib-right :
  assumes  $\bigwedge x y. x \in R \implies y \in R \implies f(x+y) = f x + f y$   $r \in R$   $s \in R$ 
  shows  $(r + s) \bowtie f = r \bowtie f + s \bowtie f$ 
proof
  fix  $x$  show  $((r + s) \bowtie f) x = (r \bowtie f + s \bowtie f) x$ 
    using assms mult-closed
    unfolding rightreg-scalar-mult-def
    by (cases  $x \in R$ ) (auto simp add: distrib-left)
qed

lemma RModule-addfunset :
  fixes  $M :: 'g :: ab\text{-group-add set}$ 
  assumes Group M
  shows RModule R rightreg-scalar-mult (addfunset R M)
proof (rule RModuleI)

  from assms show Group (addfunset R M) using Group-addfunset by fast

  show RModule-axioms R ( $\bowtie$ ) (addfunset R M)
  proof
    fix  $r f$  assume  $r: r \in R$  and  $f: f \in \text{addfunset } R M$ 
    show  $r \bowtie f \in \text{addfunset } R M$ 
    proof (rule addfunsetI)
      show  $\text{supp } (r \bowtie f) \subseteq R$ 
        using rightreg-scalar-multD2 suppD-contr by force
      show  $\text{range } (r \bowtie f) \subseteq M$ 
        using addfunsetD-range[OF f] Group.zero-closed[OF assms]
        unfolding rightreg-scalar-mult-def
        by fastforce
      from  $r$  show  $\forall x \in R. \forall y \in R. (r \bowtie f) (x + y) = (r \bowtie f) x + (r \bowtie f) y$ 
        using mult-closed add-closed addfunsetD-add[OF f]
        unfolding rightreg-scalar-mult-def
        by (simp add: distrib-right)
    qed
  next
    show  $\bigwedge r f g. r \bowtie (f + g) = r \bowtie f + r \bowtie g$  using rrsmult-distrib-left by fast
  next
    fix  $r s f$  assume  $r \in R$   $s \in R$   $f \in \text{addfunset } R M$ 
    thus  $(r + s) \bowtie f = r \bowtie f + s \bowtie f$ 
      using addfunsetD-add[of f] rrrsmult-distrib-right[of f] by simp
  next
    fix  $r s f$  assume  $r: r \in R$  and  $s: s \in R$  and  $f: f \in \text{addfunset } R M$ 

```

```

show  $r \bowtie s \bowtie f = (r * s) \bowtie f$ 
proof
  fix  $x$  from  $r$  show  $(r \bowtie s \bowtie f) x = ((r * s) \bowtie f) x$ 
    using mult-closed unfolding rightreg-scalar-mult-def
    by (cases  $x \in R$ ) (auto simp add: mult.assoc)
qed
next
fix  $f$  assume  $f: f \in \text{addfunset } R \ M$ 
show  $1 \bowtie f = f$ 
proof
  fix  $x$  show  $(1 \bowtie f) x = f x$ 
    unfolding rightreg-scalar-mult-def
    using addfunsetD-supp[OF f] suppI-contr[of x f]
      contra-subsetD[of supp f]
    by (cases  $x \in R$ ) auto
qed
qed
qed (unfold-locales)

```

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  $\text{rrsmult}$ 
  for  $H$       :: ' $g$ ::group-add set
  and  $G$       :: ' $g$  set
  and  $FG$      :: (' $f$ ::field, ' $g$ ) aezfun set
  and  $\text{smult}$   :: (' $f$ , ' $g$ ) aezfun  $\Rightarrow$  ' $v$ ::ab-group-add  $\Rightarrow$  ' $v$  (infixl  $\cdot$  70)
  and  $V$       :: ' $v$  set
  and  $\text{rrsmult}$  :: (' $f$ , ' $g$ ) aezfun  $\Rightarrow$  ((' $f$ , ' $g$ ) aezfun  $\Rightarrow$  ' $v$ )  $\Rightarrow$  ((' $f$ , ' $g$ ) aezfun  $\Rightarrow$  ' $v$ )
                                     (infixl  $\bowtie$  70)
+ fixes  $FH$     :: (' $f$ , ' $g$ ) aezfun set
  and  $\text{ind}V$     :: ((' $f$ , ' $g$ ) aezfun  $\Rightarrow$  ' $v$ ) set
  defines  $FH$    :  $FH \equiv \text{Supgroup.group-ring}$ 
  and  $\text{ind}V$    :  $\text{ind}V \equiv \text{BaseFGMod.indspace } G \ FH \ V$ 
  assumes  $\text{rrsmult}$  :  $\text{rrsmult} = \text{Ring1.rightreg-scalar-mult } FH$ 
  and Subgroup: Supgroup.Subgroup  $G$ 
begin

abbreviation  $\text{indfsmult}$  ::
  ' $f \Rightarrow$  ((' $f$ , ' $g$ ) aezfun  $\Rightarrow$  ' $v$ )  $\Rightarrow$  ((' $f$ , ' $g$ ) aezfun  $\Rightarrow$  ' $v$ ) (infixl  $\bowtie$  70)
  where  $\text{indfsmult} \equiv \text{induced-smult.fsmult}$ 
abbreviation  $\text{indflincomb}$  ::

```

$'f \text{ list} \Rightarrow (('f, 'g) \text{ aezfun} \Rightarrow 'v) \text{ list} \Rightarrow (('f, 'g) \text{ aezfun} \Rightarrow 'v)$ (**infixl** \cdot \times 70)
where $\text{indflincomb} \equiv \text{induced-smult.flincomb}$
abbreviation $\text{Hmult} ::$
 $'g \Rightarrow (('f, 'g) \text{ aezfun} \Rightarrow 'v) \Rightarrow (('f, 'g) \text{ aezfun} \Rightarrow 'v)$ (**infixl** $*$ \times 70)
where $\text{Hmult} \equiv \text{induced-smult.Gmult}$

lemma $\text{Ring1-FH} : \text{Ring1 FH}$ **using** $\text{FH Supgroup.Ring1-RG}$ **by fast**

lemma $\text{FG-subring-FH} : \text{Ring1.Subring1 FH BaseFGMod.FG}$
using $\text{FH Supgroup.Subgroup-imp-Subring[OF Subgroup]}$ **by fast**

lemma $\text{rrsmultD1} : x \in \text{FH} \implies (r \times f) x = f (x*r)$
using $\text{Ring1.rightreg-scalar-multD1[OF Ring1-FH]}$ rrsmult **by simp**

lemma $\text{rrsmultD2} : x \notin \text{FH} \implies (r \times f) x = 0$
using $\text{Ring1.rightreg-scalar-multD2[OF Ring1-FH]}$ rrsmult **by fast**

lemma $\text{rrsmult-supp} : \text{supp } (r \times f) \subseteq \text{FH}$
using $\text{Ring1.rrsmult-supp[OF Ring1-FH]}$ rrsmult **by auto**

lemma $\text{rrsmult-range} : \text{range } (r \times f) \subseteq \{0\} \cup \text{range } f$
using $\text{Ring1.rrsmult-range[OF Ring1-FH]}$ rrsmult **by auto**

lemma $\text{FHModule-addfunset} : \text{FGModule } H \text{ rrsmult } (\text{addfunset } \text{FH } V)$
proof ($\text{rule } \text{FGModule.intro}$)
from FH rrsmult **show** $\text{RModule Supgroup.group-ring } (\times) (\text{addfunset } \text{FH } V)$
using $\text{Group Supgroup.Ring1-RG Ring1.RModule-addfunset}$ **by fast**
qed (unfold-locales)

lemma $\text{FHSubmodule-indspace} :$
 $\text{FGModule.FGSubmodule } H \text{ rrsmult } (\text{addfunset } \text{FH } V) \text{ indV}$
proof ($\text{rule } \text{FGModule.FGSubmoduleI[of H]}$, $\text{rule } \text{FHModule-addfunset}$, $\text{rule } \text{conjI}$)
from Ring1-FH indV **show** Group indV **using** Group-indspace **by fast**
from indV **show** $\text{indV} \subseteq \text{addfunset } \text{FH } V$
using $\text{BaseFGMod.indspaceD}$ **by fast**

next
fix $r f$ **assume** $\text{rf} : r \in (\text{Supgroup.group-ring} :: ('f, 'g) \text{ aezfun set}) f \in \text{indV}$
from $\text{rf}(2)$ indV **have** $\text{rf2}' : f \in \text{BaseFGMod.indspace } G \text{ FH } V$ **by fast**
show $r \times f \in \text{indV}$
unfolding indV
proof ($\text{rule } \text{BaseFGMod.indspaceI}'$, $\text{rule } \text{rrsmult-supp}$)
show $\text{range } (r \times f) \subseteq V$
using $\text{rrsmult-range BaseFGMod.indspaceD-range[OF rf2']}$ zero-closed
by force
from $\text{FH rf}(1)$ $\text{rf2}'$
show $\forall x \in \text{FH}. \forall y \in \text{FH}. (r \times f) (x + y) = (r \times f) x + (r \times f) y$
using $\text{Ring1.add-closed[OF Ring1-FH]}$ $\text{rrsmultD1[of - r f]}$
 $\text{Ring1.mult-closed[OF Ring1-FH]}$ $\text{BaseFGMod.indspaceD-add}$
by ($\text{simp add: distrib-right}$)


```

{
  fix a g x assume gx: g ∈ G x ∈ FH
  with FH have a δδ g * x ∈ FH
    using FG-fddg-closed FG-subring-FH Ring1.mult-closed[OF Ring1-FH]
    by fast
  with FH rf(1) gx(2) have (r ⋈ f) (a δδ g * x) = a δδ g · ((r ⋈ f) x)
    using rrsmultD1[of - r f] Ring1.mult-closed[OF Ring1-FH]
      BaseFGMod.indspaceD-transform[OF rf2' gx(1)]
    by (simp add: mult.assoc)
}
thus ∀ a. ∀ g ∈ G. ∀ x ∈ FH. (r ⋈ f) (a δδ g * x) = a δδ g · (r ⋈ f) x by fast
qed
qed

```

lemma *FHModule-indspace* : *FGModule H rrsmult indV*

proof (rule *FGModule.intro*)

show *RModule Supgroup.group-ring (⋈) indV* **using** *FHSubmodule-indspace* **by** *fast*

qed (*unfold-locales*)

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* ⇒ ((*f*, '*g*) aezfun ⇒ '*v*)

where *induced-vector* *v* ≡ (if *v* ∈ *V*

then (λ*y*. if *y* ∈ *FH* then (*FG-proj* *y*) · *v* else 0) else 0)

lemma *induced-vector-apply1* :

v ∈ *V* ⇒ *x* ∈ *FH* ⇒ *induced-vector* *v* *x* = (*FG-proj* *x*) · *v*

using *induced-vector-def* **by** *simp*

lemma *induced-vector-apply2* : *v* ∈ *V* ⇒ *x* ∉ *FH* ⇒ *induced-vector* *v* *x* = 0

using *induced-vector-def* **by** *simp*

lemma *induced-vector-indV* :

assumes *v*: *v* ∈ *V*

shows *induced-vector* *v* ∈ *indV*

unfolding *indV*

proof (rule *BaseFGMod.indspaceI'*)

from *assms* **show** *supp* (*induced-vector* *v*) ⊆ *FH*

using *induced-vector-def* *supp-restrict0*[of *FH* λ*y*. (*FG-proj* *y*) · *v*] **by** *simp*

show *range* (*induced-vector* *v*) ⊆ *V*

proof (rule *image-subsetI*)

fix *y*

from *v* **show** (*induced-vector* *v*) *y* ∈ *V*

using *induced-vector-def* *zero-closed* *aezfun-setspace-proj-in-setspace*[of *G* *y*]

$\text{smult-closed ActingGroup.group-ringD}$
 by auto
qed
 {
 fix $x y$ **assume** $xy: x \in FH \ y \in FH$
 with v **have** $(\text{induced-vector } v) (x + y)$
 $= (\text{induced-vector } v) x + (\text{induced-vector } v) y$
 using $\text{Ring1-FH Ring1.add-closed aezfun-setspan-proj-add[of } G \ x \ y] \text{FG-proj-in-FG}$
 $\text{smult-distrib-left induced-vector-def}$
 by auto
 }
thus $\forall x \in FH. \forall y \in FH. \text{induced-vector } v (x + y)$
 $= \text{induced-vector } v x + \text{induced-vector } v y$
 by fast

{
 fix $a g x$ **assume** $g: g \in G$ **and** $x: x \in FH$
 with $v \text{ FH}$
 have $(\text{induced-vector } v) (a \ \delta\delta \ g * x) = a \ \delta\delta \ g \cdot (\text{induced-vector } v) x$
 using $\text{FG-subring-FH FG-fddg-closed Ring1-FH}$
 $\text{Ring1.mult-closed[of } FH \ a \ \delta\delta \ g] \text{FG-proj-mult-leftdelta[of } g \ a]$
 $\text{FG-fddg-closed FG-proj-in-FG smult-assoc induced-vector-def}$
 by fastforce
 }
thus $\forall a. \forall g \in G. \forall x \in FH. \text{induced-vector } v (a \ \delta\delta \ g * x)$
 $= a \ \delta\delta \ g \cdot \text{induced-vector } v x$
 by fast

qed

lemma *induced-vector-additive* :

$v \in V \implies v' \in V$
 $\implies \text{induced-vector } (v+v') = \text{induced-vector } v + \text{induced-vector } v'$
 using $\text{add-closed induced-vector-def FG-proj-in-FG smult-distrib-left}$ **by** auto

lemma *hom-induced-vector* : $\text{FGModuleHom } G \ \text{smult } V \ \text{rrsmult } \text{induced-vector}$
proof

show $\bigwedge v \ v'. v \in V \implies v' \in V$
 $\implies \text{induced-vector } (v + v') = \text{induced-vector } v + \text{induced-vector } v'$
 using $\text{induced-vector-additive}$ **by** fast

have $\text{induced-vector} = (\lambda v. \text{if } v \in V \text{ then } \lambda y. \text{if } y \in FH$
 $\text{then } (\text{FG-proj } y) \cdot v \text{ else } 0 \text{ else } 0)$

using $\text{induced-vector-def}$ **by** fast
thus $\text{supp induced-vector} \subseteq V$ using $\text{supp-restrict0[of } V]$ **by** fastforce

show $\bigwedge x \ v. x \in \text{BaseFGMod.FG} \implies v \in V$

$$\implies \text{induced-vector } (x \cdot v) = x \bowtie \text{induced-vector } v$$

proof–

fix $x v$ **assume** $xv: x \in \text{BaseFGMod.FG } v \in V$

show $\text{induced-vector } (x \cdot v) = x \bowtie \text{induced-vector } v$

proof

fix a

from $xv FH$ **show** $\text{induced-vector } (x \cdot v) a = (x \bowtie \text{induced-vector } v) a$

using $\text{smult-closed induced-vector-def FG-proj-in-FG smult-assoc rrsmultD1}$

$\text{FG-subring-FH Ring1.mult-closed[OF Ring1-FH] induced-vector-apply1}$

$\text{FG-proj-mult-right[of } x] \text{ smult-closed induced-vector-apply2 rrsmultD2}$

by auto

qed

qed

qed

lemma $\text{indspace-sum-list-fddh}$:

$\llbracket fhs \neq []; \text{set } (\text{map } \text{snd } fhs) \subseteq H; f \in \text{ind}V \rrbracket$

$\implies f (\sum (a,h) \leftarrow fhs. a \delta \delta h) = (\sum (a,h) \leftarrow fhs. f (a \delta \delta h))$

proof ($\text{induct } fhs \text{ rule: list-nonempty-induct}$)

case ($\text{single } fh$) **show** $?case$

using $\text{split-beta[of } \lambda a h. a \delta \delta h fh] \text{ split-beta[of } \lambda a h. f (a \delta \delta h) fh]$ **by** simp

next

case ($\text{cons } fh fhs$)

hence $\text{prevcase: } \text{snd } fh \in H \text{ set } (\text{map } \text{snd } fhs) \subseteq H f \in \text{ind}V$

$f (\sum (a,h) \leftarrow fhs. a \delta \delta h) = (\sum (a,h) \leftarrow fhs. f (a \delta \delta h))$

by auto

have $f (\sum (a,h) \leftarrow fh \# fhs. a \delta \delta h)$

$= f ((\text{fst } fh) \delta \delta (\text{snd } fh) + (\sum ah \leftarrow fhs. \text{case-prod } (\lambda a h. a \delta \delta h) ah))$

using $\text{split-beta[of } \lambda a h. a \delta \delta h fh]$ **by** simp

moreover from $\text{prevcase}(1) FH$ **have** $(\text{fst } fh) \delta \delta (\text{snd } fh) \in FH$

using $\text{Supgroup.RG-aezdeltafun-closed}$ **by** fast

moreover from $\text{prevcase}(2) FH$

have $(\sum ah \leftarrow fhs. \text{case-prod } (\lambda a h. a \delta \delta h) ah) \in FH$

using $\text{Supgroup.RG-aezdeltafun-closed}$

$\text{Ring1.sum-list-closed[OF Ring1-FH, of } \lambda ah. \text{case-prod } (\lambda a h. a \delta \delta h) ah$

$fhs]$

by fastforce

ultimately have $f (\sum (a,h) \leftarrow fh \# fhs. a \delta \delta h)$

$= f ((\text{fst } fh) \delta \delta (\text{snd } fh)) + f (\sum (a,h) \leftarrow fhs. a \delta \delta h)$

using $\text{ind}V \text{ prevcase}(3) \text{BaseFGMod.indspaceD-add}$ **by** simp

with $\text{prevcase}(4)$ **show** $?case$ **using** $\text{split-beta[of } \lambda a h. f (a \delta \delta h) fh]$ **by** simp

qed

lemma $\text{induced-fsmult-conv-fsmult-1ddh}$:

$f \in \text{ind}V \implies h \in H \implies (r \bowtie \bowtie f) (1 \delta \delta h) = r \ddagger. (f (1 \delta \delta h))$

using $FH \text{ ind}V \text{ induced-smult.fsmultD Supgroup.RG-aezdeltafun-closed[of } h 1::'f]$

$\text{rrsmultD1 aezdeltafun-decomp'[of } r h]$

$\text{aezdeltafun-decomp[of } r h] \text{ Supgroup.RG-aezdeltafun-closed[of } h 1::'f]$

Group.zero-closed[*OF GroupG*]
BaseFGMod.indspaceD-transform[*of f G FH V 0 (1::f) δδ h r*]
BaseFGMod.fsmultD

by *simp*

lemma *indspace-el-eq-on-1ddh-imp-eq-on-rddh* :

assumes $H\text{mod}G \subseteq H$ $H = (\bigcup h \in H\text{mod}G. G + \{h\})$ $f \in \text{ind}V$ $f' \in \text{ind}V$
 $\forall h \in H\text{mod}G. f(1 \ \delta\delta \ h) = f'(1 \ \delta\delta \ h)$ $h \in H$

shows $f(r \ \delta\delta \ h) = f'(r \ \delta\delta \ h)$

proof –

from *assms*(2,6) **obtain** h' **where** $h': h' \in H\text{mod}G$ $h \in G + \{h'\}$ **by** *fast*

from $h'(2)$ **obtain** g **where** $g: g \in G$ $h = g + h'$

using *set-plus-def*[*of G*] **by** *auto*

from $g(2)$ **have** $r \ \delta\delta \ h = r \ \delta\delta \ 0 * (1 \ \delta\delta \ (g+h'))$

using *aezdeltafun-decomp* **by** *simp*

moreover **have** $(1::f) \ \delta\delta \ (g+h') = 1 \ \delta\delta \ g * (1 \ \delta\delta \ h')$

using *times-aezdeltafun-aezdeltafun*[*of 1::f, THEN sym*] **by** *simp*

ultimately **have** $r \ \delta\delta \ h = r \ \delta\delta \ g * (1 \ \delta\delta \ h')$

using *aezdeltafun-decomp*[*of r g*]

by (*simp add: algebra-simps*)

with $\text{ind}V$ FH *assms*(1,3,4) $g(1)$ $h'(1)$

have $f(r \ \delta\delta \ h) = r \ \delta\delta \ g \cdot f(1 \ \delta\delta \ h')$ $f'(r \ \delta\delta \ h) = r \ \delta\delta \ g \cdot f'(1 \ \delta\delta \ h')$

using *Supgroup.RG-aezdeltafun-closed*[*of h' 1*]

BaseFGMod.indspaceD-transform[*of f G FH V g 1 δδ h' r*]

BaseFGMod.indspaceD-transform[*of f' G FH V g 1 δδ h' r*]

by *auto*

thus $f(r \ \delta\delta \ h) = f'(r \ \delta\delta \ h)$ **using** $h'(1)$ *assms*(5) **by** *simp*

qed

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 $f = f'$

proof

fix x **show** $f x = f' x$

proof (*cases* $x = 0$ $x \in FH$ *rule: conjcases*)

case *BothTrue*

hence $x = 0$ $\delta\delta \ 0$ **using** *zero-aezfun-transfer* **by** *simp*

with *assms* **show** *?thesis*

using *indspace-el-eq-on-1ddh-imp-eq-on-rddh*[*of HmodG f f'*] *Supgroup.zero-closed*

by *auto*

next

case *OneTrue* **with** FH **show** *?thesis* **using** *Supgroup.RG-zero-closed* **by** *fast*

next

case *OtherTrue*

with FH **obtain** rhs

where $rhs: \text{set}(\text{map} \ \text{snd} \ rhs) \subseteq H$ $x = (\sum (r,h) \leftarrow rhs. r \ \delta\delta \ h)$

using *Supgroup.RG-el-decomp-aezdeltafun*

by *fast*

```

from OtherTrue rhs(2) have rhs-nnil: rhs ≠ [] by auto
with assms(3,4) rhs
  have  $f\ x = (\sum (r,h) \leftarrow \text{rhs}. f\ (r\ \delta\delta\ h))\ f'\ x = (\sum (r,h) \leftarrow \text{rhs}. f'\ (r\ \delta\delta\ h))$ 
  using indspace-sum-list-fddh
  by auto
moreover from rhs(1) assms have  $\forall (r,h) \in \text{set}\ \text{rhs}. f\ (r\ \delta\delta\ h) = f'\ (r\ \delta\delta\ h)$ 
  using indspace-el-eq-on-1ddh-imp-eq-on-rddh[of HmodG f f'] by fastforce
ultimately show ?thesis
  using sum-list-prod-cong[of rhs  $\lambda r\ h. f\ (r\ \delta\delta\ h)$ ] by simp
next
  case BothFalse
  with indV assms(3,4) show ?thesis
    using BaseFGMod.indspaceD-suppl[of f] BaseFGMod.indspaceD-suppl[of f']
    by simp
qed
qed

```

```

lemma indspace-el-eq' :
  assumes  $\text{set}\ \text{hs} \subseteq H\ H = (\bigcup h \in \text{set}\ \text{hs}. G + \{h\})\ f \in \text{indV}\ f' \in \text{indV}$ 
     $\forall i < \text{length}\ \text{hs}. f\ (1\ \delta\delta\ (\text{hs}!i)) = f'\ (1\ \delta\delta\ (\text{hs}!i))$ 
  shows  $f = f'$ 
  using assms(1-4)
proof (rule indspace-el-eq[of set hs])
  have  $\bigwedge h. h \in \text{set}\ \text{hs} \implies f\ (1\ \delta\delta\ h) = f'\ (1\ \delta\delta\ h)$ 
  proof -
    fix h assume  $h \in \text{set}\ \text{hs}$ 
    from this obtain i where  $i < \text{length}\ \text{hs}\ h = \text{hs}!i$ 
    using in-set-conv-nth[of h] by fast
    with assms(5) show  $f\ (1\ \delta\delta\ h) = f'\ (1\ \delta\delta\ h)$  by simp
  qed
  thus  $\forall h \in \text{set}\ \text{hs}. f\ (1\ \delta\delta\ h) = f'\ (1\ \delta\delta\ h)$  by fast
qed
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::group-add set
  and smult :: ('f::field, 'g) aezfun  $\implies$  'v::ab-group-add  $\implies$  'v (infixl · 70)

```

```

and  $V$     :: 'v set
+
assumes good-char: of-nat (card  $G$ )  $\neq$  ( $0::'f$ )
and    findim  : fscalar-mult.findim fsmult  $V$ 

lemma (in Group) trivial-FinGroupRep :
  fixes smult    :: ('f::field, 'g) aezfun  $\Rightarrow$  'v::ab-group-add  $\Rightarrow$  'v
  assumes good-char : of-nat (card  $G$ )  $\neq$  ( $0::'f$ )
  and    smult-zero :  $\forall a \in \text{group-ring. smult } a \text{ } (0::'v) = 0$ 
  shows FinGroupRepresentation  $G$  smult ( $0::'v$  set)
proof (rule FinGroupRepresentation.intro)
  from smult-zero show FGModule  $G$  smult ( $0::'v$  set)
  using trivial-FGModule by fast

  have fscalar-mult.findim (aezfun-scalar-mult.fsmult smult)  $0$ 
  by auto (metis R-scalar-mult.RSpan.simps(1) aezfun-scalar-mult.R-scalar-mult
empty-set empty-subsetI set-zero)

  with good-char show FinGroupRepresentation-axioms  $G$  smult  $0$  by unfold-locales
qed

context FinGroupRepresentation
begin

abbreviation ordG :: 'f where ordG  $\equiv$  of-nat (card  $G$ )
abbreviation GRepHom  $\equiv$  FGModuleHom  $G$  smult  $V$ 
abbreviation GRepIso  $\equiv$  FGModuleIso  $G$  smult  $V$ 
abbreviation GRepEnd  $\equiv$  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  $G$ 
  using good-char good-card-imp-finite by fast

lemma FinDimVectorSpace: FinDimVectorSpace fsmult  $V$ 
  using findim fVectorSpace VectorSpace.FinDimVectorSpaceI by fast

lemma GSubspace-is-FinGroupRep :
  assumes GSubspace  $U$ 
  shows FinGroupRepresentation  $G$  smult  $U$ 
proof (
  rule FinGroupRepresentation.intro, rule GSubspace-is-FGModule[OF assms], un-
fold-locales
)
from assms show fscalar-mult.findim fsmult  $U$ 
using FinDimVectorSpace GSubspace-is-Subspace FinDimVectorSpace.Subspace-is-findim

```

by fast
qed (rule good-char)

lemma *isomorphic-imp-GRep* :
 assumes *isomorphic smult' W*
 shows *FinGroupRepresentation G smult' W*
 proof (rule *FinGroupRepresentation.intro*)
 from *assms* show *FGModule G smult' W*
 using *FGModuleIso.ImG FGModuleHom.FGModule-Im[OF FGModuleIso.axioms(1)]*
 by fast
 from *assms* have *fscalar-mult.findim (aezfun-scalar-mult.fsmult smult') W*
 using *FGModuleIso.ImG findim FGModuleIso.VectorSpaceHom*
 VectorSpaceHom.findim-domain-findim-image
 by fastforce
 with *good-char* show *FinGroupRepresentation-axioms G smult' W* by *unfold-locales*
 qed
 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
 show *GRepIso (*) 0 0*
 proof (rule *FGModuleIso.intro*)
 show *GRepHom (*) 0* using *trivial-FGModuleHom[of (*)]* by *simp*
 show *FGModuleIso-axioms V 0 0*
 proof
 from *assms* show *bij-betw 0 V 0* unfolding *bij-betw-def inj-on-def* by *simp*
 qed
 qed
 qed

lemma *eq-GSpan-single* : *v \in V \implies v \neq 0 \implies V = GSpan [v]*
 using *irr RSubmodule-RSpan-single RSpan-single-nonzero* by *fast*

end

lemma (in *Group*) *trivial-IrrFinGroupRepI* :
 fixes *smult :: ('f::field, 'g) aezfun \Rightarrow 'v::ab-group-add \Rightarrow 'v*
 assumes *of-nat (card G) \neq (0::'f)*

and $\forall a \in \text{group-ring. smult } a \ (0::'v) = 0$
shows $\text{IrrFinGroupRepresentation } G \ \text{smult } (0::'v \ \text{set})$
proof (rule $\text{IrrFinGroupRepresentation.intro}$)
from $\text{assms show FinGroupRepresentation } G \ \text{smult } 0$
using $\text{trivial-FinGroupRep}$ **by** fast
show $\text{IrrFinGroupRepresentation-axioms } G \ \text{smult } 0$
using $\text{RModule.zero-closed}$ **by** $\text{unfold-locales auto}$
qed

lemma (in Group) $\text{trivial-IrrFinGroupRepresentation-in-FG} :$
 $\text{of-nat } (\text{card } G) \neq (0::'f::\text{field})$
 $\implies \text{IrrFinGroupRepresentation } G \ (*) \ (0::('f,'g) \ \text{aezfun set})$
using $\text{trivial-IrrFinGroupRepI[of (*)]}$ **by** simp

context $\text{FinGroupRepresentation}$
begin

lemma $\text{IrrFinGroupRep-trivialGSubspace} :$
 $\text{IrrFinGroupRepresentation } G \ \text{smult } (0::'v \ \text{set})$
proof –
have $\text{ord } G \neq (0::'f)$ **using** good-char **by** fast
moreover **have** $\forall a \in \text{FG. } a \cdot 0 = 0$ **using** smult-zero **by** simp
ultimately show $?thesis$
using $\text{ActingGroup.trivial-IrrFinGroupRepI[of smult]}$ **by** fast
qed

lemma $\text{IrrI} :$
assumes $\bigwedge U. \ \text{FGModule.GSubspace } G \ \text{smult } V \ U \implies U = 0 \vee U = V$
shows $\text{IrrFinGroupRepresentation } G \ \text{smult } V$
using $\text{assms IrrFinGroupRepresentation.intro}$ **by** unfold-locales

lemma $\text{notIrr} :$
 $\neg \text{IrrFinGroupRepresentation } G \ \text{smult } V$
 $\implies \exists U. \ \text{GSubspace } U \wedge U \neq 0 \wedge U \neq V$
using IrrI **by** fast

end

6.3 Maschke's theorem

6.3.1 Averaged projection onto a G-subspace

context $\text{FinGroupRepresentation}$
begin

lemma $\text{avg-proj-eq-id-on-right} :$
assumes $\text{VectorSpace fsmult } W \ \text{add-independentS } [W, V] \ v \in V$
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 –
from P *assms*(2,3) **have** $\bigwedge g. g \in G \implies P (g * v) = g * v$
using *Gmult-closed VectorSpace.AbGroup[OF assms(1)] AbGroup*
AbGroup-inner-dirsum-el-decomp-nth-id-on-nth[of [W, V]]
by *simp*
with CP *assms*(3) **have** $\bigwedge g. g \in G \implies CP g v = v$
using *Gmult-neg-left by simp*
with *assms*(3) *good-char T* **show** $T v = v$
using *finiteG sum-fun-apply[of G CP] sum-fsmult-distrib[of v G $\lambda x. 1$]*
fsmult-assoc[of - ordG v]
by *simp*
qed

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 $CP : CP \equiv (\lambda g. Gmult (- g) \circ P \circ Gmult g)$
defines $T : T \equiv fsmult (1/ordG) \circ (\sum_{g \in G}. CP g)$
shows $T ' V = U$

proof
from *assms*(2) **have** $U : FGModule G smult U$
using *GSubspace-is-FGModule by fast*
show $T ' V \subseteq U$
proof (*rule image-subsetI*)
fix v **assume** $v : v \in V$
with *assms*(1,3,4) $P U$ **have** $\bigwedge g. g \in G \implies P (g * v) \in U$
using *Gmult-closed VectorSpace.AbGroup FGModule.AbGroup*
AbGroup-inner-dirsum-el-decomp-nth-onto-nth[of [W, U] 1]
by *fastforce*
with $U CP$ **have** $\bigwedge g. g \in G \implies CP g v \in U$
using *FGModule.Gmult-closed GroupG Group.neg-closed by fastforce*
with *assms*(2) $U T$ **show** $T v \in U$
using *finiteG FGModule.sum-closed[of G smult U G $\lambda g. CP g v$]*
sum-fun-apply[of G CP] FGModule.fsmult-closed[of G smult U]
by *fastforce*

qed
show $T ' V \supseteq U$
proof
fix u **assume** $u : u \in U$
with $u T CP P$ *assms*(1,2,3) **have** $T u = u$
using *GSubspace-is-FinGroupRep FinGroupRepresentation.avg-proj-eq-id-on-right*
by *fast*
from *this[THEN sym] assms*(1–4) u **show** $u \in T ' V$
using *Module.AbGroup RModule.AbGroup AbGroup-subset-inner-dirsum*
by *fast*
qed
qed

lemma *FGModuleEnd-avg-proj-right* :

assumes *fSubspace W GSubspace U add-independentS [W,U] V = W \oplus U*

defines *P* : $P \equiv (\bigoplus [W,U] \downarrow 1)$

defines *CP*: $CP \equiv (\lambda g. Gmult (- g) \circ P \circ Gmult g)$

defines *T* : $T \equiv (fsmult (1/ordG) \circ (\sum_{g \in G}. CP g)) \downarrow V$

shows *FGModuleEnd G smult V T*

proof (rule *VecEnd-GMap-is-FGModuleEnd*)

from *T* **have** *T-apply*: $\bigwedge v. v \in V \implies T v = (1/ordG) \# (\sum_{g \in G}. CP g v)$

using *finiteG sum-fun-apply[of G CP]* **by** *simp*

from *assms(1-4)* *P* **have** *Pgv*: $\bigwedge g v. g \in G \implies v \in V \implies P (g * v) \in V$

using *Gmult-closed VectorSpace-fSubspace VectorSpace.AbGroup[of fsmult W]*
RModule.AbGroup[of FG smult U]
GroupEnd-inner-dirsum-el-decomp-nth[of [W,U] 1]
GroupEnd.endomorph[of V]

by *force*

have *im-CP-V*: $\bigwedge v. v \in V \implies (\lambda g. CP g v) ' G \subseteq V$

proof–

fix *v* **assume** $v \in V$ **thus** $(\lambda g. CP g v) ' G \subseteq V$

using *CP Pgv[of - v] ActingGroup.neg-closed Gmult-closed finiteG* **by** *auto*

qed

have *sumCP-V*: $\bigwedge v. v \in V \implies (\sum_{g \in G}. CP g v) \in V$

using *finiteG im-CP-V sum-closed* **by** *force*

show *VectorSpaceEnd (#.) V T*

proof (

rule VectorSpaceEndI, rule VectorSpace.VectorSpaceHomI, rule fVectorSpace

)

show *GroupHom V T*

proof

fix *v v'* **assume** $vv': v \in V v' \in V$

with *T-apply* **have** $T (v + v') = (1/ordG) \# (\sum_{g \in G}. CP g (v + v'))$

using *add-closed* **by** *fast*

moreover **have** $\bigwedge g. g \in G \implies CP g (v + v') = CP g v + CP g v'$

proof–

fix *g* **assume** $g \in G$

with *CP P vv' assms(1-4)*

have $CP g (v + v') = (- g) * (P (g * v) + P (g * v'))$

using *Gmult-distrib-left Gmult-closed VectorSpace-fSubspace*
VectorSpace.AbGroup[of fsmult W] RModule.AbGroup[of FG smult U]
GroupEnd-inner-dirsum-el-decomp-nth[of [W,U] 1]
GroupEnd.hom[of V P]

by *simp*

with *g vv'* **have** $CP g (v + v')$

$= (-g) * P(g * v) + (-g) * P(g * v')$

using *Pgv ActingGroup.neg-closed Gmult-distrib-left* **by** *simp*
thus $CP\ g\ (v + v') = CP\ g\ v + CP\ g\ v'$ **using** *CP* **by** *simp*
qed

ultimately show $T\ (v + v') = T\ v + T\ v'$
using *vv' sumCP-V[of v] sumCP-V[of v'] sum.distrib[of λg. CP g v]*
T-apply
by *simp*

next
from T **show** $supp\ T \subseteq V$ **using** *supp-restrict0* **by** *fast*
qed

show $\bigwedge a\ v.\ v \in V \implies T\ (a \# \cdot v) = a \# \cdot T\ v$
proof–
fix $a::'f$ **and** v **assume** $v: v \in V$
with *T-apply* **have** $T\ (a \# \cdot v) = (1/ordG) \# \cdot (\sum_{g \in G}. CP\ g\ (a \# \cdot v))$
using *fsmult-closed* **by** *fast*
moreover **have** $\bigwedge g.\ g \in G \implies CP\ g\ (a \# \cdot v) = a \# \cdot CP\ g\ v$
proof–
fix g **assume** $g \in G$
with *assms(1-4) CP P v* **show** $CP\ g\ (a \# \cdot v) = a \# \cdot CP\ g\ v$
using *fsmult-Gmult-comm GSubspace-is-Subspace Gmult-closed fVectorSpace*
VectorSpace.VectorSpaceEnd-inner-dirsum-el-decomp-nth[of fsmult]
*VectorSpaceEnd.f-map[of fsmult (⊕ N←[W, U]. N) P a g * v]*
ActingGroup.neg-closed Pgv
by *simp*
qed

ultimately show $T\ (a \# \cdot v) = a \# \cdot T\ v$
using *v im-CP-V sumCP-V T-apply finiteG*
fsmult-sum-distrib[of a G λg. CP g v]
fsmult-assoc[of 1/ordG a (∑ g ∈ G. CP g v)]
by *simp*
qed

show $T^{-1}\ V \subseteq V$ **using** *sumCP-V fsmult-closed T-apply image-subsetI* **by** *auto*

qed

show $\bigwedge g\ v.\ g \in G \implies v \in V \implies T\ (g * \cdot v) = g * \cdot T\ v$
proof–
fix $g\ v$ **assume** $g: g \in G$ **and** $v: v \in V$
with *T-apply* **have** $T\ (g * \cdot v) = (1/ordG) \# \cdot (\sum_{h \in G}. CP\ h\ (g * \cdot v))$
using *Gmult-closed* **by** *fast*
with g **have** $T\ (g * \cdot v) = (1/ordG) \# \cdot (\sum_{h \in G}. CP\ (h + -g)\ (g * \cdot v))$
using *GroupG Group.neg-closed*
*Group.right-translate-sum[of G - g λh. CP h (g * v)]*
by *fastforce*
moreover from *CP*
have $\bigwedge h.\ h \in G \implies CP\ (h + -g)\ (g * \cdot v) = g * \cdot CP\ h\ v$

using $g v$ *Gmult-closed*[of $g v$] *ActingGroup.neg-closed*
Gmult-assoc[of $- - g g * v$, *THEN sym*]
Gmult-neg-left minus-add[of $- - g$] *Pgv Gmult-assoc*
by *simp*
ultimately show $T (g * v) = g * T v$
using $g v$ *GmultD finiteG FG-fddg-closed im-CP-V sumCP-V*
smult-sum-distrib[of $1 \delta \delta g G$]
fsmult-Gmult-comm[of $g \sum_{h \in G}. (CP h v)$] *T-apply*
by *simp*
qed

qed

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 $T (T v) = T v$
using *assms avg-proj-onto-right GSubspace-is-FinGroupRep*
FinGroupRepresentation.avg-proj-eq-id-on-right
by *fast*

end

6.3.2 The theorem

context *FinGroupRepresentation*
begin

theorem *Maschke* :

assumes *GSubspace U*
shows $\exists W. GSubspace W \wedge V = W \oplus U$
proof (*cases* $V = 0$)
case *True*
moreover from *assms True* **have** $U = 0$ **using** *RModule.zero-closed* **by** *auto*
ultimately have $V = 0 + U$ **using** *set-plus-def* **by** *fastforce*
moreover have *add-independentS* [$0, U$] **by** *simp*
ultimately have $V = 0 \oplus U$ **using** *inner-dirsum-doubleI* **by** *fast*
moreover have *GSubspace 0* **using** *trivial-RSubmodule zero-closed* **by** *auto*
ultimately show $\exists W. GSubspace W \wedge V = W \oplus U$ **by** *fast*
next
case *False*
from *assms* **obtain** W'
where W' : *VectorSpace.Subspace fsmult V* $W' \wedge V = W' \oplus U$
using *GSubspace-is-Subspace FinDimVectorSpace FinDimVectorSpace.semisimple*
by *force*
hence *vsp-W'* : *VectorSpace fsmult W'* **and** *dirsum* : $V = W' \oplus U$

using *VectorSpace.SubspaceD1*[*OF fVectorSpace*] **by** *auto*
from *False dirsum* **have** *indS: add-independentS* [*W',U*]
using *inner-dirsumD2* **by** *fastforce*
define *P* **where** $P = (\bigoplus [W',U] \downarrow 1)$
define *CP* **where** $CP = (\lambda g. Gmult (- g) \circ P \circ Gmult g)$
define *S* **where** $S = fsmult (1/ordG) \circ (\sum_{g \in G}. CP g)$
define *W* **where** $W = GroupHom.Ker V (S \downarrow V)$
from *assms W' indS S-def CP-def P-def* **have** *endo: GRepEnd* ($S \downarrow V$)
using *FGModuleEnd-avg-proj-right* **by** *fast*
moreover from *S-def CP-def P-def* **have** $\bigwedge v. v \in V \implies (S \downarrow V) ((S \downarrow V) v) = (S \downarrow V) v$
using *endo FGModuleEnd.endomorph*
avg-proj-is-proj-right[*OF vsp-W' assms indS dirsum*]
by *fastforce*
moreover have $(S \downarrow V) ' V = U$
proof-
from *assms indS P-def CP-def S-def dirsum vsp-W'* **have** $S ' V = U$
using *avg-proj-onto-right* **by** *fast*
moreover have $(S \downarrow V) ' V = S ' V$ **by** *auto*
ultimately show *?thesis* **by** *fast*
qed
ultimately have $V = W \oplus U$ *GSubspace W*
using *W-def FGModuleEnd.proj-decomp*[*of G smult V S \downarrow V*]
FGModuleEnd.GSubspace-Ker
by *auto*
thus *?thesis* **by** *fast*
qed

corollary *Maschke-proper* :

assumes *GSubspace U U* $U \neq 0$ $U \neq V$
shows $\exists W. GSubspace W \wedge W \neq 0 \wedge W \neq V \wedge V = W \oplus U$

proof-

from *assms(1)* **obtain** *W* **where** $W: GSubspace W V = W \oplus U$
using *Maschke* **by** *fast*
from *assms(3)* *W(2)* **have** $W \neq 0$ **using** *inner-dirsum-double-left0* **by** *fast*
moreover from *assms(1,2)* *W* **have** $W \neq V$
using *Subgroup-RSubmodule Group.nonempty*
inner-dirsum-double-leftfull-imp-right0[*of W U*]
by *fastforce*
ultimately show *?thesis* **using** *W* **by** *fast*
qed

end

6.3.3 Consequence: complete reducibility

lemma (in *FinGroupRepresentation*) *notIrr-decompose* :

$\neg IrrFinGroupRepresentation G smult V$
 $\implies \exists U W. GSubspace U \wedge U \neq 0 \wedge U \neq V \wedge GSubspace W \wedge W \neq 0$

$\wedge W \neq V \wedge V = U \oplus W$
using *notIrr Maschke-proper* **by** *blast*

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 $n::\text{nat}$

shows $\wedge V. \text{FinGroupRepresentation } G \text{ fgsmult } V$
 $\wedge n = \text{FGModule.fdim fgsmult } V$
 $\implies (\exists Us. \text{Ball } (\text{set } Us) (\text{IrrFinGroupRepresentation } G \text{ fgsmult})$
 $\wedge (0 \notin \text{set } Us) \wedge V = (\bigoplus U \leftarrow Us. U))$

proof (*induct n rule: full-nat-induct*)

fix $n V$

define $G\text{Rep } IG\text{Rep } G\text{Subspace } G\text{Span } \text{fdim}$

where $G\text{Rep} = \text{FinGroupRepresentation } G \text{ fgsmult}$
and $IG\text{Rep} = \text{IrrFinGroupRepresentation } G \text{ fgsmult}$
and $G\text{Subspace} = \text{FGModule.GSubspace } G \text{ fgsmult } V$
and $G\text{Span} = \text{FGModule.GSpan } G \text{ fgsmult}$
and $\text{fdim} = \text{FGModule.fdim fgsmult}$

assume $\forall m. \text{Suc } m \leq n \implies (\forall x. G\text{Rep } x \wedge m = \text{fdim } x \implies (\exists Us.$
 $\text{Ball } (\text{set } Us) IG\text{Rep} \wedge (0 \notin \text{set } Us) \wedge x = (\bigoplus U \leftarrow Us. U)))$

hence *prev-case*:

$\wedge m. m < n \implies (\wedge W. G\text{Rep } W \implies m = \text{fdim } W \implies (\exists Us.$
 $\text{Ball } (\text{set } Us) IG\text{Rep} \wedge (0 \notin \text{set } Us) \wedge W = (\bigoplus U \leftarrow Us. U)))$

using *Suc-le-eq* **by** *auto*

show $G\text{Rep } V \wedge n = \text{fdim } V \implies (\exists Us.$

$\text{Ball } (\text{set } Us) IG\text{Rep} \wedge (0 \notin \text{set } Us) \wedge V = (\bigoplus U \leftarrow Us. U))$

proof–

assume $V: G\text{Rep } V \wedge n = \text{fdim } V$

show $(\exists Us. \text{Ball } (\text{set } Us) IG\text{Rep} \wedge (0 \notin \text{set } Us) \wedge V = (\bigoplus U \leftarrow Us. U))$

proof (*cases IGRep V V = 0 rule: conjcases*)

case *BothTrue*

moreover have $\text{Ball } (\text{set } []) IG\text{Rep} \forall U \in \text{set } []. U \neq 0$ **by** *auto*

ultimately show *?thesis* **using** *inner-dirsum-Nil* **by** *fast*

next

case *OneTrue*

with $V G\text{Rep-def}$ **obtain** v **where** $v: v \in V v \neq 0$

using *FinGroupRepresentation.Group[of G fgsmult]* *Group.obtain-nonzero*

by *auto*

from $v(1) V G\text{Rep-def } G\text{Span-def } G\text{Subspace-def}$ **have** $G\text{Sub}: G\text{Subspace}$
 $(G\text{Span } [v])$

using *FinGroupRepresentation.GSubmodule-GSpan-single* **by** *fast*

moreover from $v V G\text{Rep-def } G\text{Span-def}$ **have** $n\text{zero}: G\text{Span } [v] \neq 0$

using *FinGroupRepresentation.GSpan-single-nonzero* **by** *fast*

ultimately have $V = G\text{Span } [v]$

using *OneTrue GSpan-def GSubspace-def IGRep-def IrrFinGroupRepresentation.irr*

by *fast*

```

with OneTrue
  have Ball (set [GSpan [v]]) IGRep 0 ∉ set [GSpan [v]]
    V = (⊕ U←[GSpan [v]]. U)
  using nzero GSub inner-dirsum-singleD
  by auto
thus ?thesis by fast
next
case OtherTrue with V GRep-def IGRep-def show ?thesis
  using FinGroupRepresentation.IrrFinGroupRep-trivialGSubspace by fast
next
case BothFalse
with V GRep-def IGRep-def GSubspace-def obtain U W
  where U: GSubspace U U ≠ 0 U ≠ V
  and W: GSubspace W W ≠ 0 W ≠ V
  and Vdecompose: V = U ⊕ W
  using FinGroupRepresentation.notIrr-decompose[of G fgsmult V]
  by auto
from U(1,3) W(1,3) V GRep-def GSubspace-def fdim-def
  have fdim U < fdim V fdim W < fdim V
  using FinGroupRepresentation.axioms(1)
    FGModule.GSubspace-is-Subspace[of G fgsmult V U]
    FGModule.GSubspace-is-Subspace[of G fgsmult V W]
    FinGroupRepresentation.FinDimVectorSpace[of G fgsmult V]
    FinDimVectorSpace.Subspace-dim-lt[
      of aezfun-scalar-mult.fsmult fgsmult V U
    ]
  ]
    FinDimVectorSpace.Subspace-dim-lt[
      of aezfun-scalar-mult.fsmult fgsmult V W
    ]
  ]
  by auto
from this U(1) W(1) V GSubspace-def obtain Us Ws
  where Ball (set Us) IGRep ∧ (0 ∉ set Us) ∧ U = (⊕ X←Us. X)
  and Ball (set Ws) IGRep ∧ (0 ∉ set Ws) ∧ W = (⊕ X←Ws. X)
  using prev-case[of fdim U] prev-case[of fdim W] GRep-def
    FinGroupRepresentation.GSubspace-is-FinGroupRep[
      of G fgsmult V U
    ]
  ]
    FinGroupRepresentation.GSubspace-is-FinGroupRep[
      of G fgsmult V W
    ]
  ]
  by fastforce
hence Us: Ball (set Us) IGRep 0 ∉ set Us U = (⊕ X←Us. X)
  and Ws: Ball (set Ws) IGRep 0 ∉ set Ws W = (⊕ X←Ws. X)
  by auto
from Us(1) Ws(1) have Ball (set (Us@Ws)) IGRep by auto
moreover from Us(2) Ws(2) have 0 ∉ set (Us@Ws) by auto
moreover have V = (⊕ X←(Us@Ws). X)
proof-
  from U(2) Us(3) W(2) Ws(3)

```

have $indUs$: *add-independentS* Us
and $indWs$: *add-independentS* Ws
using *inner-dirsumD2*
by *auto*
moreover from *IGRep-def* $Us(1)$ **have** *Ball* (*set* Us) ($(\in) 0$)
using *IrrFinGroupRepresentation.axioms(1)*[*of* G *fgsmult*]
FinGroupRepresentation.zero-closed[*of* G *fgsmult*]
by *fast*
moreover from $Us(3)$ $Ws(3)$ *BothFalse* *Vdecompose* $indUs$ $indWs$
have *add-independentS* $[(\sum X \leftarrow Us. X), (\sum X \leftarrow Ws. X)]$
using *inner-dirsumD2*[*of* $[U, W]$] *inner-dirsumD*[*of* Us]
inner-dirsumD[*of* Ws]
by *auto*
ultimately have *add-independentS* ($Us@Ws$)
by (*metis add-independentS-double-sum-conv-append*)
moreover from $W(1)$ $Ws(3)$ $indWs$ **have** $0 \in (\sum X \leftarrow Ws. X)$
using *inner-dirsumD* *GSubspace-def* *RModule.zero-closed* **by** *fast*
ultimately show *?thesis*
using *Vdecompose* $Us(3)$ $Ws(3)$ *inner-dirsum-append* **by** *fast*
qed
ultimately show *?thesis* **by** *fast*
qed
qed
qed

theorem (in *FinGroupRepresentation***)** *reducible* :
 $\exists Us. (\forall U \in set\ Us. IrrFinGroupRepresentation\ G\ smult\ U) \wedge (0 \notin set\ Us)$
 $\wedge V = (\bigoplus U \leftarrow Us. U)$
using *FinGroupRepresentation-axioms* *FinGroupRepresentation-reducible'* **by** *fast*

6.3.4 Consequence: decomposition relative to a homomorphism

lemma (in *FinGroupRepresentation***)** *GRepHom-decomp* :
fixes $T :: 'v \Rightarrow 'w :: ab\ group\ add$
defines $KerT : KerT \equiv (ker\ T \cap V)$
assumes *hom* : *GRepHom* $smult'$ T **and** *nonzero*: $V \neq 0$
shows $\exists U. GSubspace\ U \wedge V = U \oplus KerT$
 $\wedge FGModule.isomorphic\ G\ smult\ U\ smult'\ (T \downarrow V)$

proof –

from $KerT$ **have** $KerT'$: *GSubspace* $KerT$
using *FGModuleHom.GSubspace-Ker*[*OF* *hom*] **by** *fast*
from this obtain U **where** U : *GSubspace* U $V = U \oplus KerT$
using *Maschke*[*of* $KerT$] **by** *fast*
from *nonzero* $U(2)$ **have** *indep*: *add-independentS* $[U, KerT]$
using *inner-dirsumD2* **by** *fastforce*
have *FGModuleIso* $G\ smult\ U\ smult'\ (T \downarrow U)\ (T \downarrow V)$
proof (*rule* *FGModuleIso.intro*)
from $U(1)$ **show** *FGModuleHom* $G\ (\cdot)\ U\ smult'\ (T \downarrow U)$
using *FGModuleHom.FGModuleHom-restrict0-GSubspace*[*OF* *hom*] **by** *fast*


```

show FGModuleIso-axioms  $U (T \downarrow U) (T \text{ ' } V)$ 
  unfolding FGModuleIso-axioms-def bij-betw-def
proof (rule conjI, rule inj-onI)
  show  $(T \downarrow U) \text{ ' } U = T \text{ ' } V$ 
  proof
    from  $U(1)$  show  $(T \downarrow U) \text{ ' } U \subseteq T \text{ ' } V$  by auto
    show  $(T \downarrow U) \text{ ' } U \supseteq T \text{ ' } V$ 
    proof (rule image-subsetI)
      fix  $v$  assume  $v \in V$ 
      with  $U(2)$  obtain  $u k$  where  $uk: u \in U k \in \text{Ker}T v = u + k$ 
      using inner-dirsum-doubleD[OF indep] set-plus-def[of U KerT] by fast
      with  $\text{Ker}T U(1)$  have  $T v = (T \downarrow U) u$ 
      using kerD FGModuleHom.additive[OF hom] by force
      with  $uk(1)$  show  $T v \in (T \downarrow U) \text{ ' } U$  by fast
    qed
  qed
next
fix  $x y$  assume  $xy: x \in U y \in U (T \downarrow U) x = (T \downarrow U) y$ 
with  $U(1) \text{Ker}T$  have  $x - y \in U \cap \text{Ker}T$ 
  using FGModuleHom.eq-im-imp-diff-in-Ker[OF hom]
  GSubspace-is-FGModule FGModule.diff-closed[of G smult U x y]
  by auto
moreover from  $U(1)$  have AbGroup  $U$  using RModule.AbGroup by fast
moreover from  $\text{Ker}T'$  have AbGroup  $\text{Ker}T$ 
  using RModule.AbGroup by fast
ultimately show  $x = y$ 
  using indep AbGroup-inner-dirsum-pairwise-int0-double[of U KerT]
  by force
qed
qed
with  $U$  show ?thesis by fast
qed

```

6.4 Schur's lemma

lemma (in *IrrFinGroupRepresentation*) *Schur-Ker* :

```

GRepHom smult' T  $\implies T \text{ ' } V \neq 0 \implies \text{inj-on } T V$ 
using irr FGModuleHom.GSubspace-Ker[of G smult V smult' T]
  FGModuleHom.Ker-Im-iff[of G smult V smult' T]
  FGModuleHom.Ker0-imp-inj-on[of G smult V smult' T]
by auto

```

lemma (in *FinGroupRepresentation*) *Schur-Im* :

```

assumes IrrFinGroupRepresentation G smult' W GRepHom smult' T
   $T \text{ ' } V \subseteq W$ 
   $T \text{ ' } V \neq 0$ 
shows  $T \text{ ' } V = W$ 
proof -
  have FGModule.GSubspace  $G smult' W (T \text{ ' } V)$ 

```

proof
from *assms(2)* **show** *RModule FG smult' (T ' V)*
using *FGModuleHom.axioms(2)[of G]*
RModuleHom.RModule-Im[of FG smult V smult' T]
by *fast*
from *assms(3)* **show** $T ' V \subseteq W$ **by** *fast*
qed
with *assms(1,4)* **show** *?thesis* **using** *IrrFinGroupRepresentation.irr* **by** *fast*
qed

theorem (in *IrrFinGroupRepresentation*) *Schur1* :
assumes *IrrFinGroupRepresentation G smult' W*
GRepHom smult' T T ' V \subseteq W T ' V \neq 0
shows *GRepIso smult' T W*
proof (rule *FGModuleIso.intro*, rule *assms(2)*, *unfold-locales*)
show *bij-betw T V W*
using *IrrFinGroupRepresentation-axioms assms*
IrrFinGroupRepresentation.axioms(1)[of G]
FinGroupRepresentation.Schur-Im[of G]
IrrFinGroupRepresentation.Schur-Ker[of G smult V smult' T]
unfolding *bij-betw-def*
by *fast*
qed

theorem (in *IrrFinGroupRepresentation*) *Schur2* :
assumes *GRep : GRepEnd T*
and *fdim : fdim > 0*
and *alg-closed: $\bigwedge p: 'b$ poly. degree $p > 0 \implies \exists c. poly p c = 0$*
shows $\exists c. \forall v \in V. T v = c \# \cdot v$
proof –
from *fdim alg-closed* **obtain** *e u* **where** *eu: $u \in V u \neq 0 T u = e \# \cdot u$*
using *FGModuleEnd.VectorSpaceEnd[OF GRep]*
FinDim VectorSpace.endomorph-has-eigenvector[
OF FinDim VectorSpace, of T
]
by *fast*
define *U* **where** $U = \{v \in V. T v = e \# \cdot v\}$
moreover **from** *eu U-def* **have** $U \neq 0$ **by** *auto*
ultimately **have** $\forall v \in V. T v = e \# \cdot v$
using *GRep irr FGModuleEnd.axioms(1)[of G smult V T]*
GSubspace-eigenspace[of G smult]
by *fast*
thus *?thesis* **by** *fast*
qed

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 FG:  $FG \equiv \text{group-ring} :: ('f::\text{field}, 'g) \text{ aezfun set}$ 
shows FGModule G (*) FG
proof (rule FGModule.intro, rule Group-axioms, rule RModuleI)
show 1: Ring1 group-ring using Ring1-RG by fast
from 1 FG show Group FG using Ring1.axioms(1) by fast
from 1 FG show RModule-axioms group-ring (*) FG
using Ring1.mult-closed
by unfold-locales (auto simp add: algebra-simps)
qed

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
proof (rule FinGroupRepresentation.intro)
from FG show FGModule G (*) FG using FGModule-FG by fast
show FinGroupRepresentation-axioms G (*) FG
proof
from FG good-char obtain gs
where gs: set gs = G
 $\forall f \in FG. \exists bs. \text{length } bs = \text{length } gs$ 
 $\wedge f = (\sum (b,g) \leftarrow \text{zip } bs \text{ } gs. (b \ \delta\delta \ 0) * (1 \ \delta\delta \ g))$ 
using good-card-imp-finite FinGroupI FinGroup.group-ring-spanning-set
by fast
define xs where xs = map (( $\delta\delta$ ) (1::'f)) gs
with FG gs(1) have 1: set xs  $\subseteq$  FG using RG-aezdeltafun-closed by auto
moreover have aezfun-scalar-mult.fSpan (*) xs = FG
proof
from 1 FG show aezfun-scalar-mult.fSpan (*) xs  $\subseteq$  FG
using FGModule-FG FGModule.fSpan-closed by fast
show aezfun-scalar-mult.fSpan (*) xs  $\supseteq$  FG
proof
fix x assume  $x \in FG$ 
from this gs(2) obtain bs
where bs:  $\text{length } bs = \text{length } gs$ 
 $x = (\sum (b,g) \leftarrow \text{zip } bs \text{ } gs. (b \ \delta\delta \ 0) * (1 \ \delta\delta \ g))$ 
by fast
from bs(2) xs-def have  $x = (\sum (b,a) \leftarrow \text{zip } bs \text{ } xs. (b \ \delta\delta \ 0) * a)$ 
using sum-list-prod-map2[THEN sym] by fast
with bs(1) xs-def show  $x \in \text{aezfun-scalar-mult.fSpan} (*) \text{ } xs$ 
using aezfun-scalar-mult.fsmultD[of (*), THEN sym]
 $\text{sum-list-prod-cong}$ 
 $\text{of } \text{zip } bs \text{ } xs \ \lambda b \ a. (b \ \delta\delta \ 0) * a$ 
 $\lambda b \ a. \text{aezfun-scalar-mult.fsmult} (*) \ b \ a$ 
]
scalar-mult.lincomb-def[of aezfun-scalar-mult.fsmult (*) bs xs]
scalar-mult.SpanD-lincomb[of aezfun-scalar-mult.fsmult (*)]
by force
qed

```

qed
ultimately show $\exists xs. \text{set } xs \subseteq FG \wedge \text{aezfun-scalar-mult.fSpan } (*) \text{ } xs = FG$
by fast
qed (rule good-char)
qed

lemma (in *FinGroupRepresentation*) *FinGroupRepresentation-FG* :
FinGroupRepresentation $G (*) FG$
using good-char *ActingGroup.FinGroupRepresentation-FG* **by fast**

lemma (in *Group*) *FG-reducible* :
assumes of-nat (card G) $\neq (0::'f::\text{field})$
shows $\exists Us::('f,'g) \text{aezfun set list.}$
 $(\forall U \in \text{set } Us. \text{IrrFinGroupRepresentation } G (*) U) \wedge 0 \notin \text{set } Us$
 $\wedge \text{group-ring} = (\bigoplus U \leftarrow Us. U)$
using *assms FinGroupRepresentation-FG FinGroupRepresentation.reducible*
by fast

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* $G \text{ smult } V$
proof (rule *FinGroupRepresentation.IrrI*)
from *assms show FinGroupRepresentation* $G (\cdot) V$
using *IrrFinGroupRepresentation.axioms(1) isomorphic-sym*
FinGroupRepresentation.isomorphic-imp-GRep
by fast
from *assms show* $\bigwedge U. G\text{Subspace } U \implies U = 0 \vee U = V$
using *IrrFinGroupRepresentation.irr isomorphic-to-irr-right'* **by fast**
qed

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 $\exists U \in \text{set } Us. \text{FGModule.isomorphic } G \text{ smult } W \text{ smult } U$
proof –
from *V-decomp(1)* **have** *abGrp*: $\forall U \in \text{set } Us. \text{AbGroup } U$
using *IrrFinGroupRepresentation.AbGroup* **by fast**
from nonzero *V-decomp(3)* **have** *indep*: *add-independentS* Us
using *inner-dirsumD2* **by fast**
with *V-decomp(3)* **have** $\forall U \in \text{set } Us. U \subseteq V$
using *abGrp AbGroup-subset-inner-dirsum* **by fast**
with *subsp(1,3) V-decomp(1)*
have *GSubspaces*: $G\text{Subspace } W \ \forall U \in \text{set } Us. G\text{Subspace } U$
using *IrrFinGroupRepresentation.axioms(1)[of G smult]*

$FinGroupRepresentation.axioms(1)[of\ G\ smult]\ GSubspaceI$
by *auto*
have $\bigwedge i. i < length\ Us \implies (\bigoplus Us\downarrow i) \cdot W \neq 0$
 $\implies FGModuleIso\ G\ smult\ W\ smult\ ((\bigoplus Us\downarrow i) \downarrow W)\ (Us!i)$
proof–
fix i **assume** $i: i < length\ Us\ (\bigoplus Us\downarrow i) \cdot W \neq 0$
from $i(1)\ V-decomp(3)$ **have** $GRepEnd\ (\bigoplus Us\downarrow i)$
using $GSubspaces(2)\ indep\ GEnd-inner-dirsum-el-decomp-nth$ **by** *fast*
hence $FGModuleHom\ G\ smult\ W\ smult\ ((\bigoplus Us\downarrow i) \downarrow W)$
using $GSubspaces(1)\ FGModuleEnd.FGModuleHom-restrict0-GSubspace$
by *fast*
moreover **have** $((\bigoplus Us\downarrow i) \downarrow W) \cdot W \subseteq Us!i$
proof–
from $V-decomp(3)\ i(1)\ subsp(1,3)$ **have** $(\bigoplus Us\downarrow i) \cdot W \subseteq Us!i$
using $indep\ abGrp\ AbGroup-inner-dirsum-el-decomp-nth-onto-nth$ **by** *fast*
thus *?thesis* **by** *auto*
qed
moreover **from** $i(1)\ V-decomp(1)$
have $IrrFinGroupRepresentation\ G\ smult\ (Us!i)$
by *simp*
ultimately **show** $FGModuleIso\ G\ smult\ W\ smult\ ((\bigoplus Us\downarrow i) \downarrow W)\ (Us!i)$
using $i(2)$
 $IrrFinGroupRepresentation.Schur1[$
 $OF\ subsp(3),\ of\ smult\ Us!i\ (\bigoplus Us\downarrow i) \downarrow W$
 $]$
by *auto*
qed
moreover **from** $nonzero\ V-decomp(3)$
have $\forall i < length\ Us. (\bigoplus Us\downarrow i) \cdot W = 0 \implies W = 0$
using $inner-dirsum-Nil\ abGrp\ subsp(1)\ indep$
 $AbGroup-inner-dirsum-subset-proj-eq-0[of\ Us\ W]$
by *fastforce*
ultimately **have** $\exists i < length\ Us. FGModuleIso\ G\ smult\ W\ smult$
 $((\bigoplus Us\downarrow i) \downarrow W)\ (Us!i)$
using $subsp(2)$ **by** *auto*
thus *?thesis* **using** $set-conv-nth[of\ Us]$ **by** *auto*
qed

theorem (**in** $IrrFinGroupRepresentation$) $iso-FG-constituent$:
assumes $nonzero : V \neq 0$
and $FG-decomp: \forall U \in set\ Us. IrrFinGroupRepresentation\ G\ (*)\ U$
 $0 \notin set\ Us\ FG = (\bigoplus U \leftarrow Us. U)$
shows $\exists U \in set\ Us. isomorphic\ (*)\ U$
proof–
from $nonzero$ **obtain** v **where** $v: v \in V\ v \neq 0$ **using** $nonempty$ **by** *auto*
define T **where** $T = (\lambda x. x \cdot v) \downarrow FG$
have $FGModuleHom\ G\ (*)\ FG\ smult\ T$
proof (*rule* $FGModule.FGModuleHomI-fromaxioms$)
show $FGModule\ G\ (*)\ FG$

```

    using ActingGroup.FGModule-FG by fast
  from T-def v(1) show  $\bigwedge v v'. v \in FG \implies v' \in FG \implies T (v + v') = T v + T v'$ 
    using Ring1.add-closed[OF Ring1] smult-distrib-right by auto
  from T-def show  $\text{supp } T \subseteq FG$  using supp-restrict0 by fast
  from T-def v(1) show  $\bigwedge r m. r \in FG \implies m \in FG \implies T (r * m) = r \cdot T m$ 
    using ActingGroup.RG-mult-closed by auto
qed
then obtain W
  where W: FGModule.GSubspace G (*) FG W FG = W  $\oplus$  (ker T  $\cap$  FG)
        FGModule.isomorphic G (*) W smult (T ' FG)
  using FG-n0
        FinGroupRepresentation.GRepHom-decomp[
          OF FinGroupRepresentation-FG
        ]
  by fast
  from T-def v have T ' FG = V using eq-GSpan-single RSpan-single by auto
  with W(3) have W': FGModule.isomorphic G (*) W smult V by fast
  with W(1) nonzero have W  $\neq$  0
    using FGModule.GSubspace-is-FGModule[OF ActingGroup.FGModule-FG]
          FGModule.isomorphic-to-zero-left
  by fastforce
  moreover from W' have IrrFinGroupRepresentation G (*) W
    using IrrFinGroupRepresentation-axioms FGModuleIso.isomorphic-to-irr-right
  by fast
  ultimately have  $\exists U \in \text{set } Us. FGModule.isomorphic G (*) W (*) U$ 
    using FG-decomp W(1) good-char FG-n0
          FinGroupRepresentation.IrrGSubspace-iso-constituent[
            OF ActingGroup.FinGroupRepresentation-FG, of W
          ]
  by simp
  with W(1) W' show ?thesis
    using FGModule.GSubspace-is-FGModule[OF ActingGroup.FGModule-FG]
          FGModule.isomorphic-sym[of G (*) W smult] isomorphic-trans
  by fast
qed

```

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  $\implies$  ('f,'g) aezfun set list where

```

$remisodups [] = []$
 $| remisodups (U \# Us) = (if$
 $\quad (\exists W \in set Us. FGModule.isomorphic G (*) U (*) W)$
 $\quad then remisodups Us else U \# remisodups Us)$

lemma *set-remisodups* : $set (remisodups Us) \subseteq set Us$
by (*induct Us*) *auto*

lemma *isodistinct-remisodups* :
 $[[\forall U \in set Us. FGModule G (*) U; V \in set (remisodups Us);$
 $W \in set (remisodups Us); V \neq W]]$
 $\implies \neg (FGModule.isomorphic G (*) V (*) W)$

proof (*induct Us arbitrary: V W*)
case (*Cons U Us*)
show *?case*
proof (*cases* $\exists X \in set Us. FGModule.isomorphic G (*) U (*) X$)
case *True with Cons show ?thesis by simp*
next
case *False show ?thesis*
proof (*cases* $V=U W=U$ *rule: conjcases*)
case *BothTrue with Cons(5) show ?thesis by fast*
next
case *OneTrue with False Cons(4,5) show ?thesis*
using *set-remisodups by auto*
next
case *OtherTrue with False Cons(2,3) show ?thesis*
using *set-remisodups FGModule.isomorphic-sym[of G (*) V (*) W]*
by *fastforce*
next
case *BothFalse with Cons False show ?thesis by simp*
qed
qed
qed *simp*

definition *FG-constituents* $\equiv SOME Us.$
 $(\forall U \in set Us. IrrFinGroupRepresentation G (*) U)$
 $\wedge 0 \notin set Us \wedge group-ring = (\bigoplus U \leftarrow Us. U)$

lemma *FG-constituents-irr* :
 $of-nat (card G) \neq (0::'f::field)$
 $\implies \forall U \in set (FG-constituents::('f,'g) aezfun set list).$
 $IrrFinGroupRepresentation G (*) U$
using *someI-ex[OF FG-reducible] unfolding FG-constituents-def by fast*

lemma *FG-constituents-n0*:
 $of-nat (card G) \neq (0::'f::field)$
 $\implies 0 \notin set (FG-constituents::('f,'g) aezfun set list)$
using *someI-ex[OF FG-reducible] unfolding FG-constituents-def by fast*

lemma *FG-constituents-constituents* :
of-nat (*card G*) \neq (*0*::*f*::*field*)
 \implies (*group-ring*::(*f*,*g*) *aezfun set*) = (\bigoplus *U* \leftarrow *FG-constituents*. *U*)
using *someI-ex*[*OF FG-reducible*] **unfolding** *FG-constituents-def* **by** *fast*

definition *GIrrRep-repset* \equiv *0* \cup *set* (*remisodups FG-constituents*)

lemma *finite-GIrrRep-repset* : *finite GIrrRep-repset*
unfolding *GIrrRep-repset-def* **by** *simp*

lemma *all-irr-GIrrRep-repset* :
assumes *of-nat* (*card G*) \neq (*0*::*f*::*field*)
shows $\forall U \in$ (*GIrrRep-repset*::(*f*,*g*) *aezfun set set*).
IrrFinGroupRepresentation G (*) *U*

proof

fix *U* :: (*f*,*g*) *aezfun set* **assume** *U* \in *GIrrRep-repset*
with *assms* **show** *IrrFinGroupRepresentation G* (*) *U*
using *trivial-IrrFinGroupRepresentation-in-FG GIrrRep-repset-def*
set-remisodups FG-constituents-irr
by (*cases U = 0*) *auto*

qed

lemma *isodistinct-GIrrRep-repset* :
defines *GIRRS* \equiv *GIrrRep-repset* :: (*f*::*field*,*g*) *aezfun set set*
assumes *of-nat* (*card G*) \neq (*0*::*f*) *V* \in *GIRRS* *W* \in *GIRRS* *V* \neq *W*
shows \neg (*FGModule.isomorphic G* (*) *V* (*) *W*)

proof (*cases V = 0 W = 0* *rule: conjcases*)

case *BothTrue* **with** *assms*(5) **show** *?thesis* **by** *fast*

next

case *OneTrue* **with** *assms*(1,2,4,5) **show** *?thesis*
using *GIrrRep-repset-def set-remisodups FG-constituents-n0*
trivial-FGModule[*of* (*)] *FGModule.isomorphic-to-zero-left*[*of G* (*)]
by *fastforce*

next

case *OtherTrue*
moreover with *assms*(1,3) **have** *V* \in *set FG-constituents*
using *GIrrRep-repset-def set-remisodups* **by** *auto*
ultimately show *?thesis*
using *assms*(2) *FG-constituents-n0 FG-constituents-irr*
IrrFinGroupRepresentation.axioms(1)
FinGroupRepresentation.axioms(1)
FGModule.isomorphic-to-zero-right[*of G* (*) *V* (*)]
by *fastforce*

next

case *BothFalse*
with *assms*(1,3,4) **have** *V* \in *set* (*remisodups FG-constituents*)
 $W \in$ *set* (*remisodups FG-constituents*)
using *GIrrRep-repset-def* **by** *auto*
with *assms*(2,5) **show** *?thesis*

using *FG-constituents-irr IrrFinGroupRepresentation.axioms(1)*
FinGroupRepresentation.axioms(1) isodistinct-remisodups
by *fastforce*
qed
end

lemma (in *FGModule*) *iso-in-list-imp-iso-in-remisodups* :
 $\exists U \in \text{set } Us. \text{isomorphic } (*) U$
 $\implies \exists U \in \text{set } (\text{ActingGroup.remisodups } Us). \text{isomorphic } (*) U$
proof (*induct Us*)
case (*Cons U Us*)
from *Cons(2)* **obtain** *W* **where** *W*: $W \in \text{set } (U \# Us) \text{isomorphic } (*) W$
by *fast*
show *?case*
proof (
cases $W = U \exists X \in \text{set } Us. \text{FGModule.isomorphic } G (*) U (*) X$
rule: conjcases
)
case *BothTrue* **with** *W(2) Cons(1)* **show** *?thesis*
using *isomorphic-trans[of (*) W] by force*
next
case *OneTrue* **with** *W(2)* **show** *?thesis by simp*
next
case *OtherTrue* **with** *Cons(1) W* **show** *?thesis by auto*
next
case *BothFalse* **with** *Cons(1) W* **show** *?thesis by auto*
qed
qed *simp*

lemma (in *IrrFinGroupRepresentation*) *iso-to-GIrrRep-rep* :
 $\exists U \in \text{ActingGroup.GIrrRep-repset}. \text{isomorphic } (*) U$
using *zero-isomorphic-to-FG-zero ActingGroup.GIrrRep-repset-def*
good-char ActingGroup.FG-constituents-irr
ActingGroup.FG-constituents-n0 ActingGroup.FG-constituents-constituents
iso-FG-constituent iso-in-list-imp-iso-in-remisodups
ActingGroup.GIrrRep-repset-def
by (*cases V = 0*) *auto*

theorem (in *Group*) *iso-class-reps* :
defines $\text{GIRRS} \equiv \text{GIrrRep-repset} :: ('f::\text{field}, 'g) \text{aezfun set set}$
assumes *of-nat (card G) \neq (0::'f)*
shows *finite GIRRS*
 $\forall U \in \text{GIRRS}. \text{IrrFinGroupRepresentation } G (*) U$
 $\bigwedge U W. \llbracket U \in \text{GIRRS}; W \in \text{GIRRS}; U \neq W \rrbracket$
 $\implies \neg (\text{FGModule.isomorphic } G (*) U (*) W)$
 $\bigwedge \text{fgsmult } V. \text{IrrFinGroupRepresentation } G \text{fgsmult } V$
 $\implies \exists U \in \text{GIRRS}. \text{FGModule.isomorphic } G \text{fgsmult } V (*) U$
using *assms finite-GIrrRep-repset all-irr-GIrrRep-repset*

isodistinct-GIrrRep-repset IrrFinGroupRepresentation.iso-to-GIrrRep-rep
by auto

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* :: '*g*::*group-add set*
and *G* :: '*g set*
and *smult* :: ('*f*::*field*, '*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** \times 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 *BaseRep.indspace* *G FH V*
assumes *rrsmult* : *rrsmult* = *Ring1.rightreg-scalar-mult* *FH*
and *good-ordSupgrp*: *of-nat* (*card* *H*) \neq (0::'*f*)
and *Subgroup* : *Supgroup.Subgroup* *G*

sublocale *InducedFinGroupRepresentation* < *InducedFHModule*
using *FH indV rrsmult Subgroup* **by** *unfold-locales fast*

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*
using *good-ordSupgrp good-card-imp-finite* **by** *fast*

lemma *FinGroupSupgroup* : *FinGroup* *H*
using *finiteSupgroup Supgroup.FinGroupI* **by** *fast*

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 *flincomb-Horbit-induced-veclist-reduce* :

fixes *vs* $::$ 'v list

and *hs* $::$ 'g list

defines *hfvs* $:$ *hfvs* \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 hfvs*

and *rcoset-reps* : *Supgroup.is-rcoset-replist G hs*

shows ((concat *css*) \cdot \boxtimes (concat *hfvs*)) (1 $\delta\delta$ (*hs!**i*)) = (*css!**i*) \cdot \sharp *vs*

proof –

have *mostly-zero*:

$\bigwedge k j. k < \text{length } hs \implies j < \text{length } hs$
 $\implies ((\text{css!}j) \cdot \boxtimes (\text{hfvs!}j)) (1 \delta\delta \text{hs!}k)$
 $= (\text{if } j = k \text{ then } (\text{css!}k) \cdot \sharp \cdot \text{vs else } 0)$

proof –

fix *k j* **assume** *k*: *k* < length *hs* **and** *j*: *j* < length *hs*

hence *hsk-H*: *hs!**k* \in *H* **and** *hsj-H*: *hs!**j* \in *H*

using *Supgroup.is-rcoset-replistD-set*[*OF rcoset-reps*] **by** *auto*

define *LHS* **where** *LHS* = ((*css!**j*) \cdot \boxtimes (*hfvs!**j*)) (1 $\delta\delta$ *hs!**k*)

with *hfvs*

have *LHS* = ($\sum (r,m) \leftarrow \text{zip } (\text{css!}j) (\text{hfvs!}j). (r \boxtimes m) (1 \delta\delta \text{hs!}k)$)

using *length-negHorbit-list scalar-mult.lincomb-def*[*of induced-smult.fsmult*]
sum-list-prod-fun-apply

by *simp*

moreover **have** $\forall (r,m) \in \text{set } (\text{zip } (\text{css!}j) (\text{hfvs!}j)).$

$(\text{induced-smult.fsmult } r \ m) (1 \delta\delta \text{hs!}k) = r \ \sharp \cdot m (1 \delta\delta \text{hs!}k)$

proof (*rule prod-ballI*)

fix *r m* **assume** (*r,m*) \in set (*zip (css!j) (hfvs!j)*)

with *k j vs hfvs*

show (*induced-smult.fsmult r m*) (1 $\delta\delta$ *hs!**k*) = *r* \sharp \cdot *m* (1 $\delta\delta$ *hs!**k*)

using *Supgroup.is-rcoset-replistD-set*[*OF rcoset-reps*] *set-zip-rightD*

set-concat length-negHorbit-list[*of hs induced-vector vs*]

nth-mem[*of j hfvs*] *hsk-H induced-fsmult-conv-fsmult-1ddh*[*of m hs!k r*]

induced-vector-indV negHorbit-list-indV[*of hs induced-vector vs*]

by *force*

qed

ultimately **have**

$LHS = (\sum (r,v) \leftarrow zip (css!j) vs.$
 $r \# \cdot (induced\text{-}vector\ v) (1\ \delta\delta\ hs!k * (1\ \delta\delta - hs!j)))$
using $FH\ j\ hfvs\ induced\text{-}smult.negGorbit\text{-}list\text{-}def[of\ hs\ induced\text{-}vector\ vs]$
 $sum\text{-}list\text{-}prod\text{-}cong[of\ -\ \lambda r\ m. (induced\text{-}smult.fsmult\ r\ m) (1\ \delta\delta\ hs!k)]$
 $sum\text{-}list\text{-}prod\text{-}map2[of$
 $\lambda r\ m. r \# \cdot m (1\ \delta\delta\ hs!k) - Hmult\ (-\ hs!j)\ map\ induced\text{-}vector\ vs$
 $]$
 $sum\text{-}list\text{-}prod\text{-}map2[of\ \lambda r\ v. r \# \cdot (Hmult\ (-hs!j)\ v) (1\ \delta\delta\ hs!k)]$
 $induced\text{-}smult.GmultD\ hsk\text{-}H$
 $Supgroup.RG\text{-}aezdeltafun\text{-}closed[of\ hs!k\ 1::'f]$
 $rrsmultD1[of\ 1\ \delta\delta\ (hs!k)]$
by $force$
moreover have $(1::'f)\ \delta\delta\ hs!k * (1\ \delta\delta - hs!j) = 1\ \delta\delta (hs!k - hs!j)$
using $times\text{-}aezdeltafun\text{-}aezdeltafun[of\ 1::'f\ hs!k\ 1\ -(hs!j)]$
by $(simp\ add: algebra\text{-}simps)$
ultimately have $LHS = (\sum (r,v) \leftarrow zip (css!j) vs.$
 $r \# \cdot (induced\text{-}vector\ v) (1\ \delta\delta (hs!k - hs!j)))$
using $sum\text{-}list\text{-}prod\text{-}map2\ by\ simp$
moreover from $FH\ vs$
have $\forall (r,v) \in set (zip (css!j) vs). r \# \cdot (induced\text{-}vector\ v) (1\ \delta\delta (hs!k - hs!j))$
 $= r \# \cdot (FG\text{-}proj (1\ \delta\delta (hs!k - hs!j)) \cdot v)$
using $set\text{-}zip\text{-}rightD\ induced\text{-}vector\text{-}def\ hsk\text{-}H\ hsj\text{-}H\ Supgroup.diff\text{-}closed$
 $Supgroup.RG\text{-}aezdeltafun\text{-}closed[of\ -\ 1::'f]$
by $fastforce$
ultimately have $calc: LHS = (\sum (r,v) \leftarrow zip (css!j) vs.$
 $r \# \cdot (FG\text{-}proj (1\ \delta\delta (hs!k - hs!j)) \cdot v))$
using $sum\text{-}list\text{-}prod\text{-}cong\ by\ force$
show $LHS = (if\ j = k\ then\ (css!k) \cdot \# \cdot vs\ else\ 0)$
proof $(cases\ j = k)$
case $True$
with $calc\ have\ LHS = (\sum (r,v) \leftarrow zip (css!k) vs. r \# \cdot 1 \# \cdot v)$
using $Group.zero\text{-}closed[OF\ GroupG]$
 $aezfun\text{-}setspan\text{-}proj\text{-}aezdeltafun[of\ G\ 1::'f]$
 $BaseRep.fsmult\text{-}def$
by $simp$
moreover from $vs\ have\ \forall (r,v) \in set (zip (css!k) vs). r \# \cdot 1 \# \cdot v = r \# \cdot v$
using $set\text{-}zip\text{-}rightD\ BaseRep.fsmult\text{-}assoc\ by\ fastforce$
ultimately show $?thesis$
using $True\ sum\text{-}list\text{-}prod\text{-}cong[of\ -\ \lambda r\ v. r \# \cdot 1 \# \cdot v]$
 $scalar\text{-}mult.lincomb\text{-}def[of\ BaseRep.fsmult]$
by $simp$
next
case $False$
with $k\ j\ calc\ have\ LHS = (\sum (r,v) \leftarrow zip (css!j) vs. r \# \cdot (0 \cdot v))$
using $Supgroup.is\text{-}rcoset\text{-}replist\text{-}imp\text{-}nrelated\text{-}nth[OF\ Subgroup\ rcoset\text{-}reps]$
 $aezfun\text{-}setspan\text{-}proj\text{-}aezdeltafun[of\ G\ 1::'f]$
by $simp$
moreover from $vs\ have\ \forall (r,v) \in set (zip (css!j) vs). r \# \cdot (0 \cdot v) = 0$
using $set\text{-}zip\text{-}rightD\ BaseRep.zero\text{-}smult\ by\ fastforce$

ultimately have $LHS = (\sum (r,v) \leftarrow \text{zip } (css!j) \text{ vs. } (0::'v))$
using $\text{sum-list-prod-cong}[\text{of } - \lambda r v. r \# \cdot (0 \cdot v)]$ **by** simp
hence $LHS = (\sum rv \leftarrow \text{zip } (css!j) \text{ vs. case-prod } (\lambda r v. (0::'v)) rv)$ **by** fastforce
with False **show** $?thesis$ **by** simp
qed
qed

define terms LHS
where $\text{terms} = \text{map } (\lambda a. \text{case-prod } (\lambda cs \text{ hfvs. } (cs \cdot \boxtimes \text{ hfvs}) (1 \ \delta\delta \text{ hs!i})) a)$
 $(\text{zip } css \text{ hfvs})$
and $LHS = ((\text{concat } css) \cdot \boxtimes (\text{concat } hfvs)) (1 \ \delta\delta \text{ (hs!i)})$
hence $LHS = \text{sum-list terms}$
using scalars
 $\text{VectorSpace.lincomb-concat}[\text{OF } f\text{VectorSpace-indspace, of } css \text{ hfvs}]$
 $\text{sum-list-prod-fun-apply}$
by simp
hence $LHS = (\sum j \in \{0..<\text{length terms}\}. \text{terms!j})$
using $\text{sum-list-sum-nth}[\text{of terms}]$ **by** simp
moreover from terms-def
have $\forall j \in \{0..<\text{length terms}\}. \text{terms!j} = ((css!j) \cdot \boxtimes (\text{hfvs!j})) (1 \ \delta\delta \text{ hs!i})$
by simp
ultimately show $LHS = (css!i) \cdot \# \cdot vs$
using $\text{terms-def sum.cong scalars list-all2-lengthD}[\text{of } - css \text{ hfvs}] \text{ hfvs}$
 $\text{length-negHorbit-list}[\text{of } hs \text{ induced-vector vs}] \text{ i mostly-zero}$
 $\text{sum-single-nonzero}$
 $\text{of } \{0..<\text{length hs}\} \lambda i j. ((css!j) \cdot \boxtimes (\text{hfvs!j})) (1 \ \delta\delta \text{ (hs!i)})$
 $\lambda i. (css!i) \cdot \# \cdot vs$
]
by simp

qed

lemma $\text{indspace-fspanning-set}$:
fixes vs $:: 'v \text{ list}$
and hs $:: 'g \text{ list}$
defines $hfvs$ $: hfvs \equiv \text{negHorbit-list } hs \text{ induced-vector } vs$
assumes $\text{base-spset} : \text{set } vs \subseteq V \ V = \text{BaseRep.fSpan } vs$
and $\text{rcoset-reps} : \text{Supgroup.is-rcoset-replist } G \ hs$
shows $\text{indV} = \text{induced-smult.fSpan } (\text{concat } hfvs)$
proof $(\text{rule } \text{VectorSpace.SpanI}[\text{OF } f\text{VectorSpace-indspace}])$
from $\text{base-spset}(1) \ hfvs$ **show** $\text{set } (\text{concat } hfvs) \subseteq \text{indV}$
using $\text{Supgroup.is-rcoset-replistD-set}[\text{OF } \text{rcoset-reps}]$
 $\text{induced-vector-indV } \text{negHorbit-list-indV}$
by fast
show $\text{indV} \subseteq \text{R-scalar-mult.RSpan } UNIV \ (\text{aezfun-scalar-mult.fsmult } (\boxtimes))$
 $(\text{concat } hfvs)$
proof

fix f **assume** $f: f \in \text{indV}$

```

hence  $\forall h \in \text{set } hs. f (1 \ \delta\delta \ h) \in V$ 
  using indV BaseRep.indspaceD-range by fast
with base-spset(2)
  have coeffs-exist:  $\forall h \in \text{set } hs. \exists ahs. \text{length } ahs = \text{length } vs$ 
     $\wedge f (1 \ \delta\delta \ h) = ahs \cdot\# \cdot vs$ 
  using BaseRep.in-fSpan-obtain-same-length-coeffs
  by fast
define f-coeffs
  where f-coeffs  $h = (\text{SOME } ahs. \text{length } ahs = \text{length } vs \wedge f (1 \ \delta\delta \ h) = ahs$ 
 $\cdot\# \cdot vs)$  for  $h$ 
define ahss where ahss = map f-coeffs hs
hence len-ahss:  $\text{length } ahss = \text{length } hs$  by simp
with hfvs have len-zip-ahss-hfvs:  $\text{length } (\text{zip } ahss \ hfvs) = \text{length } hs$ 
  using length-negHorbit-list[of hs induced-vector vs] by simp
have len-ahss-el:  $\forall ahs \in \text{set } ahss. \text{length } ahs = \text{length } vs$ 
proof
  fix ahs assume  $ahs \in \text{set } ahss$ 
  from this ahss-def obtain  $h: h \in \text{set } hs \ ahs = f\text{-coeffs } h$ 
  using set-map by auto
  from  $h(1)$  have  $\exists ahs. \text{length } ahs = \text{length } vs \wedge f (1 \ \delta\delta \ h) = ahs \cdot\# \cdot vs$ 
  using coeffs-exist by fast
  with  $h(2)$  show  $\text{length } ahs = \text{length } vs$ 
  unfolding f-coeffs-def
  using someI-ex[of  $\lambda ahs. \text{length } ahs = \text{length } vs \wedge f (1 \ \delta\delta \ h) = ahs \cdot\# \cdot vs$ ]
  by fast
qed
have  $\forall (ahs, hfvs) \in \text{set } (\text{zip } ahss \ hfvs). \text{length } ahs = \text{length } hfvs$ 
proof
  fix  $x$  assume  $x: x \in \text{set } (\text{zip } ahss \ hfvs)$ 
  show case x of  $(ahs, hfvs) \Rightarrow \text{length } ahs = \text{length } hfvs$ 
  proof
    fix  $ahs \ hfvs$  assume  $x = (ahs, hfvs)$ 
    with  $x \ hfvs$  have  $\text{length } ahs = \text{length } vs \ \text{length } hfvs = \text{length } vs$ 
    using set-zip-leftD[of ahs hfvs] len-ahss-el set-zip-rightD[of ahs hfvs]
    length-negHorbit-list-sublist[of - hs induced-vector]
    by auto
    thus  $\text{length } ahs = \text{length } hfvs$  by simp
  qed
qed
with hfvs have list-all2-len-ahss-hfvs:
  list-all2  $(\lambda rs \ ms. \text{length } rs = \text{length } ms) \ ahss \ hfvs$ 
  using len-ahss length-negHorbit-list[of hs induced-vector vs]
  list-all2I[of ahss hfvs]
  by auto

define f' where f' =  $(\text{concat } ahss) \cdot\text{m}\cdot\text{m} \cdot (\text{concat } hfvs)$ 
have  $f = f'$ 
  using Supgroup.is-rcoset-replistD-set[OF rcoset-reps]
  Supgroup.group-eq-subgrp-rcoset-un[OF Subgroup rcoset-reps]

```

```

      f
proof (rule indspace-el-eq'[of hs])
  from f'-def hfvs base-spset(1) show  $f' \in \text{ind}V$ 
    using Supgroup.is-rcoset-replistD-set[OF rcoset-reps]
      induced-vector-indV negHorbit-list-indV[of hs induced-vector vs]
      FGModule.flincomb-closed[OF FHModule-indspace]
    by fast
  have  $\bigwedge i. i < \text{length } hs \implies f(1 \ \delta\delta \ (hs!i)) = f'(1 \ \delta\delta \ (hs!i))$ 
proof–
  fix i assume  $i < \text{length } hs$ 
  with f-coeffs-def have  $f(1 \ \delta\delta \ (hs!i)) = (f\text{-coeffs } (hs!i)) \cdot\# \ vs$ 
    using coeffs-exist
      someI-ex[of  $\lambda ahs. \text{length } ahs = \text{length } vs \wedge f(1 \ \delta\delta \ hs!i) = ahs \cdot\# \ vs$ ]
    by auto
  moreover from i hfvs f'-def base-spset(1) rcoset-reps ahss-def
    have  $f'(1 \ \delta\delta \ (hs!i)) = (f\text{-coeffs } (hs!i)) \cdot\# \ vs$ 
    using list-all2-len-ahss-hfvs flincomb-Horbit-induced-veclist-reduce
    by simp
  ultimately show  $f(1 \ \delta\delta \ (hs!i)) = f'(1 \ \delta\delta \ (hs!i))$  by simp
qed
thus  $\forall i < \text{length } hs. f(1 \ \delta\delta \ (hs!i)) = f'(1 \ \delta\delta \ (hs!i))$  by fast
qed
with f'-def hfvs base-spset(1) show  $f \in \text{induced-smult.fSpan } (\text{concat } hfvs)$ 
  using Supgroup.is-rcoset-replistD-set[OF rcoset-reps]
    induced-vector-indV negHorbit-list-indV[of hs induced-vector vs]
    VectorSpace.SpanI-lincomb-arb-len-coeffs[OF fVectorSpace-indspace]
  by fast

qed
qed

lemma indspace-basis :
  fixes vs      :: 'v list
  and   hs      :: 'g list
  defines hfvs   : hfvs  $\equiv \text{negHorbit-list } hs \text{ induced-vector } vs$ 
  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–
  from assms
    have  $1: \text{set } (\text{concat } hfvs) \subseteq \text{ind}V$ 
    and    $\text{ind}V = \text{induced-smult.fSpan } (\text{concat } hfvs)$ 
  using Supgroup.is-rcoset-replistD-set[OF rcoset-reps]
    induced-vector-indV negHorbit-list-indV[of hs induced-vector vs]
    indspace-fspanning-set[of vs hs]
  by auto
moreover have induced-smult.f-lin-independent (concat hfvs)
proof (
  rule VectorSpace.lin-independentI-concat-all-scalars[OF fVectorSpace-indspace],

```

```

rule 1
)
fix rss
assume rss: list-all2 (λxs ys. length xs = length ys) rss hfvs
      (concat rss) ·⊔⊔ (concat hfvs) = 0
from rss(1) have len-rss-hfvsss: length rss = length hfvs
  using list-all2-lengthD by fast
with hfvs have len-rss-hs: length rss = length hs
  using length-negHorbit-list by fastforce
show ∀rs∈set rss. set rs ⊆ 0
proof
  fix rs assume rs ∈ set rss
  from this obtain i where i: i < length rss rs = rss!i
    using in-set-conv-nth[of rs] by fast
  with hfvs rss(1) have length rs = length vs
    using list-all2-nthD len-rss-hfvsss in-set-conv-nth[of - hfvs]
      length-negHorbit-list-sublist
    by fastforce
  moreover from hfvs rss i base-spset(1) rcoset-reps have rs ·⊔· vs = 0
    using len-rss-hs flincomb-Horbit-induced-veclist-reduce by force
  ultimately show set rs ⊆ 0
    using base-spset flin-independentD-all-scalars by force
qed
qed
ultimately show ?thesis by fast
qed

```

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-
from hfvs base-spset
  have hfvs = (map induced-vector vs) # (negHorbit-list hs induced-vector vs)
  using induced-vector-indV
    FGModule.negGorbit-list-Cons0[OF FHModule-indspace]
  by fastforce
with cs extrazeros base-spset rcoset-reps v
  show induced-vector v = (cs@extrazeros) ·⊔⊔ (concat hfvs)
  using scalar-mult.lincomb-append[of cs - induced-smult.fsmult]
    Supgroup.is-rcoset-replistD-set induced-vector-indV
    negHorbit-list-indV[of hs induced-vector vs]

```



```

    VectorSpace.lincomb-replicate0-left[OF fVectorSpace-indspace]
    FGModuleHom.VectorSpaceHom[OF hom-induced-vector]
    VectorSpaceHom.distrib-lincomb
  by fastforce
qed

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 –

from *BaseRep.findim* **obtain** *vs* **where** *vs: set vs* $\subseteq V$ $V = \text{BaseRep.fSpan } vs$

by *fast*

obtain *hs* **where** *hs: Supgroup.is-rcoset-replist G hs*

using *ex-rcoset-replist* **by** *fast*

define *hfuss* **where** *hfuss = negHorbit-list hs induced-vector vs*

with *vs hs*

have *set (concat hfuss) \subseteq indV indV = induced-smult.fSpan (concat hfuss)*

using *Supgroup.is-rcoset-replistD-set[OF hs] induced-vector-indV*

negHorbit-list-indV[of hs induced-vector vs] inductive-fspanning-set

by *auto*

thus *?thesis* **by** *fast*

qed

theorem *FinGroupRepresentation-indspace* :

FinGroupRepresentation H rrsmult indV

using *FHModule-indspace*

proof (*rule FinGroupRepresentation.intro*)

from *good-ordSupgrp* **show** *FinGroupRepresentation-axioms H (\boxtimes) indV*

using *indspace-findim* **by** *unfold-locales fast*

qed

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 => 'v::ab-group-add => 'v (infixl · 70)
and V      :: 'v set
and rrsmult :: ('f, 'g) aezfun => (('f, 'g) aezfun => 'v)
              => (('f, 'g) aezfun => 'v) (infixl ⌘ 70)
and smult' :: ('f, 'g) aezfun => 'w::ab-group-add => 'w (infixr ★ 70)
and W      :: 'w set
begin

abbreviation fsmult' :: 'f => 'w => 'w          (infixr ‡★ 70)
  where fsmult' ≡ HRep.fsmult
abbreviation flincomb' :: 'f list => 'w list => 'w (infixr ·‡★ 70)
  where flincomb' ≡ HRep.flincomb
abbreviation Hmult'    :: 'g => 'w => 'w          (infixr **★ 70)
  where Hmult' ≡ HRep.Gmult

definition Tsmult1 ::
  'f => (((f, 'g) aezfun => 'v) => 'w) => (((f, 'g) aezfun => 'v) => 'w) (infixr ★⌘ 70)
  where Tsmult1 ≡ λa T. λf. a ‡★ (T f)

definition Tsmult2 :: 'f => ('v => 'w) => ('v => 'w) (infixr ★· 70)
  where Tsmult2 ≡ λa T. λv. a ‡★ (T v)

lemma FHModuleW : FGModule H (★) W ..

lemma FGModuleW : FGModule G smult' W
  using FHModuleW Subgroup HRep.restriction-to-subgroup-is-module
  by fast

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 ≡ (SOME vs. is-Vfbasis vs)

lemma Vfbasis : is-Vfbasis Vfbasis
  using Vfbasis-def FinDim VectorSpace.basis-ex[OF GRep.FinDim VectorSpace] someI-ex
  by simp

lemma Vfbasis-V : set Vfbasis ⊆ V
  using Vfbasis by fast

definition nonzero-H-rcoset-reps :: 'g list
  where nonzero-H-rcoset-reps ≡ (SOME hs. Group.is-rcoset-replist H G (0 # hs))

definition H-rcoset-reps :: 'g list where H-rcoset-reps ≡ 0 # nonzero-H-rcoset-reps

lemma H-rcoset-reps : Group.is-rcoset-replist H G H-rcoset-reps
  using H-rcoset-reps-def nonzero-H-rcoset-reps-def GRep.ex-rcoset-replist-hd0 someI-ex
  by simp

```

lemma *H-rcoset-reps-H* : set *H-rcoset-reps* \subseteq *H*
using *H-rcoset-reps Group.is-rcoset-replistD-set*[*OF HRep.GroupG*] **by** *fast*

lemma *nonzero-H-rcoset-reps-H* : set *nonzero-H-rcoset-reps* \subseteq *H*
using *H-rcoset-reps-H H-rcoset-reps-def* **by** *simp*

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* \Longrightarrow set (concat (*negHorbit-homVfbasis T*)) \subseteq *W*
using *H-rcoset-reps-H Vfbasis-V*
HRep.negGorbit-list-V[of *H-rcoset-reps T Vfbasis*]
by *fast*

lemma *negHorbit-HomSet-indVfbasis-W* :
T \in *GRepHomSet smult' W* \Longrightarrow set (concat (*negHorbit-homVfbasis T*)) \subseteq *W*
using *FGModuleHomSetD-Im negHorbit-Hom-indVfbasis-W* **by** *fast*

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*)
using *Vfbasis H-rcoset-reps indVfbasis-def indspace-basis*[of *Vfbasis H-rcoset-reps*]
by *auto*

lemma *indVfbasis-indV* : *hfvs* \in set *indVfbasis* \Longrightarrow set *hfvs* \subseteq *indV*
using *indVfbasis* **by** *auto*

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* ($\lambda T. T \circ$ *GRep.induced-vector*) (*HRepHomSet smult' W*)

lemma φ -*im* : φ ' *HRepHomSet* (\star) *W* \subseteq *GRepHomSet* (\star) *W*
proof (*rule image-subsetI*)

fix *T* **assume** *T*: *T* \in *HRepHomSet* (\star) *W*
show φ *T* \in *GRepHomSet* (\star) *W*
proof (*rule FGModuleHomSetI*)

```

from  $T$  have  $FGModuleHom\ G\ rrsmult\ indV\ smult'$   $T$ 
  using  $FGModuleHomSetD-FGModuleHom\ GRep.Subgroup$ 
     $FGModuleHom.restriction-to-subgroup-is-hom$ 
  by  $fast$ 
thus  $BaseRep.GRepHom\ (\star)\ (\varphi\ T)$ 
  using  $T\ \varphi-def\ GRep.hom-induced-vector\ GRep.induced-vector-indV$ 
     $FGModuleHom.FGModHom-composite-left$ 
  by  $fastforce$ 

show  $\varphi\ T\ 'V\ \subseteq\ W$ 
using  $T\ \varphi-def\ GRep.induced-vector-indV\ FGModuleHomSetD-Im$  by  $fastforce$ 

qed

qed

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  $\psi-condition :: ('v \Rightarrow 'w) \Rightarrow (((f, 'g)\ aezfun \Rightarrow 'v) \Rightarrow 'w) \Rightarrow bool$ 
  where  $\psi-condition\ T\ S$ 
     $\equiv\ VectorSpaceHom\ induced-smult.fsmult\ indV\ fsmult'\ S$ 
     $\wedge\ map\ (map\ S)\ indVfbasis = negHorbit-homVfbasis\ T$ 

```

```

lemma  $inverse-im-exists'$  :

```

```

  assumes  $T \in GRepHomSet\ (\star)\ W$ 

```

```

  shows  $\exists! S. VectorSpaceHom\ induced-smult.fsmult\ indV\ fsmult'\ S$ 

```

```

     $\wedge\ map\ S\ (concat\ indVfbasis) = concat\ (negHorbit-homVfbasis\ T)$ 

```

```

proof (

```

```

   $rule\ VectorSpace.basis-im-defines-hom,$   $rule\ fVectorSpace-indspace,$ 

```

```

   $rule\ HRep.fVectorSpace,$   $rule\ indVfbasis$ 

```

```

)

```

```

from  $assms$  show  $set\ (concat\ (negHorbit-homVfbasis\ T)) \subseteq W$ 

```

```

  using  $negHorbit-HomSet-indVfbasis-W$  by  $fast$ 

```

```

show  $length\ (concat\ (negHorbit-homVfbasis\ T)) = length\ (concat\ indVfbasis)$ 

```

```

  using  $length-concat-negGorbit-list\ indVfbasis-def$ 

```

```

     $induced-smult.length-concat-negGorbit-list[of\ H-rcoset-reps\ induced-vector]$ 

```

```

  by  $simp$ 

```

```

qed

```

```

lemma  $inverse-im-exists$  :

```

```

  assumes  $T \in GRepHomSet\ (\star)\ W$ 

```

```

  shows  $\exists! S. \psi-condition\ T\ S$ 

```

```

proof –
  have  $\exists S. \psi\text{-condition } T S$ 
proof –
  from assms obtain  $S$ 
    where  $S: \text{VectorSpaceHom induced-smult.fsmult indV fsmult}' S$ 
       $\text{map } S (\text{concat indVfbasis}) = \text{concat } (\text{negHorbit-homVfbasis } T)$ 
    using inverse-im-exists'
    by fast
  from  $S(2)$  have  $\text{concat } (\text{map } (\text{map } S) \text{ indVfbasis})$ 
     $= \text{concat } (\text{negHorbit-homVfbasis } T)$ 
    using map-concat[of S] by simp
  moreover have list-all2  $(\lambda xs ys. \text{length } xs = \text{length } ys)$ 
     $(\text{map } (\text{map } S) \text{ indVfbasis}) (\text{negHorbit-homVfbasis } T)$ 
proof (rule iffD2[OF list-all2-iff], rule conjI)
  show  $\text{length } (\text{map } (\text{map } S) \text{ indVfbasis}) = \text{length } (\text{negHorbit-homVfbasis } T)$ 
    using indVfbasis-def induced-smult.length-negGorbit-list
      HRep.length-negGorbit-list[of H-rcoset-reps T]
    by auto
  show  $\forall (xs,ys) \in \text{set } (\text{zip } (\text{map } (\text{map } S) \text{ indVfbasis})$ 
     $(\text{negHorbit-homVfbasis } T)). \text{length } xs = \text{length } ys$ 
proof (rule prod-ballI)
  fix  $xs\ ys$ 
  assume  $xs\ ys: (xs,ys) \in \text{set } (\text{zip } (\text{map } (\text{map } S) \text{ indVfbasis})$ 
     $(\text{negHorbit-homVfbasis } T))$ 
  from this obtain  $zs$  where  $zs: zs \in \text{set indVfbasis } xs = \text{map } S\ zs$ 
    using set-zip-leftD by fastforce
  with  $xs\ ys$  show  $\text{length } xs = \text{length } ys$ 
    using indVfbasis-def set-zip-rightD[of xs ys]
      HRep.length-negGorbit-list-sublist[of ys H-rcoset-reps T Vfbasis]
      induced-smult.length-negGorbit-list-sublist
    by simp
  qed
qed
ultimately have  $\text{map } (\text{map } S) \text{ indVfbasis} = \text{negHorbit-homVfbasis } T$ 
  using concat-eq[of map (map S) indVfbasis] by fast
  with  $S(1)$  show ?thesis using  $\psi$ -condition-def by fast
qed
moreover have  $\bigwedge S U. \psi\text{-condition } T S \implies \psi\text{-condition } T U \implies S = U$ 
proof –
  fix  $S U$  assume  $\psi\text{-condition } T S \ \psi\text{-condition } T U$ 
  hence  $\text{VectorSpaceHom induced-smult.fsmult indV fsmult}' S$ 
     $\text{map } S (\text{concat indVfbasis}) = \text{concat } (\text{negHorbit-homVfbasis } T)$ 
     $\text{VectorSpaceHom induced-smult.fsmult indV fsmult}' U$ 
     $\text{map } U (\text{concat indVfbasis}) = \text{concat } (\text{negHorbit-homVfbasis } T)$ 
  using  $\psi$ -condition-def map-concat[of S] map-concat[of U] by auto
  with assms show  $S = U$  using inverse-im-exists' by fast
qed
ultimately show ?thesis by fast
qed

```

definition $\psi :: ('v \Rightarrow 'w) \Rightarrow (((f, 'g) \text{ aezfun} \Rightarrow 'v) \Rightarrow 'w)$
where $\psi \equiv \text{restrict0 } (\lambda T. \text{ THE } S. \psi\text{-condition } T S) (G\text{RepHomSet } (\star) W)$

lemma $\psi D : T \in G\text{RepHomSet } (\star) W \Longrightarrow \psi\text{-condition } T (\psi T)$
using $\psi\text{-def inverse-im-exists[of } T] \text{ theI'[of } \lambda S. \psi\text{-condition } T S]$ **by simp**

lemma $\psi D\text{-VectorSpaceHom} :$
 $T \in G\text{RepHomSet } (\star) W$
 $\Longrightarrow \text{VectorSpaceHom induced-smult.fsmult indV fsmult' } (\psi T)$
using $\psi D \psi\text{-condition-def}$ **by fast**

lemma $\psi D\text{-im} :$
 $T \in G\text{RepHomSet } (\star) W \Longrightarrow \text{map } (\text{map } (\psi T)) \text{ indVfbasis}$
 $= \text{ aezfun-scalar-mult.negGorbit-list } (\star) H\text{-rcoset-reps } T \text{ Vfbasis}$
using $\psi D \psi\text{-condition-def}$ **by fast**

lemma $\psi D\text{-im-single} :$
assumes $T \in G\text{RepHomSet } (\star) W h \in \text{set } H\text{-rcoset-reps } v \in \text{set } V\text{fbasis}$
shows $\psi T ((- h) * \boxtimes (\text{induced-vector } v)) = (-h) ** (T v)$

proof –

from $\text{assms}(2,3)$ **obtain** $i j$

where $i : i < \text{length } H\text{-rcoset-reps } h = H\text{-rcoset-reps}!i$

and $j : j < \text{length } V\text{fbasis } v = V\text{fbasis}!j$

using $\text{set-conv-nth[of } H\text{-rcoset-reps]} \text{ set-conv-nth[of } V\text{fbasis]}$ **by auto**

moreover

hence $\text{map } (\text{map } (\psi T)) \text{ indVfbasis } !i !j = \psi T ((-h) * \boxtimes (\text{induced-vector } v))$

using indVfbasis-def

$\text{aezfun-scalar-mult.length-negGorbit-list[}$
 $\text{of } \text{rrsmult } H\text{-rcoset-reps induced-vector}$
]

$\text{aezfun-scalar-mult.negGorbit-list-nth[}$
 $\text{of } i H\text{-rcoset-reps rrsmult induced-vector}$
]

by auto

ultimately show $?thesis$

using $\text{assms}(1) H\text{Rep.negGorbit-list-nth[of } i H\text{-rcoset-reps } T]$ $\psi D\text{-im}$ **by simp**

qed

lemma $\psi T\text{-}W :$

assumes $T \in G\text{RepHomSet } (\star) W$

shows $\psi T \text{ ` } \text{indV} \subseteq W$

proof ($\text{rule image-subsetI}$)

from assms **have** $T : \text{VectorSpaceHom induced-smult.fsmult indV fsmult' } (\psi T)$

using $\psi D\text{-VectorSpaceHom}$ **by fast**

fix f **assume** $f \in \text{indV}$

from this obtain cs

where $cs : \text{length } cs = \text{length } (\text{concat indVfbasis}) f = cs \cdot \boxtimes \boxtimes (\text{concat indVfbasis})$

using $\text{indVfbasis scalar-mult.in-Span-obtain-same-length-coeffs}$

by *fast*
 from *cs(1)* obtain *css*
 where *css*: $cs = \text{concat } css \text{ list-all2 } (\lambda xs \ ys. \text{length } xs = \text{length } ys) \text{ } css \text{ indVfbasis}$
 using *match-concat*
 by *fast*
 from *assms cs(2)* *css*
 have $\psi \ T \ f = \psi \ T \ (\sum (cs, hfvs) \leftarrow \text{zip } css \ \text{indVfbasis}. \ cs \ \cdot \ \boxtimes \ hfvs)$
 using *VectorSpace.lincomb-concat[OF fVectorSpace-indspace]* by *simp*
 also have $\dots = (\sum (cs, hfvs) \leftarrow \text{zip } css \ \text{indVfbasis}. \ \psi \ T \ (cs \ \cdot \ \boxtimes \ hfvs))$
 using *set-zip-rightD[of - - css indVfbasis]* *indVfbasis-indV*
 VectorSpace.lincomb-closed[OF GRep.fVectorSpace-indspace]
 VectorSpaceHom.im-sum-list-prod[OF T]
 by *force*
 finally have $\psi \ T \ f = (\sum (cs, \psi \ Thfvs) \leftarrow \text{zip } css \ (\text{map } (\text{map } (\psi \ T)) \ \text{indVfbasis}).$
 $cs \ \cdot \ \# \star \ \psi \ Thfvs)$
 using *set-zip-rightD[of - - css indVfbasis]* *indVfbasis-indV*
 VectorSpaceHom.distrib-lincomb[OF T]
 sum-list-prod-cong[of
 $\text{zip } css \ \text{indVfbasis} \ \lambda cs \ hfvs. \ \psi \ T \ (cs \ \cdot \ \boxtimes \ hfvs)$
 $\lambda cs \ hfvs. \ cs \ \cdot \ \# \star \ (\text{map } (\psi \ T) \ hfvs)$
]
 sum-list-prod-map2[of \lambda cs \ \psi \ Thfvs. \ cs \ \cdot \ \# \star \ \psi \ Thfvs \ css \ \text{map } (\psi \ T)]
 by *fastforce*
 moreover from *css(2)*
 have $\text{list-all2 } (\lambda xs \ ys. \text{length } xs = \text{length } ys) \text{ } css \ (\text{map } (\text{map } (\psi \ T)) \ \text{indVfbasis})$
 using *list-all2-iff[of - css indVfbasis]* *set-zip-map2*
 list-all2-iff[of - css map (map (\psi T)) indVfbasis]
 by *force*
 ultimately have $\psi \ T \ f = (\text{concat } css) \ \cdot \ \# \star \ (\text{concat } (\text{negHorbit-homVfbasis } T))$
 using *HRep.flincomb-concat map-concat[of \psi T] \psi D-im[OF assms]*
 by *simp*
 thus $\psi \ T \ f \in W$
 using *assms negHorbit-HomSet-indVfbasis-W HRep.flincomb-closed* by *simp*
 qed

lemma $\psi \ T \text{-Hmap-on-indVfbasis}$:
 assumes $T \in \text{GRepHomSet } (\star) \ W$
 shows $\bigwedge x \ f. \ x \in H \implies f \in \text{set } (\text{concat } \text{indVfbasis})$
 $\implies \psi \ T \ (x \ \cdot \ \boxtimes \ f) = x \ \cdot \ \star \ (\psi \ T \ f)$

proof –

fix $x \ f$ **assume** $x: x \in H$ **and** $f: f \in \text{set } (\text{concat } \text{indVfbasis})$
from f **obtain** i **where** $i: i < \text{length } \text{indVfbasis}$ $f \in \text{set } (\text{indVfbasis}!i)$
 using *set-concat set-conv-nth[of indVfbasis]* by *auto*
from $i(1)$ **have** $i': i < \text{length } H\text{-rcoset-reps}$
 using *indVfbasis-def*
 aezfun-scalar-mult.length-negGorbit-list[
 of rrsmult H-rcoset-reps induced-vector
]
 by *simp*

```

define hi where hi = H-rcoset-reps!i
with i' have hi-H: hi ∈ H using set-conv-nth H-rcoset-reps-H by fast
from hi-def i(2) have f ∈ set (map (Hmult (-hi) ∘ induced-vector) Vfbasis)
using indVfbasis-def i'
    aezfun-scalar-mult.negGorbit-list-nth[
      of i H-rcoset-reps rrsmult induced-vector
    ]
by simp
from this obtain v where v: v ∈ set Vfbasis f = (-hi) *⊠ (induced-vector v)
by auto
from v(1) have v-V: v ∈ V and Tv-W: T v ∈ W
using Vfbasis-V FGModuleHomSetD-Im[OF assms] by auto
from x have hi - x ∈ H using hi-H Supgroup.diff-closed by fast
from this obtain j
  where j: j < length H-rcoset-reps hi - x ∈ G + {H-rcoset-reps!j}
using set-conv-nth[of H-rcoset-reps] H-rcoset-reps
    Group.group-eq-subgrp-rcoset-un[OF HRep.GroupG Subgroup H-rcoset-reps]
by force
from j(1) have j': j < length indVfbasis
using indVfbasis-def
    aezfun-scalar-mult.length-negGorbit-list[
      of rrsmult H-rcoset-reps induced-vector
    ]
by simp
define hj where hj = H-rcoset-reps!j
with j(1) have hj-H: hj ∈ H using set-conv-nth H-rcoset-reps-H by fast
from hj-def j(2) obtain g where g: g ∈ G hi - x = g + hj
unfolding set-plus-def by fast
from g(2) have x-hi: x - hi = - hj + - g
using minus-diff-eq[of hi x] minus-add[of g] by simp
from g(1) have -g *· v ∈ V
using v-V ActingGroup.neg-closed BaseRep.Gmult-closed by fast
from this obtain cs
  where cs: length cs = length Vfbasis -g *· v = cs ·⊠ Vfbasis
using Vfbasis
    VectorSpace.in-Span-obtain-same-length-coeffs[OF GRep.fVectorSpace]
by fast

from v(2) x have ψ T (x *⊠ f) = ψ T ((x-hi) *⊠ (induced-vector v))
using hi-H Supgroup.neg-closed v-V induced-vector-indV
    FGModule.Gmult-assoc[OF GRep.FHModule-indspace]
by (simp add: algebra-simps)
also from g(1) have ... = ψ T ((-hj) *⊠ (induced-vector (-g *· v)))
using x-hi hj-H Subgroup Supgroup.neg-closed v-V induced-vector-indV
    FGModule.Gmult-assoc[OF GRep.FHModule-indspace]
    ActingGroup.neg-closed
    FGModuleHom.G-map[OF hom-induced-vector]
by auto
also from cs(2) hj-def j(1) have ... = ψ T (cs ·⊠⊠ (indVfbasis!j))

```



```

using hj-H Vfbasis-V FGModuleHom.distrib-flincomb[OF hom-induced-vector]
      indVfbasis-def Supgroup.neg-closed[of hj] induced-vector-indV
      FGModule.Gmult-flincomb-comm[
        OF GRep.FHModule-indspace,
        of -hj map induced-vector Vfbasis
      ]
      aezfun-scalar-mult.negGorbit-list-nth[
        of j H-rcoset-reps rrsmult induced-vector
      ]
by fastforce
also have ... = cs ·#* ((map (map (ψ T)) indVfbasis)!j)
      using ψD-VectorSpaceHom[OF assms] indVfbasis-indV j' set-conv-nth
          VectorSpaceHom.distrib-lincomb[of induced-smult.fsmult indV fsmult]
by simp
also from j(1) hj-def have ... = (- hj) ** cs ·#* (map T Vfbasis)
      using ψD-im[OF assms]
          aezfun-scalar-mult.negGorbit-list-nth[of j H-rcoset-reps smult' T] hj-H
          Group.neg-closed[OF HRep.GroupG]
          Vfbasis-V FGModuleHomSetD-Im[OF assms]
          HRep.Gmult-flincomb-comm[of - hj map T Vfbasis]
by fastforce
also from cs(2) g(1) have ... = (- hj) ** (-g) ** (T v)
      using v-V FGModuleHomSetD-FGModuleHom[OF assms] Vfbasis-V
          FGModuleHom.distrib-flincomb[of G smult V smult]
          ActingGroup.neg-closed
          FGModuleHom.G-map[of G smult V smult' T -g v]
by auto
also from g(1) v(1) have ... = (x - hi) ** (T v)
      using FGModuleHomSetD-FGModuleHom[OF assms] Vfbasis-V Supgroup.neg-closed
          hj-H Subgroup FGModuleHomSetD-Im[OF assms]
          HRep.Gmult-assoc[of -hj -g T v] x-hi
by auto
also from x(1) have ... = x ** (- hi) ** (T v)
      using hi-H Supgroup.neg-closed Tv-W HRep.Gmult-assoc
by (simp add: algebra-simps)
finally show ψ T (x *α f) = x ** (ψ T f)
      using assms(1) v hi-def i' set-conv-nth[of H-rcoset-reps] ψD-im-single by fast-
force
qed

lemma ψT-hom :
  assumes T ∈ GRepHomSet (★) W
  shows HRepHom (★) (ψ T)
  using indVfbasis ψD-VectorSpaceHom[OF assms] FHModuleW
proof (
  rule FGModule.VecHom-GMap-on-fbasis-is-FGModuleHom[
    OF GRep.FHModule-indspace
  ]
)

```

```

show  $\psi T \text{ ' } indV \subseteq W$  using indVfbasis  $\psi T\text{-}W$ [OF assms] by fast
show  $\bigwedge g v. g \in H \implies v \in set (concat\ indVfbasis)$ 
       $\implies \psi T (g * \alpha v) = g ** \psi T v$ 
using  $\psi T\text{-}Hmap\text{-}on\text{-}indVfbasis$ [OF assms] by fast
qed

```

```

lemma  $\psi\text{-}im : \psi \text{ ' } GRepHomSet (\star) W \subseteq HRepHomSet (\star) W$ 
using  $\psi T\text{-}W$   $\psi T\text{-}hom$  FGModuleHomSetI by fastforce

```

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  $T \in GRepHomSet\ smult' W$ 
  shows  $(\varphi \circ \psi) T = T$ 
proof
  fix v show  $(\varphi \circ \psi) T v = T v$ 
  proof (cases  $v \in V$ )
    case True
    from this obtain cs where  $cs: length\ cs = length\ Vfbasis\ v = cs \cdot \# \cdot Vfbasis$ 
    using Vfbasis
      VectorSpace.in-Span-obtain-same-length-coeffs[OF GRep.fVectorSpace]
    by fast
    define extrazeros
    where  $extrazeros = replicate ((length\ nonzero\text{-}H\text{-}rcoset\text{-}reps) * (length\ Vfbasis))$ 
    ( $0::'f$ )
    with cs have  $GRep.induced\text{-}vector\ v = (cs @ extrazeros) \cdot \alpha \alpha (concat\ indVfbasis)$ 
    using H-rcoset-reps induced-vector-decomp[OF Vfbasis]
    unfolding H-rcoset-reps-def indVfbasis-def
    by auto
    with assms
    have  $(\varphi \circ \psi) T v = (cs @ extrazeros) \cdot \# \star (map (\psi T) (concat\ indVfbasis))$ 
    using  $\psi\text{-}im$   $\varphi\text{-}def$  indVfbasis
      VectorSpaceHom.distrib-lincomb[OF \psi D-VectorSpaceHom]
    by auto
    also have  $\dots = (cs @ extrazeros) \cdot \# \star (map T Vfbasis$ 
       $@ concat (HRep.negGorbit\text{-}list\ nonzero\text{-}H\text{-}rcoset\text{-}reps T Vfbasis))$ 
    using map-concat[of \psi T]  $\psi D\text{-}im$ [OF assms] H-rcoset-reps-def
      FGModuleHomSetD-Im[OF assms] Vfbasis-V HRep.negGorbit-list-Cons0
    by fastforce
    also from  $cs(1)$ 
    have  $\dots = cs \cdot \# \star (map T Vfbasis) + extrazeros$ 
       $\cdot \# \star (concat (HRep.negGorbit\text{-}list\ nonzero\text{-}H\text{-}rcoset\text{-}reps T Vfbasis))$ 
    using scalar-mult.lincomb-append[of cs - fsmult']
  
```

```

    by simp
  also have ... = cs ·#* (map T Vfbasis)
    using nonzero-H-rcoset-reps-H Vfbasis FGModuleHomSetD-Im[OF assms]
      HRep.negGorbit-list-V
      VectorSpace.lincomb-replicate0-left[OF HRep.fVectorSpace]
    unfolding extrazeros-def
    by force
  also from cs(2) have ... = T v
    using FGModuleHomSetD-FGModuleHom[OF assms]
      FGModuleHom.VectorSpaceHom Vfbasis
      VectorSpaceHom.distrib-lincomb[of aezfun-scalar-mult.fsmult smult]
    by fastforce
  finally show ?thesis by fast
next
case False
with assms show ?thesis
  using  $\psi$ -im  $\varphi$ -def GRep.induced-vector-def  $\psi$ D-VectorSpaceHom
    VectorSpaceHom.im-zero
    FGModuleHomSetD-FGModuleHom[of T G smult V]
    FGModuleHom.supp suppI-contra
  by fastforce
qed
qed

```

lemma φ -inverse-im : $\varphi \text{ ' } HRepHomSet (\star) W \supseteq GRepHomSet (\star) W$
 using $\varphi\psi$ ψ -im by force

lemma $bij\text{-}\varphi$: $bij\text{-}betw \varphi (HRepHomSet (\star) W) (GRepHomSet (\star) W)$
 unfolding $bij\text{-}betw\text{-}def$
proof

have $\bigwedge S T. \llbracket S \in HRepHomSet (\star) W; T \in HRepHomSet (\star) W; \varphi S = \varphi T \rrbracket \implies S = T$

proof (rule VectorSpaceHom.same-image-on-spanset-imp-same-hom)
 fix S T

assume ST: $S \in HRepHomSet (\star) W$ $T \in HRepHomSet (\star) W$ $\varphi S = \varphi T$

from ST(1,2) have ST': $HRepHom smult' S$ $HRepHom smult' T$
 using FGModuleHomSetD-FGModuleHom[of - H rrsmult] by auto

from ST'

show VectorSpaceHom induced-smult.fsmult indV fsmult' S

VectorSpaceHom induced-smult.fsmult indV fsmult' T

using FGModuleHom.VectorSpaceHom[of H rrsmult indV smult']

by auto

show $indV = induced\text{-}smult.fSpan (concat indVfbasis)$

set (concat indVfbasis) $\subseteq indV$

using indVfbasis by auto

```

show  $\forall f \in \text{set } (\text{concat } \text{indVfbasis}). S f = T f$ 
proof
  fix  $f$  assume  $f \in \text{set } (\text{concat } \text{indVfbasis})$ 
  from this obtain  $hfvs$  where  $hfvs: hfvs \in \text{set } \text{indVfbasis } f \in \text{set } hfvs$ 
  using set-concat by fast
  from  $hfvs(1)$  obtain  $h$ 
  where  $h: h \in \text{set } H\text{-rcoset-reps}$ 
   $hfvs = \text{map } (H\text{mult } (-h) \circ \text{induced-vector}) \text{Vfbasis}$ 
  using indVfbasis-def
   $\text{induced-smult.negGorbit-list-def}[of H\text{-rcoset-reps induced-vector}]$ 
  by auto
  from  $hfvs(2)$   $h(2)$  obtain  $v$ 
  where  $v: v \in \text{set } \text{Vfbasis } f = (-h) * \times (\text{induced-vector } v)$ 
  by auto
  from  $v$   $h(1)$  ST(1) have  $S f = (-h) * * (\varphi S v)$ 
  using ST'(1) H-rcoset-reps-H Group.neg-closed[OF HRep.GroupG]
   $G\text{Rep.induced-vector-indV Vfbasis-V } \varphi\text{-def } F\text{GModuleHom.G-map}$ 
  by fastforce
  moreover from  $v$   $h(1)$  ST(2) have  $T f = (-h) * * (\varphi T v)$ 
  using ST'(2) H-rcoset-reps-H Group.neg-closed[OF HRep.GroupG] GRep.induced-vector-indV
   $\text{Vfbasis-V } \varphi\text{-def } F\text{GModuleHom.G-map}$ 
  by fastforce
  ultimately show  $S f = T f$  using ST(3) by simp

```

qed

qed

thus *inj-on* $\varphi (H\text{RepHomSet } (\star) W)$ **unfolding** *inj-on-def* **by** *fast*

show $\varphi ' H\text{RepHomSet } (\star) W = G\text{RepHomSet } (\star) W$

using *φ -im* *φ -inverse-im* **by** *fast*

qed

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- φ* :

VectorSpaceIso Tsmult1 (HRepHomSet (\star) W) Tsmult2 φ
(GRepHomSet (\star) W)

proof (*rule VectorSpaceIso.intro, rule VectorSpace.VectorSpaceHomI-fromaxioms*)

from *Tsmult1-def* **show** *VectorSpace Tsmult1 (HRepHomSet (\star) W)*

using *FHModule-indspace FHModuleW*

```

      FGModule.VectorSpace-FGModuleHomSet
    by simp

  from  $\varphi$ -def show  $\text{supp } \varphi \subseteq \text{HRepHomSet } (\star) W$ 
    using  $\text{suppD-contral}[of \varphi]$  by fastforce

  have  $\text{bij-betw } \varphi (\text{HRepHomSet } (\star) W) (\text{GRepHomSet } (\star) W)$ 
    using  $\text{bij-}\varphi$  by fast
  thus  $\text{VectorSpaceIso-axioms } (\text{HRepHomSet } (\star) W) \varphi (\text{GRepHomSet } (\star) W)$ 
    by  $\text{unfold-locales}$ 

next
  fix  $S T$  assume  $S \in \text{HRepHomSet } (\star) W T \in \text{HRepHomSet } (\star) W$ 
  thus  $\varphi (S + T) = \varphi S + \varphi T$ 
    using  $\varphi$ -def  $\text{Group.add-closed}$ 
      FGModule.Group-FGModuleHomSet[OF FHModule-indspace FHModuleW]
  by auto

next
  fix  $a T$  assume  $T: T \in \text{HRepHomSet } (\star) W$ 
  moreover with  $\text{Tsmult1-def}$  have  $aT: a \star T \in \text{HRepHomSet } (\star) W$ 
    using FGModule.VectorSpace-FGModuleHomSet[
      OF FHModule-indspace FHModuleW
    ]
      VectorSpace.smult-closed
  by simp
  ultimately show  $\varphi (a \star T) = a \star (\varphi T)$ 
    using  $\varphi$ -def  $\text{Tsmult1-def}$   $\text{Tsmult2-def}$  by auto

qed

theorem FrobeniusReciprocity :
  VectorSpace.isomorphic Tsmult1 (HRepHomSet smult' W) Tsmult2
    (GRepHomSet smult' W)
  using VectorSpaceIso- $\varphi$  by fast

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.