# Representations of Finite Groups

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

April 18, 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 *zip* . . . 5
1.3.2 *concat* . . . 6
1.3.3 *strip-while* . . . 7
1.3.4 *sum-list* . . . 7
1.3.5 *listset* . . . 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 *Group* and finite variant *FinGroup* . . . 36
2.1.2 Abelian variant locale *AbGroup* . . . 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: *lincomb* . . . 70

3.2.2 Spanning: *RSpan* 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: *basis-for* . . . 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* : $(\bigwedge a\ b.\ (a,b) \in AxB \implies P\ a\ b) \implies \forall (a,b) \in AxB.\ P\ a\ b$
**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*)←*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*)←*zip xs ys*] = *map* (*case-prod f*) (*zip xs ys*)
**by** (*induct xs ys rule*: *list-induct2'*) *auto*

**lemma** *set-zip-map2* :
(*a*,*z*) ∈ *set* (*zip xs* (*map f ys*)) ⟹ ∃ *b*. (*a*,*b*) ∈ *set* (*zip xs ys*) ∧ *z* = *f b*
**by** (*induct xs ys rule*: *list-induct2'*) *auto*

### 1.3.2 *concat*

**lemma** *concat-eq* :
*list-all2* (λ*xs ys. length xs* = *length ys*) *xss yss* ⟹ *concat xss* = *concat yss*
⟹ *xss* = *yss*
**by** (*induct xss yss rule*: *list-all2-induct*) *auto*

**lemma** *match-concat* :
**fixes** *bss* :: *'b list list*
**defines** *eq-len*: *eq-len* ≡ λ*xs ys. length xs* = *length ys*
**shows** ∀ *as*::*'a list. length as* = *length* (*concat bss*)
⟶ (∃ *css*::*'a list list. as* = *concat css* ∧ *list-all2 eq-len css bss*)
**proof** (*induct bss*)
**from** *eq-len*
**show** ∀ *as. length as* = *length* (*concat* []) ⟶ (∃ *css. as* = *concat css* ∧ *list-all2 eq-len css* [])
**by** *simp*
**next**
**fix** *fs* :: *'b list* **and** *fss* :: *'b list list*
**assume** *prevcase*: ∀ *as. length as* = *length* (*concat fss*) ⟶ (∃ *css. as* = *concat css* ∧ *list-all2 eq-len css fss*)
**have** ⋀*as. length as* = *length* (*concat* (*fs* # *fss*)) ⟹ (∃ *css. as* = *concat css* ∧ *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* ∧ *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* ∧ *list-all2 eq-len hss* (*fs*#*fss*)
**using** *someI-ex*[*of* λ*css. ys* = *concat css* ∧ *list-all2 eq-len css fss*] **by** *auto*
**qed**
**thus** ∀ *as. length as* = *length* (*concat* (*fs* # *fss*))

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

### 1.3.3 *strip-while*

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

### 1.3.4 *sum-list*

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

**lemma** *sum-list-prod-cong* :
$\forall$ (*x*,*y*) $\in$ *set xys*. *f x y* = *g x y*
$\Longrightarrow$ ($\sum$ (*x*,*y*)$\leftarrow$*xys*. *f x y*) = ($\sum$ (*x*,*y*)$\leftarrow$*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*)$\leftarrow$*zip as* (*map f bs*). *g a y*) = ($\sum$ (*a*,*b*)$\leftarrow$*zip as bs*. *g a* (*f b*))
**by** (*induct as bs rule*: *list-induct2'*) *auto*

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

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

**lemma** (**in** *comm-monoid-add*) *sum-list-plus* :
*length xs* = *length ys*
$\Longrightarrow$ *sum-list xs* + *sum-list ys* = *sum-list* [*a*+*b*. (*a*,*b*)$\leftarrow$*zip xs 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*::*semiring-0*
**shows** *r* $*$ ($\sum$ (*x*,*y*)$\leftarrow$*xys*. *f x y*) = ($\sum$ (*x*,*y*)$\leftarrow$*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*::*semiring-0*
**shows** ($\sum$ (*x*,*y*)$\leftarrow$*xys*. *f x y*) $*$ *r* = ($\sum$ (*x*,*y*)$\leftarrow$*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** $n < length\ xs \Longrightarrow sum\text{-}list\ (xs[n := y]) = sum\text{-}list\ xs - xs!n + y$
**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* : $x \in X \Longrightarrow xs \in listset\ Xs \Longrightarrow x\#xs \in listset\ (X\#Xs)$
**unfolding** *listset-def set-Cons-def* **by** *simp*

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

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

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

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

lemma sum-list-listset : as ∈ listset As ⟹ sum-list as ∈ (∑ A←As. A)
proof−
  have length as = length As ⟹ as ∈ listset As ⟹ sum-list as ∈ (∑ A←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 ∈ listset As ⟹ sum-list as ∈ (∑ A←As. A) using listset-length by fast
qed

lemma listsetI-nth :
  length xs = length Xs ⟹ ∀ n<length xs. xs!n ∈ Xs!n ⟹ xs ∈ 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 ∈ listset (X#Xs)
    using listset-ConsI[of x X xs Xs] by fastforce
qed

lemma listsetD-nth : xs ∈ listset Xs ⟹ ∀ n<length xs. xs!n ∈ Xs!n
proof−
  have length xs = length Xs ⟹ xs ∈ listset Xs ⟹ ∀ n<length xs. xs!n ∈ 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 ∈ X xs ∈ listset Xs
      using listset-ConsD[of x] by auto
    with Cons(2) have 1: (x#xs)!0 ∈ (X#Xs)!0 ∀ n<length xs. xs!n ∈ Xs!n
      by auto
    have ⋀n. n < length (x#xs) ⟹ (x#xs)!n ∈ (X#Xs)!n
    proof−
      fix n assume n < length (x#xs)
      with 1 show (x#xs)!n ∈ (X#Xs)!n by (cases n) auto
    qed
    thus ∀ n < length (x#xs). (x#xs)!n ∈ (X#Xs)!n by fast
  qed
  thus xs ∈ listset Xs ⟹ ∀ n<length xs. xs!n ∈ Xs!n using listset-length by fast
qed
```

**lemma** *set-listset-el-subset* :
*xs* ∈ *listset Xs* ⟹ ∀ *X*∈*set Xs*. *X* ⊆ *A* ⟹ *set xs* ⊆ *A*
**proof**−
**have** ⟦ *length xs* = *length Xs*; *xs* ∈ *listset Xs*; ∀ *X*∈*set Xs*. *X* ⊆ *A* ⟧
⟹ *set xs* ⊆ *A*
**proof** (*induct xs Xs rule*: *list-induct2*)
**case** *Cons* **thus** *?case* **using** *listset-ConsD* **by** *force*
**qed** *simp*
**thus** *xs* ∈ *listset Xs* ⟹ ∀ *X*∈*set Xs*. *X* ⊆ *A* ⟹ *set xs* ⊆ *A*
**using** *listset-length* **by** *fast*
**qed**

## 1.4 Functions

### 1.4.1 Miscellaneous facts

**lemma** *sum-fun-apply* : *finite A* ⟹ (∑ *a*∈*A*. *f a*) *x* = (∑ *a*∈*A*. *f a x*)
**by** (*induct set*: *finite*) *auto*

**lemma** *sum-single-nonzero* :
*finite A* ⟹ (∀ *x*∈*A*. ∀ *y*∈*A*. *f x y* = (*if y* = *x then g x else 0*))
⟹ (∀ *x*∈*A*. *sum* (*f x*) *A* = *g x*)
**proof** (*induct A rule*: *finite-induct*)
**case** (*insert a A*)
**show** ∀ *x*∈*insert a A*. *sum* (*f x*) (*insert a A*) = *g x*
**proof**
**fix** *x* **assume** *x*: *x* ∈ *insert a A*
**show** *sum* (*f x*) (*insert a A*) = *g x*
**proof** (*cases x* = *a*)
**case** *True*
**moreover with** *insert*(*2*,*4*) **have** ∀ *y*∈*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*′) ∘ *S* = (*T* ∘ *S*) + (*T*′ ∘ *S*)
**by** *auto*

### 1.4.2 Support of a function

**definition** *supp* :: (′*a* ⇒ ′*b*::*zero*) ⇒ ′*a set* **where** *supp f* = {*x*. *f x* ≠ *0*}

**lemma** *suppI*: *f x* ≠ *0* ⟹ *x* ∈ *supp f*
**using** *supp-def* **by** *fast*

**lemma** *suppI-contra*: *x* ∉ *supp f* ⟹ *f x* = *0*
**using** *suppI* **by** *fast*

**lemma** *suppD*: $x \in supp\ f \Longrightarrow f\ x \neq 0$
**using** *supp-def* **by** *fast*

**lemma** *suppD-contra*: $f\ x = 0 \Longrightarrow x \notin supp\ f$
**using** *suppD* **by** *fast*

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

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

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

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

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

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

**lemma** *supp-restrict0* : $supp\ (f{\downarrow}A) \subseteq A$
**proof**−
**have** $\bigwedge a.\ a \notin A \Longrightarrow a \notin supp\ (f{\downarrow}A)$ **using** *suppD-contra*[*of* $f{\downarrow}A$] **by** *simp*
**thus** *?thesis* **by** *fast*
**qed**

**lemma** *bij-betw-restrict0* : $bij\text{-}betw\ f\ A\ B \Longrightarrow bij\text{-}betw\ (f \downarrow A)\ A\ B$
**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* ::
$('a$::*group-add*$ \Rightarrow 'b$::{*comm-monoid-add*,*times*}$) \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b)$

**where** *convolution f g*
$= (\lambda x.\ \sum y | x - y \in supp\ f \wedge y \in supp\ g.\ (f\ (x - y)) * g\ y)$
— More often than not, this definition will be used in the case that $'b$ is of class *mult-zero*, in which case the conditions $x - y \in supp\ f$ and $y \in supp\ g$ 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** $f\ g :: 'a$::*group-add* $\Rightarrow$ $'b$::{*comm-monoid-add*,*mult-zero*}
**shows** $f = 0 \vee g = 0 \Longrightarrow$ *convolution f g* $= 0$
**unfolding** *convolution-def*
**by** *auto*

**lemma** *convolution-symm* :
**fixes** $f\ g :: 'a$::*group-add* $\Rightarrow$ $'b$::{*comm-monoid-add*,*times*}
**shows** *convolution f g*
$= (\lambda x.\ \sum y | y \in supp\ f \wedge -y + x \in supp\ g.\ (f\ y) * g\ (-y + x))$
**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** $S1 = \{y.\ x - y \in supp\ f \wedge y \in supp\ g\}$
**and** $S2 = \{y.\ y \in supp\ f \wedge -y + x \in supp\ g\}$
**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-iS2*: $S1 = i\ `\ S2$
**proof** (*rule seteqI*)
**fix** $y$ **assume** *y-S1*: $y \in S1$
**define** $z$ **where** $z = x - y$
**hence** *y-eq*: $-z + x = y$ **by** (*auto simp add*: *algebra-simps*)
**hence** $-z + x \in supp\ g$ **using** *y-S1 S1-def* **by** *fast*
**moreover have** $z \in supp\ f$ **using** *z-def y-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-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-S2*: $z \in S2$ **and** *y-eq*: $y = -z + x$
**using** *i-def* **by** *fast*
**from** *y-eq* **have** $x - y = z$ **by** (*auto simp add*: *algebra-simps*)
**hence** $x - y \in supp\ f \wedge y \in supp\ g$ **using** *y-eq z-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*
$= (\sum y | y \in supp\ f \wedge -y + x \in supp\ g.\ (f\ y) * g\ (-y + x))$
**unfolding** *S1-def c1-def S2-def c2-def convolution-def* **by** *fast*
**qed**

**lemma** *supp-convolution-subset-sum-supp* :
**fixes** *f g* :: *'a*::*group-add* ⇒ *'b*::{*comm-monoid-add*,*times*}
**shows** *supp* (*convolution f g*) ⊆ *supp f* + *supp g*
**proof**−
**define** *SS* **where** *SS x* = {*y*. *x*−*y* ∈ *supp f* ∧ *y* ∈ *supp g*} **for** *x*
**have** *convolution f g* = (λ*x*. *sum* (λ*y*. (*f* (*x* − *y*)) ∗ *g y*) (*SS x*))
**unfolding** *SS-def convolution-def* **by** *fast*
**moreover have** ⋀*x*. *x* ∉ *supp f* + *supp g* ⟹ *SS x* = {}
**proof**−
**have** ⋀*x*. *SS x* ≠ {} ⟹ *x* ∈ *supp f* + *supp g*
**proof**−
**fix** *x*::*'a* **assume** *SS x* ≠ {}
**from** *this* **obtain** *y* **where** *x* − *y* ∈ *supp f* **and** *y-G*: *y* ∈ *supp g*
**using** *SS-def* **by** *fast*
**from** *this* **obtain** *z* **where** *z-F*: *z* ∈ *supp f* **and** *z-eq*: *x* − *y* = *z* **by** *fast*
**from** *z-eq* **have** *x* = *z* + *y* **using** *diff-eq-eq* **by** *fast*
**with** *z-F y-G* **show** *x* ∈ *supp f* + *supp g* **by** *fast*
**qed**
**thus** ⋀*x*. *x* ∉ *supp f* + *supp g* ⟹ *SS x* = {} **by** *fast*
**qed**
**ultimately have** ⋀*x*. *x* ∉ *supp f* + *supp g*
⟹ *convolution f g x* = *sum* (λ*y*. (*f* (*x* − *y*)) ∗ *g y*) {}
**by** *simp*
**hence** ⋀*x*. *x* ∉ *supp f* + *supp g* ⟹ *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*⇒*'b*::*zero*. *finite* (*supp f*)}

**lemma** *aezfun-setD*: *f* ∈ *aezfun-set* ⟹ *finite* (*supp f*)
**unfolding** *aezfun-set-def* **by** *fast*

**lemma** *aezfun-setI*: *finite* (*supp f*) ⟹ *f* ∈ *aezfun-set*
**unfolding** *aezfun-set-def* **by** *fast*

**lemma** *zerofun-is-aezfun* : *0* ∈ *aezfun-set*
**unfolding** *supp-def aezfun-set-def* **by** *auto*

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

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

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

**lemma** *restrict-and-extend0-aezfun-is-aezfun* :
**assumes** $f \in$ *aezfun-set*
**shows** $f{\downarrow}A \in$ *aezfun-set*
**proof** (*rule aezfun-setI*)
**have** $\bigwedge a.\ a \notin$ *supp* $f \cap A \Longrightarrow a \notin$ *supp* $(f{\downarrow}A)$
**proof**−
**fix** *a* **assume** $a \notin$ *supp* $f \cap A$
**thus** $a \notin$ *supp* $(f{\downarrow}A)$ **using** *suppI-contra*[*of a*] *suppD-contra*[*of* $f{\downarrow}A$ *a*]
**by** (*cases* $a \in A$) *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* :: $'b$::*zero* $\Rightarrow 'a \Rightarrow ('a \Rightarrow 'b)$ (**infix** $\delta$ *70*)
**where** $b\ \delta\ a = (\lambda x.$ *if* $x = a$ *then* $b$ *else 0*$)$

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

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

**lemma** *deltafun0* : *0 δ a = 0*
**unfolding** *deltafun-def* **by** *auto*

**lemma** *deltafun-plus* :
**fixes** *b c* :: *'b*::*monoid-add*
**shows** *(b+c) δ a = (b δ a) + (c δ a)*
**unfolding** *deltafun-def*
**by** *auto*

**lemma** *supp-delta0fun* : *supp (0 δ a) = {}*
**unfolding** *supp-def deltafun-def* **by** *simp*

**lemma** *supp-deltafun* : *b ≠ 0 ⟹ supp (b δ a) = {a}*
**unfolding** *supp-def deltafun-def* **by** *simp*

**lemma** *deltafun-is-aezfun* : *b δ a ∈ aezfun-set*
**proof** (*cases b = 0*)
**case** *True*
**hence** *supp (b δ 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'* :
*finite A ⟹ ∃ as. distinct as ∧ set as = A*
*∧ ( ∀ f*::*'a ⇒ 'b*::*semiring-1. supp f ⊆ A*
*⟶ (∃ bs. length bs = length as ∧ f = (∑ (b,a)←zip bs as. b δ 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*
*⋀f*::*'a ⇒ 'b. supp f ⊆ A*
*⟹ ∃ bs. length bs = length as ∧ f = (∑ (b,a)←zip bs as. b δ a)*
**by** *fast*
**from** *as*(*1*,*2*) *insert*(*2*) **have** *distinct (a#as) set (a#as) = insert a A* **by** *auto*
**moreover**
**have** *⋀f*::*'a ⇒ 'b*::*semiring-1. supp f ⊆ insert a A*
*⟹ (∃ bs. length bs = length (a#as)*
*∧ f = (∑ (b,a)←zip bs (a#as). b δ a))*
**proof**−
**fix** *f* :: *'a ⇒ 'b* **assume** *supp-f* : *supp f ⊆ insert a A*
**define** *g* **where** *g x = (if x = a then 0 else f x)* **for** *x*
**have** *supp g ⊆ A*
**proof**
**fix** *x* **assume** *x*: *x ∈ supp g*

**with** *x supp-f g-def* **have** $x \in$ *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*: *length bs* = *length as* $g = (\sum (b,a) \leftarrow zip\ bs\ as.\ b\ \delta\ a)$
**by** *fast*
**from** *bs*(*1*) **have** *length* ((*f a*) # *bs*) = *length* (*a*#*as*) **by** *auto*
**moreover from** *g-def bs*(*2*) **have** $f = (\sum (b,a) \leftarrow zip\ ((f\ a)\ \#\ bs)\ (a\#as).\ b\ \delta\ a)$
**using** *deltafun-apply-eq*[*of f a a*] *deltafun-apply-neq*[*of - a f a*] **by** (*cases*) *auto*
**ultimately**
**show** $\exists$ *bs*. *length bs* = *length* (*a*#*as*) $\wedge$ $f = (\sum (b,a) \leftarrow zip\ bs\ (a\#as).\ b\ \delta\ a)$
**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*::*group-add* $\Rightarrow$ *'b*::{*comm-monoid-add*,*mult-zero*}
**assumes** *g* $\in$ *aezfun-set*
**shows** *convolution f g* = $(\lambda x.\ \sum y \in supp\ g.\ (f\ (x - y)) * g\ y\ )$
**proof**
**fix** *x*::*'a*
**define** *SS* **where** *SS* = $\{y.\ x - y \in supp\ f \wedge y \in supp\ g\}$
**have** *finite* (*supp g*) **using** *assms* **unfolding** *aezfun-set-def* **by** *fast*
**moreover have** *SS* $\subseteq$ *supp g* **unfolding** *SS-def* **by** *fast*
**moreover have** $\bigwedge y.\ y \in supp\ g - SS \Longrightarrow (f\ (x - y)) * g\ y = 0$ **using** *SS-def*
**unfolding** *supp-def* **by** *auto*
**ultimately show** *convolution f g x* = $(\sum y \in supp\ g.\ (f\ (x - y)) * g\ y\ )$
**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*::*group-add* $\Rightarrow$ *'b*::{*comm-monoid-add*,*mult-zero*}
**assumes** *f* $\in$ *aezfun-set*
**shows** *convolution f g* = $(\lambda x.\ \sum y \in supp\ f.\ (f\ y) * g\ (-y + x))$
**proof**
**fix** *x*::*'a*
**define** *SS* **where** *SS* = $\{y.\ y \in supp\ f \wedge -y + x \in supp\ g\}$
**have** *finite* (*supp f*) **using** *assms* **unfolding** *aezfun-set-def* **by** *fast*
**moreover have** *SS* $\subseteq$ *supp f* **using** *SS-def* **by** *fast*
**moreover have** $\bigwedge y.\ y \in supp\ f - SS \Longrightarrow (f\ y) * g\ (-y + x) = 0$
**using** *SS-def* **unfolding** *supp-def* **by** *auto*
**ultimately**

```
    have (∑ y∈SS. (f y) * g (−y + x)) = (∑ y∈supp f. (f y) * g (−y + x) )
    unfolding convolution-def
    using     SS-def sum.mono-neutral-left[of supp f SS λy. (f y) * g (−y + x)]
    by        fast
  thus convolution f g x = (∑ y∈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::{comm-monoid-add,mult-zero}
  and   a :: 'a::group-add
  and   f :: 'a ⇒ 'b
  shows convolution (b δ a) f = (λx. 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⇒'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::{comm-monoid-add,mult-zero}
  and   f :: 'a::group-add ⇒ 'b and a::'a
  shows convolution f (b δ a) = (λx. 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⇒'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* ≠ *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* ⇒ *'b*::{*comm-monoid-add*,*times*}
**shows** *f* ∈ *aezfun-set* ⟹ *g* ∈ *aezfun-set* ⟹ *convolution f g* ∈ *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* ⇒ *'b*::*semiring-0*
**assumes** *f-aez*: *f* ∈ *aezfun-set* **and** *h-aez*: *h* ∈ *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*∈*supp f*. ($\sum$ *z*∈*supp h*. *f y* $*$ *g* (−*y* + *x* − *z*) $*$ *h z*) )
**proof**−
**have** *convolution fg h x* = ($\sum$ *z*∈*supp h*. *fg* (*x* − *z*) $*$ *h z* )
**using** *h-aez convolution-eq-sum-over-supp-right*[*of h fg*] **by** *simp*
**moreover have** ⋀*z*. *fg* (*x* − *z*) $*$ *h z*
= ($\sum$ *y*∈*supp f*. *f y* $*$ *g* (−*y* + *x* − *z*) $*$ *h z*)

```
    proof−
      fix z::'a
      have fg (x − z) = (∑ y∈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 = (∑ y∈supp f. f y * g (−y + (x − z)) * h z )
        using sum-distrib-right by simp
      thus fg (x − z) * h z = (∑ y∈supp f. f y * g (−y + x − z) * h z )
        by (simp add: algebra-simps)
    qed
    ultimately
      have  convolution fg h x
                = (∑ z∈supp h. (∑ y∈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
                  = (∑ y∈supp f. (∑ z∈supp h. f y * g (−y + x − z) * h z) )
  proof−
    have convolution f gh x = (∑ y∈supp f. f y * gh (−y + x) )
      using f-aez convolution-symm-eq-sum-over-supp-left[of f gh] by simp
    moreover have ⋀y. f y * gh (−y + x)
                    = (∑ z∈supp h. f y * g (−y + x − z) * h z)
    proof−
      fix y::'a
      have triple-cong: ⋀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) = (∑ z∈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) = (∑ z∈supp h. f y * (g (−y + x − z) * h z))
        using sum-distrib-left by simp
      also have ... = (∑ z∈supp h. f y * g (−y + x − z) * h z)
        using triple-cong sum.cong by simp
      finally
        show f y * gh (−y + x) = (∑ z∈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::group-add ⇒ 'b::semiring-0
  assumes g ∈ aezfun-set h ∈ aezfun-set
  shows   convolution f (g + h) = convolution f g + convolution f h
proof
  define gh GH where gh = g + h and GH = supp g ∪ supp h
```

**have** *fin-GH*: *finite GH* **using** *GH-def assms* **unfolding** *aezfun-set-def* **by** *fast*
**have** *gh-aezfun*: *gh* ∈ *aezfun-set* **using** *gh-def assms sum-of-aezfun-is-aezfun* **by** *fast*
**fix** *x*::*'a*
**have** *zero-ext-g* : ⋀*y*. *y* ∈ *GH* − *supp g* ⟹ (*f* (*x* − *y*)) ∗ *g y* = *0*
**and** *zero-ext-h* : ⋀*y*. *y* ∈ *GH* − *supp h* ⟹ (*f* (*x* − *y*)) ∗ *h y* = *0*
**and** *zero-ext-gh*: ⋀*y*. *y* ∈ *GH* − *supp gh* ⟹ (*f* (*x* − *y*)) ∗ *gh y* = *0*
**unfolding** *supp-def* **by** *auto*
**have** *convolution f gh x* = (∑ *y*∈*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** ... = (∑ *y*∈*GH*. (*f* (*x* − *y*)) ∗ *gh y*)
**using** *fin-GH supp-sum-subset-union-supp zero-ext-gh*
*sum.mono-neutral-left*[*of GH supp gh* (λ*y*. (*f* (*x* − *y*)) ∗ *gh y*)]
**by** *fast*
**also from** *gh-def*
**have** ... = (∑ *y*∈*GH*. (*f* (*x* − *y*)) ∗ *g y*) + (∑ *y*∈*GH*. (*f* (*x* − *y*)) ∗ *h y*)
**using** *sum.distrib* **by** (*simp add*: *algebra-simps*)
**finally show** *convolution f gh x* = (*convolution f g* + *convolution f h*) *x*
**using** *assms GH-def fin-GH zero-ext-g zero-ext-h*
*sum.mono-neutral-right*[*of GH supp g* (λ*y*. (*f* (*x* − *y*)) ∗ *g y*)]
*sum.mono-neutral-right*[*of GH supp h* (λ*y*. (*f* (*x* − *y*)) ∗ *h y*)]
*convolution-eq-sum-over-supp-right*[*of g f*]
*convolution-eq-sum-over-supp-right*[*of h f*]
**by** *fastforce*
**qed**

**lemma** *convolution-distrib-right* :
**fixes** *f g h* :: *'a*::*group-add* ⇒ *'b*::*semiring-0*
**assumes** *f* ∈ *aezfun-set g* ∈ *aezfun-set*
**shows** *convolution* (*f* + *g*) *h* = *convolution f h* + *convolution g h*
**proof**
**define** *fg FG* **where** *fg* = *f* + *g* **and** *FG* = *supp f* ∪ *supp g*
**have** *fin-FG*: *finite FG* **using** *FG-def assms* **unfolding** *aezfun-set-def* **by** *fast*
**have** *fg-aezfun*: *fg* ∈ *aezfun-set* **using** *fg-def assms sum-of-aezfun-is-aezfun* **by** *fast*
**fix** *x*::*'a*
**have** *zero-ext-f* : ⋀*y*. *y* ∈ *FG* − *supp f* ⟹ (*f y*) ∗ *h* (−*y* + *x*) = *0*
**and** *zero-ext-g* : ⋀*y*. *y* ∈ *FG* − *supp g* ⟹ (*g y*) ∗ *h* (−*y* + *x*) = *0*
**and** *zero-ext-fg*: ⋀*y*. *y* ∈ *FG* − *supp fg* ⟹ (*fg y*) ∗ *h* (−*y* + *x*) = *0*
**unfolding** *supp-def* **by** *auto*
**from** *assms* **have** *convolution fg h x* = (∑ *y*∈*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** ... = (∑ *y*∈*FG*. (*fg y*) ∗ *h* (−*y* + *x*))
**using** *fin-FG supp-sum-subset-union-supp zero-ext-fg*
*sum.mono-neutral-left*[*of FG supp fg* (λ*y*. (*fg y*) ∗ *h* (−*y* + *x*))]
**by** *fast*
**also from** *fg-def*
**have** ... = (∑ *y*∈*FG*. (*f y*) ∗ *h* (−*y* + *x*)) + (∑ *y*∈*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* (λ*y*. (*f y*) ∗ *h* (−*y* + *x*))]
*sum.mono-neutral-right*[*of FG supp g* (λ*y*. (*g y*) ∗ *h* (−*y* + *x*))]
*convolution-symm-eq-sum-over-supp-left*[*of f h*]
*convolution-symm-eq-sum-over-supp-left*[*of g h*]
**by** *fastforce*
**qed**

### 1.5.4 Type definition, instantiations, and instances

**typedef** (**overloaded**) ('*a*::*zero*,'*b*) *aezfun* = *aezfun-set* :: ('*b*⇒'*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* ⟹ *a* = *b* **by** *transfer fast*

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

**lemma** *zero-aezfun-transfer* : *Abs-aezfun* ((*0*::'*b*::*zero*) *δ* (*0*::'*a*::*zero*)) = *0*
**proof**−
**define** *zb za* **where** *zb* = (*0*::'*b*) **and** *za* = (*0*::'*a*)
**hence** *zb δ za* = *0* **using** *deltafun0*[*of za*] **by** *fast*
**moreover have** *aezfun 0* = *0* **using** *zero-aezfun.rep-eq* **by** *fast*
**ultimately have** *zb δ 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* ⇒ ('*a*, '*b*) *aezfun* ⇒ ('*a*, '*b*) *aezfun*
**is** λ*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* = (∑ *a*←*as*. *aezfun a x*)
**by** (*induct as*) *auto*

**lemma** *sum-list-map-aezfun-apply* [*simp*] :
*aezfun* (∑ *a*←*as*. *f a*) *x* = (∑ *a*←*as*. *aezfun* (*f a*) *x*)
**by** (*induct as*) *auto*

**lemma** *sum-list-map-aezfun* [*simp*] :
*aezfun* (∑ *a*←*as*. *f a*) = (∑ *a*←*as*. *aezfun* (*f a*))
**using** *sum-list-map-aezfun-apply*[*of f*] *sum-list-fun-apply*[*of aezfun* ∘ *f*] **by** *auto*

**lemma** *sum-list-prod-map-aezfun-apply* :
*aezfun* (∑ (*x*,*y*)←*xys*. *f x y*) *a* = (∑ (*x*,*y*)←*xys*. *aezfun* (*f x y*) *a*)
**by** (*induct xys*) *auto*

**lemma** *sum-list-prod-map-aezfun* :
*aezfun* (∑ (*x*,*y*)←*xys*. *f x y*) = (∑ (*x*,*y*)←*xys*. *aezfun* (*f x y*))
**using** *sum-list-prod-map-aezfun-apply*[*of f*]
*sum-list-prod-fun-apply*[*of* λ*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 aezfun (a + b) x = aezfun (b + a) x
      using add.commute[of aezfun a x] by simp
  qed
  thus a + b = b + a by transfer fast
  show 0 + a = a by simp
qed

lemma sum-aezfun-apply [simp] :
  finite A ⟹ aezfun (∑ A) x = (∑ a∈A. aezfun a x)
  by (induct set: finite) auto

instantiation aezfun :: (group-add, type) minus
begin
  lift-definition minus-aezfun :: ('a, 'b) aezfun ⇒ ('a, 'b) aezfun ⇒ ('a, 'b) aezfun
    is    λf g. f − g
    using diff-of-aezfun-is-aezfun
    by    fast
  instance ..
end

lemma minus-aezfun-apply [simp]: aezfun (a−b) x = aezfun a x − aezfun b x
  by transfer simp

instantiation aezfun :: (group-add, type) uminus
begin
  lift-definition uminus-aezfun :: ('a, 'b) aezfun ⇒ ('a, 'b) aezfun is λf. − f
    using neg-of-aezfun-is-aezfun by fast
  instance ..
end

lemma uminus-aezfun-apply [simp]: aezfun (−a) x = − aezfun a x
  by transfer simp

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

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

instance aezfun :: (group-add, type) group-add
proof
  fix a b :: ('a::group-add, 'b) aezfun
  show − a + a = 0 a + − b = a − b by 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 δ 0*
**using** *deltafun-is-aezfun* **by** *fast*
**instance** ..
**end**

**lemma** *one-aezfun-transfer* : *Abs-aezfun* (*1 δ 0*) = *1*
**proof**−
**define** *z n* **where** *z* = (*0*::*'b*::*zero*) **and** *n* = (*1*::*'a*::{*one*,*zero*})
**hence** *aezfun 1* = *n δ z* **using** *one-aezfun.rep-eq* **by** *fast*
**hence** *Abs-aezfun* (*n δ 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 δ 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* ≠ *0* ⟹ *aezfun 1 x* = *0*
**using** *deltafun-apply-neq* **by** *simp*

**instance** *aezfun* :: (*zero-neq-one*, *zero*) *zero-neq-one*
**proof**
**have** (*0*::*'a*) ≠ *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*) ≠ *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* ⇒ (*'a*, *'b*) *aezfun* ⇒ (*'a*, *'b*) *aezfun*
**is** *λ f g. convolution f g*
**using** *convolution-of-aezfun-is-aezfun*
**by** *fast*
**instance** ..

**end**

**lemma** *convolution-transfer* :
**assumes** $f \in$ *aezfun-set* $g \in$ *aezfun-set*
**shows** *Abs-aezfun* (*convolution f g*) = *Abs-aezfun f* ∗ *Abs-aezfun g*
**proof** (*rule aezfun-transfer*)
**from** *assms* **have** *aezfun* (*Abs-aezfun* (*convolution f g*)) = *convolution f g*
**using** *convolution-of-aezfun-is-aezfun Abs-aezfun-inverse* **by** *fast*
**moreover from** *assms*
**have** *aezfun* (*Abs-aezfun f* ∗ *Abs-aezfun g*) = *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** *aezfun* (*Abs-aezfun* (*convolution f g*))
= *aezfun* (*Abs-aezfun f* ∗ *Abs-aezfun g*)
**by** *simp*
**qed**

**instance** *aezfun* :: ({*comm-monoid-add*,*mult-zero*}, *group-add*) *mult-zero*
**proof**
**fix** *a* :: ('*a*, '*b*) *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*) *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*) *aezfun*
**show** *1* ∗ *a* = *a*
**proof**−
**have** *aezfun* (*1* ∗ *a*) = *convolution* (*1* δ *0*) (*aezfun a*) **by** *transfer fast*
**hence** *aezfun* (*1* ∗ *a*) = (*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** *aezfun* (*a* ∗ *1*) = *convolution* (*aezfun a*) (*1* δ *0*) **by** *transfer fast*
**hence** *aezfun* (*a* ∗ *1*) = (*aezfun a*)
**using** *one-neq-zero convolution-delta-right*[*of aezfun a*] **by** *simp*
**thus** *?thesis* **by** *transfer*
**qed**
**qed**

**instance** *aezfun* :: (*ring-1*, *group-add*) *ring-1* **..**

### 1.5.5 Transfer facts

**abbreviation** *aezdeltafun* :: *'b*::*zero* ⇒ *'a* ⇒ (*'b*,*'a*) *aezfun* (**infix** δδ *70*)
**where** *b* δδ *a* ≡ *Abs-aezfun* (*b* δ *a*)

**lemma** *aezdeltafun* : *aezfun* (*b* δδ *a*) = *b* δ *a*
**using** *deltafun-is-aezfun*[*of b a*] *Abs-aezfun-inverse* **by** *fast*

**lemma** *aezdeltafun-plus* : (*b*+*c*) δδ *a* = (*b* δδ *a*) + (*c* δδ *a*)
**using** *aezdeltafun*[*of b*+*c a*] *deltafun-plus aezdeltafun*[*of b a*] *aezdeltafun*[*of c a*]
*plus-aezfun.rep-eq*[*of b* δδ *a*]
*aezfun-transfer*[*of* (*b*+*c*) δδ *a* (*b* δδ *a*) + (*c* δδ *a*)]
**by** *fastforce*

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

**lemma** *aezfun-restrict-and-extend0* : (*aezfun x*)↓*A* ∈ *aezfun-set*
**using** *aezfun.aezfun restrict-and-extend0-aezfun-is-aezfun*[*of aezfun x*] **by** *fast*

**lemma** *aezdeltafun-decomp* :
**fixes** *b* :: *'b*::*semiring-1*
**shows** *b* δδ *a* = (*b* δδ *0*) ∗ (*1* δδ *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*::*semiring-1*
**shows** *b* δδ *a* = (*1* δδ *a*) ∗ (*b* δδ *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* :

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

**lemma** *supp-aezfun-diff* :
*supp (aezfun (x − y)) ⊆ supp (aezfun x) ∪ supp (aezfun y)*
**proof**−
**have** *supp (aezfun (x − y)) = supp ( (aezfun x) − (aezfun y) )* **by** *transfer fast*
**thus** *?thesis* **using** *supp-diff-subset-union-supp* **by** *fast*
**qed**

**lemma** *supp-aezfun-times* :
*supp (aezfun (x ∗ y)) ⊆ supp (aezfun x) + supp (aezfun y)*
**proof**−
**have** *supp (aezfun (x ∗ y)) = supp (convolution (aezfun x) (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 ⇒ ('b::zero,'a) aezfun set*
**where** *aezfun-setspan A = {x. supp (aezfun x) ⊆ A}*

**lemma** *aezfun-setspanD* : *x ∈ aezfun-setspan A ⟹ supp (aezfun x) ⊆ A*
**unfolding** *aezfun-setspan-def* **by** *fast*

**lemma** *aezfun-setspanI* : *supp (aezfun x) ⊆ A ⟹ x ∈ aezfun-setspan A*
**unfolding** *aezfun-setspan-def* **by** *fast*

**lemma** *aezfun-common-supp-spanning-set* :
**assumes** *finite A*
**shows** *∃ as. distinct as ∧ set as = A ∧ (*
*∀ x::('b::semiring-1,'a) aezfun ∈ aezfun-setspan A.*
*∃ bs. length bs = length as ∧ x = (∑ (b,a)←zip bs as. b δδ a)*
*)*
**proof**−
**from** *assms aezfun-common-supp-spanning-set'[of A]* **obtain** *as*
**where** *as*: *distinct as set as = A*
*∀ f::'a ⇒ 'b. supp f ⊆ A*
*⟶ (∃ bs. length bs = length as ∧ f = (∑ (b,a)←zip bs as. b δ a))*
**by** *fast*
**have** *⋀x::('b,'a) aezfun. x ∈ aezfun-setspan A ⟹*
*(∃ bs. length bs = length as ∧ x = (∑ (b,a)←zip bs as. b δδ a))*
**proof**−
**fix** *x::('b,'a) aezfun* **assume** *x ∈ aezfun-setspan A*
**with** *as(3)* **obtain** *bs*

```
    where bs: length bs = length as aezfun x = (∑ (b,a)←zip bs as. b δ a)
    using aezfun-setspanD
    by    fast
  have ⋀b a. (b,a) ∈ set (zip bs as) ⟹ b δ a = aezfun (b δδ a)
  proof−
    fix b a assume (b,a) ∈ set (zip bs as)
    show b δ a = aezfun (b δδ a) using aezdeltafun[of b a] by simp
  qed
  with bs show ∃ bs. length bs = length as ∧ x = (∑ (b,a)←zip bs as. b δδ a)
    using sum-list-prod-cong[of zip bs as deltafun λb a. aezfun (b δδ 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   ∃ gs. distinct gs ∧ set gs = G ∧ (
              ∀ x::('r::semiring-1,'g) aezfun ∈ aezfun-setspan G.
                ∃ rs. length rs = length gs
                  ∧ x = (∑ (r,g)←zip rs gs. (r δδ 0) * (1 δδ g))
         )
proof−
 from  aezfun-common-supp-spanning-set[OF assms] obtain gs
   where gs: distinct gs set gs = G
           ∀ x::('r,'g) aezfun ∈ aezfun-setspan G.
                ∃ rs. length rs = length gs
                  ∧ x = (∑ (r,g)←zip rs gs. r δδ g)
   by    fast
 have ⋀x::('r,'g) aezfun. x ∈ aezfun-setspan G
          ⟹ ∃ rs. length rs = length gs
            ∧ x = (∑ (r,g)←zip rs gs. (r δδ 0) * (1 δδ g))
 proof−
   fix x::('r,'g) aezfun assume x ∈ aezfun-setspan G
   with gs(3) obtain rs
     where length rs = length gs x = (∑ (r,g)←zip rs gs. r δδ g)
     using aezfun-setspanD
     by    fast
   thus ∃ rs. length rs = length gs
             ∧ x = (∑ (r,g)←zip rs gs. (r δδ 0) * (1 δδ g))
     using aezdeltafun-decomp sum-list-prod-cong[
           of zip rs gs λr g. r δδ g λr g. (r δδ 0) * (1 δδ g)
         ]
     by    auto
 qed
 with gs(1,2) show ?thesis by fast
```

**qed**

**lemma** *aezfun-decomp-aezdeltafun* :
**fixes** $c$ :: ($'r$::*semiring-1*,$'a$) *aezfun*
**shows** $\exists$ *ras. set* (*map snd ras*) = *supp* (*aezfun c*) $\wedge$ $c$ = ($\sum$ ($r$,$a$)$\leftarrow$*ras. r* $\delta\delta$ *a*)
**proof**−
**from** *aezfun-finite-supp*[*of c*] **obtain** *as*
**where** *as*: *set as* = *supp* (*aezfun c*)
$\forall$ $x$::($'r$,$'a$) *aezfun* $\in$ *aezfun-setspan* (*supp* (*aezfun c*)).
$\exists$ *bs. length bs* = *length as*
$\wedge$ $x$ = ($\sum$ ($b$,$a$)$\leftarrow$*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*: *length bs* = *length as c* = ($\sum$ ($b$,$a$)$\leftarrow$*zip bs as. b* $\delta\delta$ *a*)
**using** *aezfun-setspanI*[*of c supp* (*aezfun c*)]
**by** *fast*
**from** *bs*(*1*) *as*(*1*) **have** *set* (*map snd* (*zip bs as*)) = *supp* (*aezfun c*) **by** *simp*
**with** *bs*(*2*) **show** *?thesis* **by** *fast*
**qed**

**lemma** *aezfun-setspan-el-decomp-aezdeltafun* :
**fixes** $c$ :: ($'r$::*semiring-1*,$'a$) *aezfun*
**shows** $c \in$ *aezfun-setspan A*
$\Longrightarrow$ $\exists$ *ras. set* (*map snd ras*) $\subseteq$ $A$ $\wedge$ $c$ = ($\sum$ ($r$,$a$)$\leftarrow$*ras. r* $\delta\delta$ *a*)
**using** *aezfun-setspanD aezfun-decomp-aezdeltafun*
**by** *fast*

**lemma** *aezdelta0fun-commutes'* :
**fixes** *b1 b2* :: $'b$::*comm-semiring-1*
**shows** *b1* $\delta\delta$ *a* $*$ (*b2* $\delta\delta$ *0*) = *b2* $\delta\delta$ *0* $*$ (*b1* $\delta\delta$ *a*)
**using** *times-aezdeltafun-aezdeltafun*[*of b1 a*]
*times-aezdeltafun-aezdeltafun*[*of b2 0 b1 a*]
**by** (*simp add*: *algebra-simps*)

**lemma** *aezdelta0fun-commutes* :
**fixes** $b$ :: $'b$::*comm-semiring-1*
**shows** $c$ $*$ ($b$ $\delta\delta$ *0*) = $b$ $\delta\delta$ *0* $*$ $c$
**proof**−
**from** *aezfun-decomp-aezdeltafun* **obtain** *ras*
**where** *c*: $c$ = ($\sum$ ($r$,$a$)$\leftarrow$*ras. r* $\delta\delta$ *a*)
**by** *fast*
**thus** *?thesis*
**using** *sum-list-mult-const-prod*[*of* $\lambda r$ *a. r* $\delta\delta$ *a ras*] *aezdelta0fun-commutes'*
*sum-list-prod-cong*[*of 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 ras*]
**by** *auto*
**qed**

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

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

**definition** *aezfun-setspan-proj* :: *'a set* ⇒ (*'b*::*zero*,*'a*) *aezfun* ⇒ (*'b*::*zero*,*'a*) *aezfun*
**where** *aezfun-setspan-proj A x* ≡ *Abs-aezfun* ((*aezfun x*)↓*A*)

**lemma** *aezfun-setspan-projD1* :
*a* ∈ *A* ⟹ *aezfun* (*aezfun-setspan-proj A x*) *a* = *aezfun x a*
**using** *aezfun-restrict-and-extend0*[*of A x*] *Abs-aezfun-inverse*[*of* (*aezfun x*)↓*A*]
**unfolding** *aezfun-setspan-proj-def*
**by** *simp*

**lemma** *aezfun-setspan-projD2* :
*a* ∉ *A* ⟹ *aezfun* (*aezfun-setspan-proj A x*) *a* = *0*
**using** *aezfun-restrict-and-extend0*[*of A x*] *Abs-aezfun-inverse*[*of* (*aezfun x*)↓*A*]
**unfolding** *aezfun-setspan-proj-def*
**by** *simp*

**lemma** *aezfun-setspan-proj-in-setspan* :
*aezfun-setspan-proj A x* ∈ *aezfun-setspan A*
**using** *aezfun-setspan-projD2*[*of* - *A*]
*suppD-contra*[*of aezfun* (*aezfun-setspan-proj A x*)]
*aezfun-setspanI*[*of aezfun-setspan-proj A x A*]
**by** *auto*

**lemma** *aezfun-setspan-proj-zero* : *aezfun-setspan-proj A 0* = *0*
**proof**−
**have** *aezfun* (*aezfun-setspan-proj A 0*) = *aezfun 0*
**proof**
**fix** *a* **show** *aezfun* (*aezfun-setspan-proj A 0*) *a* = *aezfun 0 a*
**using** *aezfun-setspan-projD1*[*of a A 0*] *aezfun-setspan-projD2*[*of a A 0*]
**by** (*cases a*∈*A*) *auto*
**qed**
**thus** *?thesis* **using** *aezfun-transfer* **by** *fast*
**qed**

**lemma** *aezfun-setspan-proj-aezdeltafun* :
*aezfun-setspan-proj A* (*b* *δδ* *a*) = (*if a* ∈ *A then b* *δδ* *a else 0*)
**proof**−
**have** *aezfun* (*aezfun-setspan-proj A* (*b* *δδ* *a*))
= *aezfun* (*if a* ∈ *A then b* *δδ* *a else 0*)
**proof**
**fix** *x* **show** *aezfun* (*aezfun-setspan-proj A* (*b* *δδ* *a*)) *x*
= *aezfun* (*if a* ∈ *A then b* *δδ* *a else 0*) *x*
**proof** (*cases x* ∈ *A*)
**case** *True* **thus** *?thesis*
**using** *aezfun-setspan-projD1*[*of x A b* *δδ* *a*] *aezdeltafun*[*of b a*]
*deltafun-apply-neq*[*of x*]
**by** *fastforce*
**next**
**case** *False*

**hence** *aezfun (aezfun-setspan-proj A (b δδ a)) x = 0*
**using** *aezfun-setspan-projD2[of x A]* **by** *simp*
**moreover from** *False*
**have** *a ∈ A ⟹ aezfun (if a ∈ A then b δδ 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 ∈ A) auto*
**qed**
**thus** *?thesis* **using** *aezfun-transfer* **by** *auto*
**qed**

**lemma** *aezfun-setspan-proj-sum-list* :
*aezfun-setspan-proj A (∑ x←xs. f x) = (∑ x←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 (∑ (x,y)←xys. f x y)*
*= (∑ (x,y)←xys. aezfun-setspan-proj A (f x y))*
**using** *aezfun-setspan-proj-sum-list[of A λxy. case-prod f xy]*
*prod.case-distrib[of aezfun-setspan-proj A f]*
**by** *simp*

## 1.6 Polynomials

**lemma** *nonzero-coeffs-nonzero-poly* : *as ≠ [] ⟹ set as ≠ 0 ⟹ Poly as ≠ 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** $\bigwedge n$. *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 ≠ 0 ∨ p ≠ 0 ⟹ P* (*pCons a p*) *0*
**and** *rpCons*: $\bigwedge b\ q$. *b ≠ 0 ∨ q ≠ 0 ⟹ P 0* (*pCons b q*)
**and** *pCons2*: $\bigwedge a\ p\ b\ q$. *a ≠ 0 ∨ p ≠ 0 ⟹ b ≠ 0 ∨ q ≠ 0 ⟹ P p q*
*⟹ 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 ∈ A ⟹* ($\bigwedge a$. *a ∈ A ⟹ a = 0*) *⟹ 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* ⇒ *bool*
**where** *add-independentS* [] = *True*
| *add-independentS* (*A*#*As*) = ( *add-independentS As*
∧ (∀ *x*∈(∑ *B*←*As*. *B*). ∀ *a*∈*A*. *a* + *x* = *0* ⟶ *a* = *0*) )

**lemma** *add-independentS-doubleI*:
**assumes** ⋀*b a*. *b*∈*B* ⟹ *a*∈*A* ⟹ *a* + *b* = *0* ⟹ *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** ⋀*b a*. *b*∈*B* ⟹ *a*∈*A* ⟹ *a* + *b* = *0* ⟹ *a* = *0*
**using** *assms sum-list-sets-single*[*of B*] **by** *simp*

**lemma** *add-independentS-double-iff* :
*add-independentS* [*A*,*B*] = (∀ *b*∈*B*. ∀ *a*∈*A*. *a* + *b* = *0* ⟶ *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*,∑ *B*←*As*. *B*] ∧ *add-independentS As*)
**using** *add-independentS-double-iff*[*of A*] **by** *auto*

**lemma** *add-independentS-double-sum-conv-append* :
⟦ ∀ *X*∈*set As*. *0* ∈ *X*; *add-independentS As*; *add-independentS Bs*;
*add-independentS* [∑ *X*←*As*. *X*, ∑ *X*←*Bs*. *X*] ⟧
⟹ *add-independentS* (*As*@*Bs*)
**proof** (*induct As*)
**case** (*Cons A As*)
**have** *add-independentS* [∑ *X*←*As*. *X*, ∑ *X*←*Bs*. *X*]
**proof** (*rule add-independentS-doubleI*)
**fix** *b a* **assume** *ba*: *b* ∈ (∑ *X*←*Bs*. *X*) *a* ∈ (∑ *X*←*As*. *X*) *a* + *b* = *0*
**from** *Cons*(*2*) *ba*(*2*) **have** *a* ∈ (∑ *X*←*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* ∑ *X*←*A* # *As*. *X* ∑ *X*←*Bs*. *X b a*]
**by** *simp*
**qed**
**moreover have** ⋀*x a*. ⟦ *x* ∈ (∑ *X*←*As*@*Bs*. *X*); *a* ∈ *A*; *a* + *x* = *0* ⟧
⟹ *a* = *0*
**proof**−
**fix** *x a* **assume** *x-a*: *x* ∈ (∑ *X*←*As*@*Bs*. *X*) *a* ∈ *A a* + *x* = *0*
**from** *x-a*(*1*) **obtain** *xa xb*
**where** *xa-xb*: *x* = *xa* + *xb xa* ∈ (∑ *X*←*As*. *X*) *xb* ∈ (∑ *X*←*Bs*. *X*)
**using** *set-plus-elim*[*of x* ∑ *X*←*As*. *X*]
**by** *auto*
**from** *x-a*(*2*) *xa-xb*(*2*) **have** *a* + *xa* ∈ *A* + (∑ *X*←*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 ∑X←A # As. X ∑X←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
       ⋀x a. ⟦ x∈(∑X←As. X); a ∈ A; a+x = 0 ⟧ ⟹ a = 0
 shows   add-independentS (A#As)
 using assms by simp

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

lemma add-independentS-append-reduce-left :
 add-independentS (As@Bs) ⟹ 0 ∈ (∑X←Bs. X) ⟹ 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 ∈ (∑X←As. X) and a: a ∈ A and sum: a+x = 0
   from x Cons(3) have x + 0 ∈ (∑X←As. X) + (∑X←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) ⟹ add-independentS [∑X←As. X, ∑X←Bs. X]
proof (induct As)
 case (Cons A As)
 show add-independentS [∑X←(A#As). X, ∑X←Bs. X]
 proof (rule add-independentS-doubleI)
   fix b x assume bx: b ∈ (∑X←Bs. X) x ∈ (∑X←A # As. X) x + b = 0
   from bx(2) obtain a as
     where a-as: a ∈ A as ∈ listset As x = a + (∑z←as. z)
     using set-sum-list-element-Cons
     by    fast
   from Cons(2) have add-independentS [A,∑X←As@Bs. X]
     using add-independentS-Cons-conv-sum-right[of A As@Bs] by simp
   moreover from a-as(2) bx(1)
     have  (∑z←as. z) + b ∈ (∑X←(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* ⇒ *'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* :: *pttrn => 'a list => 'b => 'b*
(($3\bigoplus$ -←-. -) [*0*, *51*, *10*] *10*)
**translations** — Beware of argument permutation!
$\bigoplus M{\leftarrow}Ms.\ b$ == *CONST inner-dirsum* (*CONST map* (%*M*. *b*) *Ms*)

**abbreviation** *inner-dirsum-double* ::
*'a::monoid-add set* ⇒ *'a set* ⇒ *'a set* (**infixr** ⊕ *70*)
**where** *inner-dirsum-double A B* ≡ *inner-dirsum* [*A*,*B*]

**lemma** *inner-dirsumI* :
$M = (\sum N{\leftarrow}Ns.\ N) \Longrightarrow$ *add-independentS Ns* $\Longrightarrow M = (\bigoplus N{\leftarrow}Ns.\ N)$
**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* $\Longrightarrow (\bigoplus M{\leftarrow}Ms.\ M) = (\sum M{\leftarrow}Ms.\ M)$
**unfolding** *inner-dirsum-def* **by** *simp*

**lemma** *inner-dirsumD2* : ¬ *add-independentS Ms* $\Longrightarrow (\bigoplus M{\leftarrow}Ms.\ M) = 0$
**unfolding** *inner-dirsum-def* **by** *simp*

**lemma** *inner-dirsum-Nil* : $(\bigoplus A{\leftarrow}[].\ A) = 0$
**unfolding** *inner-dirsum-def* **by** *simp*

**lemma** *inner-dirsum-singleD* : $(\bigoplus N{\leftarrow}[M].\ N) = M$
**using** *inner-dirsumD*[*of* [*M*]] *sum-list-sets-single*[*of M*] **by** *simp*

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

**lemma** *inner-dirsum-Cons* :
*add-independentS* (*A* # *As*) $\Longrightarrow$ $(\bigoplus X{\leftarrow}(A\#As).\ X) = A \oplus (\bigoplus X{\leftarrow}As.\ X)$
**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*) $\Longrightarrow$ $0 \in (\sum X{\leftarrow}Bs.\ X)$
$\Longrightarrow$ $(\bigoplus X{\leftarrow}(As@Bs).\ X) = (\bigoplus X{\leftarrow}As.\ X) \oplus (\bigoplus X{\leftarrow}Bs.\ X)$
**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* $\Longrightarrow$ *as* $\in$ *listset As* $\Longrightarrow$ *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*: $\bigwedge g\ h.\ g \in G \Longrightarrow h \in G \Longrightarrow g - h \in G$

**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* $\Longrightarrow$ *FinGroup G* **by** *unfold-locales*

**context** *Group*
**begin**

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

**lemma** *SubgroupD1* : *Subgroup H* $\Longrightarrow$ *Group H* **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* $\Longrightarrow$ −*g* $\in$ *G*
  **using** *zero-closed diff-closed*[*of 0 g*] **by** *simp*

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

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

**lemma** *sum-list-closed* : *set* (*map f as*) $\subseteq$ *G* $\Longrightarrow$ ($\sum$ *a*←*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* $\Longrightarrow$ ($\sum$ (*x*,*y*)←*xys*. *f x y*) $\in$ *G*
  **using** *sum-list-closed* **by** *fast*

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

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

**lemma** *list-diff-closed* :
*set gs* $\subseteq$ *G* $\Longrightarrow$ *set hs* $\subseteq$ *G* $\Longrightarrow$ *set* [*x*−*y*. (*x*,*y*)←*zip gs hs*] $\subseteq$ *G*
**using** *diff-closed* **by** (*induct gs hs rule*: *list-induct2'*) *auto*

**lemma** *add-closed-converse-right* : *g*+*x* ∈ *G* $\Longrightarrow$ *g* ∈ *G* $\Longrightarrow$ *x* ∈ *G*
**using** *neg-add-closed add.assoc*[*of* −*g g x*] **by** *fastforce*

**lemma** *add-closed-inverse-right* : *x* ∉ *G* $\Longrightarrow$ *g* ∈ *G* $\Longrightarrow$ *g*+*x* ∉ *G*
**using** *add-closed-converse-right* **by** *fast*

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

**lemma** *add-closed-inverse-left* : *g* ∉ *G* $\Longrightarrow$ *x* ∈ *G* $\Longrightarrow$ *g*+*x* ∉ *G*
**using** *add-closed-converse-left* **by** *fast*

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

**lemma** *right-translate-sum* : *g* ∈ *G* $\Longrightarrow$ ($\sum$ *h*∈*G*. *f h*) = ($\sum$ *h*∈*G*. *f* (*h* + *g*))
**using** *right-translate-bij*[*of g*] *bij-betw-def*[*of* λ*h*. *h* + *g*]
*sum.reindex*[*of* λ*h*. *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* $A \Longrightarrow f\ `\ A \subseteq G \Longrightarrow (\sum a{\in}A.\ f\ a) \in G$
**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* :: $'g\ set \Rightarrow ('g{\times}'g)\ set$
**where** *rcoset-rel H* $\equiv \{(g,g').\ g \in G \wedge g' \in G \wedge g - g' \in H\}$

**lemma** (**in** *Group*) *rcosets* :
**assumes** *subgrp*: *Subgroup H* **and** *g*: $g \in G$
**shows** (*rcoset-rel H*)``{*g*} = *H* + {*g*}
**proof** (*rule seteqI*)
**fix** *x* **assume** $x \in$ (*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** $x \in$ (*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) ⊆ G × G using rcoset-rel-def by auto
  next
    fix x assume x ∈ G
    with assms show (x,x) ∈ (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) ∈ (rcoset-rel H)
    with assms show (b,a) ∈ (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) ∈ (rcoset-rel H) (y,z) ∈ (rcoset-rel H)
    with assms show (x,z) ∈ (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 ⟹ (rcoset-rel H)``{0} = H
  using zero-closed rcosets unfolding set-plus-def by simp

definition is-rcoset-replist :: 'g set ⇒ 'g list ⇒ bool
  where is-rcoset-replist H gs
            ≡ set gs ⊆ G ∧ distinct (map (λg. (rcoset-rel H)``{g}) gs)
              ∧ G = (⋃g∈set gs. (rcoset-rel H)``{g})

lemma is-rcoset-replistD-set : is-rcoset-replist H gs ⟹ set gs ⊆ G
  unfolding is-rcoset-replist-def by fast

lemma is-rcoset-replistD-distinct :
  is-rcoset-replist H gs ⟹ distinct (map (λg. (rcoset-rel H)``{g}) gs)
  unfolding is-rcoset-replist-def by fast

lemma is-rcoset-replistD-cosets :
  is-rcoset-replist H gs ⟹ G = (⋃g∈set gs. (rcoset-rel H)``{g})
  unfolding is-rcoset-replist-def by fast

lemma group-eq-subgrp-rcoset-un :
  Subgroup H ⟹ is-rcoset-replist H gs ⟹ G = (⋃g∈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** $\bigwedge i\ j.\ i < length\ gs \Longrightarrow j < length\ gs \Longrightarrow i \neq j \Longrightarrow gs!i - gs!j \notin H$
**proof**
**fix** *i j* **assume** *ij*: $i < length\ gs\ j < length\ gs\ i{\neq}j\ gs!i - gs!j \in H$
**from** *assms*(2) *ij*(1,2,4) **have** $(gs!i, gs!j) \in rcoset\text{-}rel\ H$
**using** *set-conv-nth is-rcoset-replistD-set rcoset-rel-def* **by** *fastforce*
**with** *assms*(1) *ij*(1,2)
**have** $(map\ (\lambda g.\ (rcoset\text{-}rel\ H)``\{g\})\ gs)!i$
$= (map\ (\lambda g.\ (rcoset\text{-}rel\ H)``\{g\})\ gs)!j$
**using** *rcoset-equiv equiv-class-eq*
**by** *fastforce*
**with** *assms*(2) *ij*(1−3) **show** *False*
**using** *is-rcoset-replistD-distinct distinct-conv-nth*[
*of map* $(\lambda g.\ (rcoset\text{-}rel\ H)``\{g\})$ *gs*
]
**by** *auto*
**qed**

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

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

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

**lemma** *rcoset-replist-replacehd* :
**assumes** *Subgroup H* $g' \in (rcoset\text{-}rel\ H)``\{g\}$ *is-rcoset-replist H* $(g\ \#\ gs)$
**shows** *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** $set\ gs \subseteq G$

```
    using is-rcoset-replistD-set by fastforce
  moreover from assms(1−3)
    have (rcoset-rel H)‘‘{g'} ∉ set (map (λx. (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 (λg. (rcoset-rel H)‘‘{g}) gs)
    using is-rcoset-replistD-distinct by fastforce
  moreover from assms(1−3)
    have  G = (rcoset-rel H)‘‘{g'} ∪ (⋃x∈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   ∃ 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 ∀ x∈G//r. ∃ g. g ∈ x
  proof
    fix x assume x ∈ G//r
    from this obtain g where g: g ∈ G x = r‘‘{g}
      using quotient-def[of G r] by auto
    hence g ∈ x using equiv-r equivE[of G r] refl-onD[of G r] by auto
    thus ∃ g. g ∈ x by fast
  qed
  from this obtain f where f: ∀ x∈G//r. f x ∈ x using bchoice by force
  from r-def have r ⊆ G × 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 ⊆ G
  proof
    fix g assume g ∈ set gs
    with gs(2) obtain x where x: x ∈ G//r g = f x by fast
    with f show g ∈ G using equiv-r Union-quotient by fast
  qed

  moreover have distinct (map (λg. r‘‘{g}) gs)
  proof−
    have ⋀i j. ⟦ i < length (map (λg. r‘‘{g}) gs);
                 j < length (map (λg. r‘‘{g}) gs); i ≠ j ⟧
                   ⟹ (map (λg. r‘‘{g}) gs)!i ≠ (map (λg. r‘‘{g}) gs)!j
    proof
```

```
    fix i j assume ij:
      i < length (map (λg. r``{g}) gs)
      j< length (map (λg. r``{g}) gs)
      i ≠ j
      (map (λg. r``{g}) gs)!i = (map (λg. r``{g}) gs)!j
    from ij(1,2) have gs!i ∈ set gs gs!j ∈ set gs using set-conv-nth by auto
    from this gs(2) obtain x y
      where x: x ∈ G//r gs!i = f x and y: y ∈ 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) ∈ r
        using eq-equiv-class-iff[OF equiv-r] by force
      from x y f show gs!i ∈ x gs!j ∈ 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 = (⋃ g∈set gs. r``{g})
 proof (rule subset-antisym, rule subsetI)
  fix g assume g ∈ G
  hence rg: r``{g} ∈ G//r using quotientI by fast
  with gs(2) obtain g' where g': g' ∈ set gs g' = f (r``{g}) by fast
  from f g'(2) rg have g ∈ r``{g'} using equiv-r equivE sym-def[of r] by force
  with g'(1) show g ∈ (⋃ g∈set gs. r``{g}) by fast
 next
  from r-def show G ⊇ (⋃ g∈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   ∃ gs. is-rcoset-replist H (0#gs)
proof−
 from assms obtain xs where xs: is-rcoset-replist H xs
  using ex-rcoset-replist by fast
 with assms obtain x where x: x ∈ set xs x ∈ H
  using rcoset-replist-Hrep by fast
 from x(2) have (0,x) ∈ rcoset-rel H using rcoset0[OF assms] by auto
 moreover have sym (rcoset-rel H)
  using rcoset-equiv[OF assms] equivE[of G rcoset-rel H] by simp
 ultimately have 0 ∈ (rcoset-rel H)``{x} using sym-def by fast
 with x(1) xs assms show ∃ gs. 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⇒'b::zero*) ⇒ *'a set*
**where** *ker f* = {*a*. *f a* = *0*}

**lemma** *kerI* : *f a* = *0* ⟹ *a* ∈ *ker f*
**unfolding** *ker-def* **by** *fast*

**lemma** *kerD* : *a* ∈ *ker f* ⟹ *f a* = *0*
**unfolding** *ker-def* **by** *fast*

**lemma** *ker-im-iff* : (*A* ≠ {} ∧ *A* ⊆ *ker f*) = (*f* ‘ *A* = *0*)
**proof**
**assume** *A*: *A* ≠ {} ∧ *A* ⊆ *ker f*
**show** *f* ‘ *A* = *0*
**proof** (*rule zeroset-eqI*)
**from** *A* **obtain** *a* **where** *a*: *a* ∈ *A* **by** *fast*
**with** *A* **have** *f a* = *0* **using** *kerD* **by** *fastforce*
**from** *this*[*THEN sym*] *a* **show** *0* ∈ *f* ‘ *A* **by** *fast*
**next**
**fix** *b* **assume** *b* ∈ *f* ‘ *A*
**from** *this* **obtain** *a* **where** *a* ∈ *A b* = *f a* **by** *fast*
**with** *A* **show** *b* = *0* **using** *kerD* **by** *fast*
**qed**
**next**
**assume** *fA*: *f* ‘ *A* = *0*
**have** *A* ≠ {}
**proof**−
**from** *fA* **obtain** *a* **where** *a* ∈ *A f a* = *0* **by** *force*
**thus** *?thesis* **by** *fast*
**qed**
**moreover have** *A* ⊆ *ker f*
**proof**
**fix** *a* **assume** *a* ∈ *A*
**with** *fA* **have** *f a* = *0* **by** *auto*
**thus** *a* ∈ *ker f* **using** *kerI* **by** *fast*
**qed**
**ultimately show** *A* ≠ {} ∧ *A* ⊆ *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* : ⋀*g g'*. *g* ∈ *G* ⟹ *g'* ∈ *G* ⟹ *T* (*g* + *g'*) = *T g* + *T g'*
  **and** *supp*: *supp T* ⊆ *G*

**abbreviation** (**in** *GroupHom*) *Ker* ≡ *ker T* ∩ *G*
**abbreviation** (**in** *GroupHom*) *ImG* ≡ *T* ‘ *G*

**locale** *GroupEnd* = *GroupHom G T*
  **for** *G* :: *'g*::*group-add set*
  **and** *T* :: *'g* ⇒ *'g*
+ **assumes** *endomorph*: *ImG* ⊆ *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*↓*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* ∈ *Ker*
  **using** *im-zero kerI zero-closed* **by** *fast*

**lemma** *comp-zero* : *T* ∘ *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-contra*[*of g T*] *suppI-contra*[*of* −*g T*]
  **by** *fastforce*

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

**lemma** *eq-im-imp-diff-in-Ker* : $[\![\ g \in G;\ h \in G;\ T\ g = T\ h\ ]\!] \Longrightarrow g - h \in Ker$
**using** *im-diff kerI diff-closed*[*of g h*] **by** *force*

**lemma** *im-sum-list-prod* :
*set* (*map* (*case-prod f*) *xys*) $\subseteq G$
$\Longrightarrow T\ (\sum (x,y) \leftarrow xys.\ f\ x\ y) = (\sum (x,y) \leftarrow xys.\ T\ (f\ x\ y))$
**proof** (*induct xys*)
**case** *Nil*
**show** *?case* **using** *im-zero* **by** *simp*
**next**
**case** (*Cons xy xys*)
**define** *Tf* **where** $Tf = T \circ (case\text{-}prod\ f)$
**have** $T\ (\sum (x,y) \leftarrow (xy \# xys).\ f\ x\ y) = T\ (\ (case\text{-}prod\ f)\ xy + (\sum (x,y) \leftarrow xys.\ f\ x\ y)\ )$
**by** *simp*
**moreover from** *Cons*(*2*) **have** (*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** $\ldots = (\sum (x,y) \leftarrow (xy \# xys).\ Tf\ (x,y))$ **by** *simp*
**finally show** *?case* **using** *Tf-def* **by** *simp*
**qed**

**lemma** *distrib-comp-sum-left* :
*range* $S \subseteq G \Longrightarrow$ *range* $S' \subseteq G \Longrightarrow T \circ (S + S') = (T \circ S) + (T \circ S')$
**using** *hom* **by** (*auto simp add*: *fun-eq-iff*)

**lemma** *Ker-Im-iff* : $(Ker = G) = (ImG = 0)$
**using** *nonempty ker-im-iff*[*of G T*] **by** *fast*

**lemma** *Ker0-imp-inj-on* :
**assumes** $Ker \subseteq 0$
**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** $x - y \in Ker$ **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** $Ker = 0$
**using** *zero-in-Ker kerD zero-closed im-zero inj-onD*[*OF assms*]

**by** *fastforce*

**lemma** *nonzero-Ker-el-imp-n-inj* :
$g \in G \Longrightarrow g \neq 0 \Longrightarrow T\ g = 0 \Longrightarrow \neg$ *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* $(T \downarrow H)$
**proof** (*intro-locales*, *rule SubgroupD1*[*OF assms*], *unfold-locales*)
**show** *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* $((T \downarrow H)\ `\ H)$
**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* $(S \circ T)$
**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** $\bigwedge g.\ g \notin G \Longrightarrow (S \circ T)\ g = 0$
**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* : $T\ `\ G \subseteq H \Longrightarrow (id{\downarrow}H) \circ T = T$
**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** $\bigwedge g.\ g \in G \Longrightarrow T\ (T\ g) = T\ g$
**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* ∈ *ImG g* ∈ *Ker g* + *h* = *0*
**from** *gh*(*1*) **obtain** *g′* **where** *g′* ∈ *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** *invT* ≡ (*the-inv-into G T*) ↓ *H*

**lemma** *ImG* : *ImG* = *H* **using** *bijective bij-betw-imp-surj-on* **by** *fast*

**lemma** *GroupH* : *Group H* **using** *ImG Group-Im* **by** *fast*

**lemma** *invT-onto* : *invT* ‘ *H* = *G*
**using** *bijective bij-betw-imp-inj-on*[*of T*] *ImG the-inv-into-onto*[*of T*] **by** *force*

**lemma** *inj-on-invT* : *inj-on invT H*
**using** *bijective bij-betw-imp-inj-on*[*of T G*] *ImG inj-on-the-inv-into*[*of T*]
**unfolding** *inj-on-def*
**by** *force*

**lemma** *bijective-invT* : *bij-betw invT H G*
**using** *inj-on-invT invT-onto* **unfolding** *bij-betw-def* **by** *fast*

**lemma** *invT-into* : *h* ∈ *H* ⟹ *invT h* ∈ *G*
**using** *bijective bij-betw-imp-inj-on ImG the-inv-into-into*[*of T*] **by** *force*

**lemma** *T-invT* : *h* ∈ *H* ⟹ *T* (*invT h*) = *h*
**using** *bijective bij-betw-imp-inj-on ImG f-the-inv-into-f*[*of T*] **by** *force*

**lemma** *invT-eq*: *g* ∈ *G* ⟹ *T g* = *h* ⟹ *invT h* = *g*
**using** *bijective bij-betw-imp-inj-on ImG the-inv-into-f-eq*[*of T*] **by** *force*

**lemma** *inv* : *GroupIso H invT G*
**proof** (*intro-locales*, *rule GroupH*, *unfold-locales*)
**show** *supp invT* ⊆ *H* **using** *supp-restrict0* **by** *fast*
**show** *bij-betw invT H G* **using** *bijective-invT* **by** *fast*
**next**
**fix** *h h′* **assume** *h* ∈ *H h′* ∈ *H*
**thus** *invT* (*h* + *h′*) = *invT h* + *invT 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* ⇒ *'h::group-add set* ⇒ (*'g* ⇒ *'h*) *set*
**where** *GroupHomSet G H* ≡ {*T*. *GroupHom G T*} ∩ {*T*. *T* ‘ *G* ⊆ *H*}

**lemma** *GroupHomSetI* :
*GroupHom G T* ⟹ *T* ‘ *G* ⊆ *H* ⟹ *T* ∈ *GroupHomSet G H*
**unfolding** *GroupHomSet-def* **by** *fast*

**lemma** *GroupHomSetD-GroupHom* :
*T* ∈ *GroupHomSet G H* ⟹ *GroupHom G T*
**unfolding** *GroupHomSet-def* **by** *fast*

**lemma** *GroupHomSetD-Im* : *T* ∈ *GroupHomSet G H* ⟹ *T* ‘ *G* ⊆ *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* ≠ {}
**using** *trivial-GroupHom AbGroup.zero-closed*[*OF assms*] *GroupHomSetI*
**by** *fastforce*
**next**
**fix** *S T* **assume** *ST*: *S* ∈ *GroupHomSet G H T* ∈ *GroupHomSet G H*
**show** *S* − *T* ∈ *GroupHomSet G H*
**proof** (*rule GroupHomSetI*, *unfold-locales*)
**from** *ST* **show** *supp* (*S* − *T*) ⊆ *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* ⊆ *H*
**proof** (*rule image-subsetI*)
**fix** *g* **assume** *g* ∈ *G*
**with** *ST* **have** *S g* ∈ *H T g* ∈ *H*
**using** *GroupHomSetD-Im*[*of S G*] *GroupHomSetD-Im*[*of T G*] **by** *auto*
**thus** (*S* − *T*) *g* ∈ *H* **using** *AbGroup.diff-closed*[*OF assms*] **by** *simp*
**qed**
**next**
**fix** *g g'* **assume** *g* ∈ *G g'* ∈ *G*
**with** *ST* **show** (*S* − *T*) (*g* + *g'*) = (*S* − *T*) *g* + (*S* − *T*) *g'*
**using** *GroupHomSetD-GroupHom*[*of S*] *GroupHom.hom*[*of G S*]

*GroupHomSetD-GroupHom*[*of T*] *GroupHom.hom*[*of G T*]
**by** *simp*
**qed**
**qed**

## 2.4 Facts about collections of groups

**lemma** *listset-Group-plus-closed* :
⟦ ∀ *G*∈*set Gs*. *Group G*; *as* ∈ *listset Gs*; *bs* ∈ *listset Gs*⟧
⟹ [*a*+*b*. (*a*,*b*)←*zip as bs*] ∈ *listset Gs*
**proof**−
**have** ⟦ *length as* = *length bs*; *length bs* = *length Gs*;
*as* ∈ *listset Gs*; *bs* ∈ *listset Gs*; ∀ *G*∈*set Gs*. *Group G*⟧
⟹ [*a*+*b*. (*a*,*b*)←*zip as bs*] ∈ *listset Gs*
**proof** (*induct as bs Gs rule*: *list-induct3*)
**case** (*Cons a as b bs G Gs*)
**thus** [*x*+*y*. (*x*,*y*)←*zip* (*a*#*as*) (*b*#*bs*)] ∈ *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** ⟦ ∀ *G*∈*set Gs*. *Group G*; *as* ∈ *listset Gs*; *bs* ∈ *listset Gs*⟧
⟹ [*a*+*b*. (*a*,*b*)←*zip as bs*] ∈ *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* ≠ {} **using** *AbGroup.nonempty* **by** *blast*
**next**
**fix** *x y* **assume** *x* ∈ *H* + *G y* ∈ *H* + *G*
**from** *this* **obtain** *xh xg yh yg*
**where** *xy*: *xh* ∈ *H xg* ∈ *G x* = *xh* + *xg yh* ∈ *H yg* ∈ *G y* = *yh* + *yg*
**unfolding** *set-plus-def* **by** *fast*
**hence** *x* − *y* = (*xh* − *yh*) + (*xg* − *yg*) **by** *simp*
**thus** *x* − *y* ∈ *H* + *G* **using** *assms xy*(*1*,*2*,*4*,*5*) *AbGroup.diff-closed* **by** *auto*
**qed**

**lemma** *AbGroup-sum-list* :
(∀ *G*∈*set Gs*. *AbGroup G*) ⟹ *AbGroup* (∑ *G*←*Gs*. *G*)
**using** *trivial-Group AbGroup.intro AbGroup-set-plus*
**by** (*induct Gs*) *auto*

**lemma** *AbGroup-subset-sum-list* :
∀ *G* ∈ *set Gs*. *AbGroup G* ⟹ *H* ∈ *set Gs* ⟹ *H* ⊆ (∑ *G*←*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* :
$[\![\ \forall\, G{\in}set\ Gs.\ AbGroup\ G;\ add\text{-}independentS\ Gs;\ G \in set\ Gs;\ G' \in set\ Gs;$
$G \neq G'\ ]\!] \Longrightarrow G \cap G' = 0$
**proof** (*induct Gs arbitrary*: $G\ G'$)
**case** (*Cons H Hs*)
**from** *Cons*(*1*−*3*) **have** $\bigwedge A\ B.\ [\![\ A \in set\ Hs;\ B \in set\ Hs;\ A \neq B\ ]\!]$
$\Longrightarrow A \cap B \subseteq 0$
**by** *simp*
**moreover have** $\bigwedge A.\ A \in set\ Hs \Longrightarrow A \neq H \Longrightarrow\ A \cap H \subseteq 0$
**proof**
**fix** $A\ g$ **assume** $A$: $A \in set\ Hs\ A \neq H$ **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** $\bigwedge A\ B.\ [\![\ A \in set\ (H\#Hs);\ B \in set\ (H\#Hs);\ A \neq B\ ]\!]$
$\Longrightarrow A \cap B \subseteq 0$
**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** $\forall\, G{\in}set\ [G,G'].\ AbGroup\ G$ **by** *simp*
**from** *assms*(*3*) **show** *add-independentS* [*G*,*G'*] **by** *fast*
**show** $G \in set\ [G,G']\ G' \in set\ [G,G']$ **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* :
$\forall G \in set\ Gs.\ AbGroup\ G \Longrightarrow AbGroup\ (\bigoplus G \leftarrow Gs.\ G)$
**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** $\bigwedge b.\ b \in B \Longrightarrow b = 0$
**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* :
$[\![\ \forall G \in set\ Gs.\ AbGroup\ G;\ add\text{-}independentS\ Gs;\ H \in set\ Gs\ ]\!]$
$\Longrightarrow H \subseteq (\bigoplus G \leftarrow Gs.\ G)$
**using** *AbGroup-subset-sum-list inner-dirsumD* **by** *fast*

**lemma** *AbGroup-nth-subset-inner-dirsum* :
⟦ ∀ *G* ∈ *set Gs*. *AbGroup G*; *add-independentS Gs*; *n* < *length Gs* ⟧
⟹ *Gs*!*n* ⊆ (⨁ *G*←*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* ∈ *G* ⊕ *H*
**shows** ∃!*gh*. *fst gh* ∈ *G* ∧ *snd gh* ∈ *H* ∧ *x* = *fst gh* + *snd gh*
**proof** (*rule ex-ex1I*)
**from** *assms*(*3*,*4*) **obtain** *g h* **where** *x* = *g* + *h* *g* ∈ *G* *h* ∈ *H*
**using** *inner-dirsum-doubleD set-plus-elim* **by** *blast*
**from** *this* **have** *1*: *fst* (*g*,*h*) ∈ *G* *snd* (*g*,*h*) ∈ *H* *x* = *fst* (*g*,*h*) + *snd* (*g*,*h*)
**by** *auto*
**thus** ∃ *gh*. *fst gh* ∈ *G* ∧ *snd gh* ∈ *H* ∧ *x* = *fst gh* + *snd gh* **by** *fast*
**next**
**fix** *gh gh*′ **assume** *A*:
*fst gh* ∈ *G* ∧ *snd gh* ∈ *H* ∧ *x* = *fst gh* + *snd gh*
*fst gh*′ ∈ *G* ∧ *snd gh*′ ∈ *H* ∧ *x* = *fst gh*′ + *snd gh*′
**show** *gh* = *gh*′
**proof**
**from** *A assms*(*1*,*2*) **have** *fst gh* − *fst gh*′ ∈ *G* *snd gh* − *snd gh*′ ∈ *H*
**using** *AbGroup.diff-closed* **by** *auto*
**moreover from** *A* **have** *z*: (*fst gh* − *fst gh*′) + (*snd gh* − *snd gh*′) = *0*
**by** (*simp add*: *algebra-simps*)
**ultimately show** *fst gh* = *fst gh*′
**using** *assms*(*3*)
*add-independentS-doubleD*[*of G H snd gh* − *snd gh*′ *fst gh* − *fst gh*′]
**by** *simp*
**with** *z* **show** *snd gh* = *snd gh*′ **by** *simp*
**qed**
**qed**

**lemma** *AbGroup-inner-dirsum-el-decomp-ex1* :
⟦ ∀ *G* ∈ *set Gs*. *AbGroup G*; *add-independentS Gs* ⟧
⟹ ∀ *x* ∈ (⨁ *G*←*Gs*. *G*). ∃!*gs*∈*listset Gs*. *x* = *sum-list gs*
**proof** (*induct Gs*)
**case** *Nil*
**have** ⋀*x*::′*a*. *x* ∈ (⨁ *H*←[]. *H*) ⟹ ∃!*gs*∈*listset* []. *x* = *sum-list gs*
**proof**
**fix** *x*::′*a* **assume** *x* ∈ (⨁ *G*←[]. *G*)
**moreover define** *f* :: ′*a* ⇒ ′*a list* **where** *f x* = [] **for** *x*
**ultimately show** *f x* ∈ *listset* [] ∧ *x* = *sum-list* (*f x*)
**using** *inner-dirsum-Nil* **by** *auto*
**next**
**fix** *x*::′*a* **and** *gs*
**assume** *x*: *x* ∈ (⨁ *G*←[]. *G*)
**and** *gs*: *gs* ∈ *listset* [] ∧ *x* = *sum-list gs*
**thus** *gs* = [] **by** *simp*

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

**lemma** *AbGroup-inner-dirsum-pairwise-int0* :
$[\![ \forall G \in set\ Gs.\ AbGroup\ G;\ add\text{-}independentS\ Gs;\ G \in set\ Gs;\ G' \in set\ Gs;$
$G \neq G' ]\!] \Longrightarrow G \cap G' = 0$
**proof** (*induct Gs arbitrary*: *G G'*)
**case** (*Cons H Hs*)
**from** *Cons(1−3)* **have** $\bigwedge A\ B.\ [\![ A \in set\ Hs;\ B \in set\ Hs;\ A \neq B ]\!] \Longrightarrow A \cap B \subseteq 0$
**by** *simp*
**moreover have** $\bigwedge A.\ A \in set\ Hs \Longrightarrow A \neq H \Longrightarrow A \cap H \subseteq 0$
**proof**
**fix** *A g* **assume** *A*: $A \in set\ Hs\ A \neq H$ **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** $\bigwedge A\ B.\ [\![ A \in set\ (H\#Hs);\ B \in set\ (H\#Hs);\ A \neq B ]\!] \Longrightarrow A \cap B \subseteq 0$
**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** $\forall G{\in}set\ [G,G'].\ AbGroup\ G$ **by** *simp*
**from** *assms(3)* **show** *add-independentS* $[G,G']$ **by** *fast*
**show** $G \in set\ [G,G']\ G' \in set\ [G,G']$ **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* ::
$'g$::*ab-group-add set list* $\Rightarrow$ ($'g \Rightarrow$ $'g$ *list*) ($\bigoplus$ -$\leftarrow$)
**where** $\bigoplus Gs\leftarrow = (\lambda x.$ *if* $x \in (\bigoplus G\leftarrow Gs.\ G)$
*then THE gs.* $gs \in$ *listset* $Gs \wedge x =$ *sum-list gs else* [])

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

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

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

**lemma** (**in** *AbGroup*) *abSubgroup-inner-dirsum-el-decomp-set* :
$[\![\ \forall H \in$ *set Hs. Subgroup H*; *add-independentS Hs*; $x \in (\bigoplus H\leftarrow Hs.\ H)\ ]\!]$
$\Longrightarrow$ *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* :
$[\![\ \forall G \in$ *set Gs. AbGroup G*; *add-independentS Gs*; $x \in (\bigoplus G\leftarrow Gs.\ G)$;
$gs \in$ *listset Gs*; $x =$ *sum-list gs* $]\!] \Longrightarrow (\bigoplus Gs\leftarrow x) = gs$
**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** $\forall G \in$ *set Gs. AbGroup G add-independentS Gs* $x \in (\bigoplus G\leftarrow Gs.\ G)$
$y \in (\bigoplus G\leftarrow Gs.\ G)$
**shows** $(\bigoplus Gs\leftarrow(x+y)) = [a+b.\ (a,b)\leftarrow zip\ (\bigoplus Gs\leftarrow x)\ (\bigoplus Gs\leftarrow y)]$
**proof**−
**define** *xs ys* **where** $xs = (\bigoplus Gs\leftarrow x)$ **and** $ys = (\bigoplus Gs\leftarrow y)$

**with** *assms*
**have** *xy*: *xs* ∈ *listset Gs x* = *sum-list xs ys* ∈ *listset Gs y* = *sum-list ys*
**using** *AbGroup-inner-dirsum-el-decompI*
**by** *auto*
**from** *assms*(*1*) *xy*(*1*,*3*) **have** [*a*+*b*. (*a*,*b*)←*zip xs ys*] ∈ *listset Gs*
**using** *AbGroup.axioms listset-Group-plus-closed* **by** *fast*
**moreover from** *xy* **have** *x* + *y* = *sum-list* [*a*+*b*. (*a*,*b*)←*zip xs ys*]
**using** *listset-length*[*of xs Gs*] *listset-length*[*of ys Gs*, *THEN sym*] *sum-list-plus*
**by** *simp*
**ultimately show** (⨁ *Gs*←(*x*+*y*)) = [*a*+*b*. (*a*,*b*)←*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* :
⟦ ∀ *G* ∈ *set Gs*. *AbGroup G*; *add-independentS Gs*; *x* ∈ (⨁ *G*←*Gs*. *G*) ⟧
⟹ *length* (⨁ *Gs*←*x*) = *length Gs*
**using** *AbGroup-inner-dirsum-el-decompI listset-length* **by** *fastforce*

**lemma** *AbGroup-inner-dirsum-el-decomp-in-nth* :
**assumes** ∀ *G* ∈ *set Gs*. *AbGroup G add-independentS Gs n* < *length Gs*
*x* ∈ *Gs*!*n*
**shows** (⨁ *Gs*←*x*) = (*replicate* (*length Gs*) *0*)[*n* := *x*]
**proof**−
**from** *assms* **have** *x*: *x* ∈ (⨁ *G*←*Gs*. *G*)
**using** *AbGroup-nth-subset-inner-dirsum* **by** *fast*
**define** *xgs* **where** *xgs* = (*replicate* (*length Gs*) (*0*::'*a*))[*n* := *x*]
**hence** *length xgs* = *length Gs* **by** *simp*
**moreover have** ∀ *k*<*length xgs*. *xgs*!*k* ∈ *Gs*!*k*
**proof**−
**have** ⋀*k*. *k* < *length xgs* ⟹ *xgs*!*k* ∈ *Gs*!*k*
**proof**−
**fix** *k* **assume** *k* < *length xgs*
**with** *assms*(*1*,*4*) *xgs-def* **show** *xgs*!*k* ∈ *Gs*!*k*
**using** *AbGroup.zero-closed*[*of Gs*!*k*] **by** (*cases k* = *n*) *auto*
**qed**
**thus** *?thesis* **by** *fast*
**qed**
**ultimately have** *xgs* ∈ *listset Gs* **using** *listsetI-nth* **by** *fast*
**moreover from** *xgs-def assms*(*3*) **have** *x* = *sum-list xgs*
**using** *sum-list-update*[*of n replicate* (*length Gs*) *0 x*] *nth-replicate sum-list-replicate0*
**by** *simp*
**ultimately show** (⨁ *Gs*←*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* :
⟦ ∀ *G* ∈ *set Gs*. *AbGroup G*; *add-independentS Gs*; *k* < *length Gs*;

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

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

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

**lemma** *AbGroup-inner-dirsum-subset-proj-eq-0* :
**assumes** $Gs \neq []\ \forall\ G \in set\ Gs.\ AbGroup\ G\ add\text{-}independentS\ Gs$
$X \subseteq (\bigoplus G{\leftarrow}Gs.\ G)\ \forall\ i < length\ Gs.\ (\bigoplus Gs{\downarrow}i)\ `\ X = 0$
**shows** $X = 0$
**proof**−
**have** $X \subseteq 0$
**proof**
**fix** $x$ **assume** $x$: $x \in X$
**with** *assms*(*2*−*4*) **have** $x = (\sum i{=}0..{<}\ length\ Gs.\ (\bigoplus Gs{\downarrow}i)\ x)$
**using** *AbGroup-inner-dirsum-el-decompI sum-list-sum-nth*[*of* $(\bigoplus Gs{\leftarrow}x)$]
*AbGroup-length-inner-dirsum-el-decomp*
**by** *fastforce*
**moreover from** *x assms*(*5*) **have** $\forall\ i{<}length\ Gs.\ (\bigoplus Gs{\downarrow}i)\ x = 0$ **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** $\forall\ G \in set\ Gs.\ AbGroup\ G\ add\text{-}independentS\ Gs\ n < length\ Gs$
**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-axioms* $(\bigoplus G{\leftarrow}Gs.\ G)\ \bigoplus Gs{\downarrow}n$
**proof**

**show** *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** $(\bigoplus Gs{\leftarrow}(x+y)) = [x+y.\ (x,y){\leftarrow}zip\ (\bigoplus Gs{\leftarrow}x)\ (\bigoplus Gs{\leftarrow}y)]$
**using** *AbGroup-inner-dirsum-el-decomp-plus* **by** *fast*
**hence** $(\bigoplus Gs{\leftarrow}(x+y)) = map\ (case\text{-}prod\ (+))\ (zip\ (\bigoplus Gs{\leftarrow}x)\ (\bigoplus Gs{\leftarrow}y))$
**using** *concat-map-split-eq-map-split-zip* **by** *simp*
**moreover from** *assms xy*
**have** $n < length\ (\bigoplus Gs{\leftarrow}x)$ $n < length\ (\bigoplus Gs{\leftarrow}y)$
$n < length\ (zip\ (\bigoplus Gs{\leftarrow}x)\ (\bigoplus Gs{\leftarrow}y))$
**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*::*ring-1 set*
+ **assumes** *one-closed* : $1 \in R$
**and** *mult-closed*: $\bigwedge r\ s.\ r \in R \Longrightarrow s \in R \Longrightarrow r * s \in R$
**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* ⇒ *bool* **where** *Subring1 S* ≡ *Ring1 S* ∧ *S* ⊆ *R*

**lemma** *Subring1D1* : *Subring1 S* ⟹ *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* ≡ *aezfun-setspan G*

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

**lemma** *RG-one-closed* : (*1*::(*'r*::*zero-neq-one*,*'g*) *aezfun*) ∈ *group-ring*
**proof**−
**have** *supp* (*aezfun* (*1*::(*'r*,*'g*) *aezfun*)) ⊆ *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*) ∈ *group-ring*
**proof**−
**have** *aezfun* (*0*::(*'r*,*'g*) *aezfun*) = (*0*::*'g*⇒*'r*) **using** *zero-aezfun.rep-eq* **by** *fast*
**hence** *supp* (*aezfun* (*0*::(*'r*,*'g*) *aezfun*)) = *supp* (*0*::*'g*⇒*'r*) **by** *simp*
**moreover have** *supp* (*0*::*'g*⇒*'r*) ⊆ *G* **using** *supp-zerofun-subset-any* **by** *fast*
**ultimately show** *?thesis* **using** *group-ringD* **by** *fast*
**qed**

**lemma** *RG-n0* : *group-ring* ≠ (*0*::(*'r*::*zero-neq-one*, *'g*) *aezfun set*)
**using** *RG-one-closed zero-neq-one* **by** *force*

**lemma** *RG-mult-closed* :
**defines** *RG*: $RG \equiv$ *group-ring* :: ('*r*::*ring-1*, '*g*) *aezfun set*
**shows** $x \in RG \Longrightarrow y \in RG \Longrightarrow x * y \in RG$
**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*: $RG \equiv$ *group-ring* :: ('*r*::*ring-1*, '*g*) *aezfun set*
**shows** *Ring1 RG*
**proof**
**from** *RG* **show** $RG \neq \{\}$ $1 \in RG$ $\bigwedge x\ y.\ x \in RG \Longrightarrow y \in RG \Longrightarrow x * y \in RG$
**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*: $RG \equiv$ *group-ring* :: ('*r*::*ring-1*, '*g*) *aezfun set*
**assumes** $g \in G$
**shows** $r\ \delta\delta\ g \in RG$
**proof**−
**have** *supp*: *supp* (*aezfun* ($r\ \delta\delta\ g$)) = *supp* ($r\ \delta\ g$)
**using** *aezdeltafun arg-cong*[*of* - - *supp*] **by** *fast*
**have** *supp* (*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* : ($r$::'*r*::*ring-1*) $\delta\delta\ 0 \in$ *group-ring*
**using** *zero-closed RG-aezdeltafun-closed*[*of 0*] **by** *fast*

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

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

**lemma** *Subgroup-imp-Subring* :
**fixes** *H* :: *'g set*
**and** *FH* :: (*'r*::*ring-1*,*'g*) *aezfun set*
**and** *FG* :: (*'r*,*'g*) *aezfun set*
**defines** *FH* ≡ *Group.group-ring H*
**and** *FG* ≡ *group-ring*
**shows** *Subgroup H* ⟹ *Ring1.Subring1 FG FH*
**using** *assms Group.Ring1-RG Group.RG-el-decomp-aezdeltafun RG-sum-list-rddg-closed*
**by** *fast*

**end**

**lemma** (**in** *FinGroup*) *group-ring-spanning-set* :
∃ *gs*. *distinct gs* ∧ *set gs* = *G*
∧ (∀ *f*∈ (*group-ring* :: (*'b*::*semiring-1*, *'g*) *aezfun set*). ∃ *bs*.
*length bs* = *length gs* ∧ *f* = (∑ (*b*,*g*)←*zip bs gs*. (*b* *δδ* *0*) ∗ (*1* *δδ* *g*)) )
**using** *finite aezfun-common-supp-spanning-set-decomp*[*of G*] *group-ringD*
**by** *fast*

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

**context** *Group*
**begin**

**abbreviation** *RG-proj* ≡ *aezfun-setspan-proj G*

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

**lemma** *RG-proj-mult-leftdelta'* :
**fixes** *r s* :: *'r*::{*comm-monoid-add*,*mult-zero*}
**shows** *g* ∈ *G* ⟹ *RG-proj* (*r* *δδ* *g* ∗ (*s* *δδ* *g'*)) = *r* *δδ* *g* ∗ *RG-proj* (*s* *δδ* *g'*)
**using** *add-closed add-closed-inverse-right times-aezdeltafun-aezdeltafun*[*of r g*]
*aezfun-setspan-proj-aezdeltafun*[*of G r*∗*s*]
*aezfun-setspan-proj-aezdeltafun*[*of G s*]
**by** *simp*

**lemma** *RG-proj-mult-leftdelta* :
**fixes** *r* :: *'r*::*semiring-1*
**assumes** *g* ∈ *G*
**shows** *RG-proj* ((*r* *δδ* *g*) ∗ *x*) = *r* *δδ* *g* ∗ *RG-proj x*
**proof**−
**from** *aezfun-decomp-aezdeltafun* **obtain** *rgs*
**where** *rgs*: *x* = (∑ (*s*,*h*)←*rgs*. *s* *δδ* *h*)
**using** *RG-el-decomp-aezdeltafun*
**by** *fast*

**hence** *RG-proj* ((*r* *δδ* *g*) ∗ *x*) = (∑ (*s*,*h*)←*rgs*. *RG-proj* ((*r* *δδ* *g*) ∗ (*s* *δδ* *h*)))
**using** *sum-list-const-mult-prod*[*of r δδ g λs h. s δδ h*] *RG-proj-sum-list-prod*
**by** *simp*
**also from** *assms rgs* **have** ... = (*r* *δδ* *g*) ∗ *RG-proj x*
**using** *RG-proj-mult-leftdelta′*[*of g r*]
*sum-list-const-mult-prod*[*of r δδ g λs h. RG-proj* (*s* *δδ* *h*)]
*RG-proj-sum-list-prod*[*of λs h. s δδ h rgs*]
**by** *simp*
**finally show** *?thesis* **by** *fast*
**qed**

**lemma** *RG-proj-mult-rightdelta′* :
**fixes** *r s* :: *′r*::{*comm-monoid-add*,*mult-zero*}
**assumes** *g′* ∈ *G*
**shows** *RG-proj* (*r* *δδ* *g* ∗ (*s* *δδ* *g′*)) = *RG-proj* (*r* *δδ* *g*) ∗ (*s* *δδ* *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-proj-mult-rightdelta* :
**fixes** *r* :: *′r*::*semiring-1*
**assumes** *g* ∈ *G*
**shows** *RG-proj* (*x* ∗ (*r* *δδ* *g*)) = (*RG-proj x*) ∗ (*r* *δδ* *g*)
**proof**−
**from** *aezfun-decomp-aezdeltafun* **obtain** *rgs*
**where** *rgs*: *x* = (∑ (*s*,*h*)←*rgs*. *s* *δδ* *h*)
**using** *RG-el-decomp-aezdeltafun*
**by** *fast*
**hence** *RG-proj* (*x* ∗ (*r* *δδ* *g*)) = (∑ (*s*,*h*)←*rgs*. *RG-proj* ((*s* *δδ* *h*) ∗ (*r* *δδ* *g*)))
**using** *sum-list-mult-const-prod*[*of λs h. s δδ h rgs*] *RG-proj-sum-list-prod*
**by** *simp*
**with** *assms rgs* **show** *?thesis*
**using** *RG-proj-mult-rightdelta′*[*of g - - r*]
*sum-list-prod-cong*[*of*
*rgs λs h. RG-proj* ((*s* *δδ* *h*) ∗ (*r* *δδ* *g*))
*λs h. RG-proj* (*s* *δδ* *h*) ∗ (*r* *δδ* *g*)
]
*sum-list-mult-const-prod*[*of λs h. RG-proj* (*s* *δδ* *h*) *rgs*]
*RG-proj-sum-list-prod*[*of λs h. s δδ h rgs*]
*sum-list-mult-const-prod*[*of λs h. RG-proj* (*s* *δδ* *h*) *rgs r δδ g*]
*RG-proj-sum-list-prod*[*of λs h. s δδ h rgs*]
**by** *simp*
**qed**

**lemma** *RG-proj-mult-right* :
*x* ∈ (*group-ring* :: (*′r*::*ring-1*, *′g*) *aezfun set*)
⟹ *RG-proj* (*y* ∗ *x*) = *RG-proj y* ∗ *x*
**using** *RG-el-decomp-aezdeltafun sum-list-const-mult-prod*[*of y λr g. r δδ 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-proj* $(y * (r\ \delta\delta\ g))$ $\lambda r\ g.$ *RG-proj* $y * (r\ \delta\delta\ g)$
]
*sum-list-const-mult-prod*[*of RG-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* : $[\![\ r \in R;\ m \in M\ ]\!] \Longrightarrow r \cdot m \in M$
**and** *smult-distrib-left* [*simp*] : $[\![\ r \in R;\ m \in M;\ n \in M\ ]\!]$
$\Longrightarrow r \cdot (m + n) = r \cdot m + r \cdot n$
**and** *smult-distrib-right* [*simp*] : $[\![\ r \in R;\ s \in R;\ m \in M\ ]\!]$
$\Longrightarrow (r + s) \cdot m = r \cdot m + s \cdot m$
**and** *smult-assoc* [*simp*] : $[\![\ r \in R;\ s \in R;\ m \in M\ ]\!]$
$\Longrightarrow r \cdot s \cdot m = (r * s) \cdot m$
**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* ⇒ *'m*::*ab-group-add* ⇒ *'m* (**infixr** · *70*)
**assumes** *Ring1 R* ∀ *r*∈*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* ∈ *R s* ∈ *R m* ∈ *Z n* ∈ *Z*
**from** *rsmn*(*1*,*3*) *Z-def assms*(*2*) **show** *r* · *m* ∈ *Z* **by** *simp*
**from** *rsmn*(*1*,*3*,*4*) *Z-def assms*(*2*) **show** *r* · (*m*+*n*) = *r* · *m* + *r* · *n* **by** *simp*
**from** *rsmn*(*1*−*3*) *Z-def assms* **show** (*r* + *s*) · *m* = *r* · *m* + *s* · *m*
**using** *Ring1.add-closed* **by** *auto*
**from** *rsmn*(*1*−*3*) *Z-def assms* **show** *r* · (*s* · *m*) = (*r*∗*s*) · *m*
**using** *Ring1.mult-closed* **by** *auto*
**next**
**define** *Z* **where** *Z* = (*0*::*'m set*)
**fix** *m* **assume** *m* ∈ *Z* **with** *Z-def assms* **show** *1* · *m* = *m*
**using** *Ring1.one-closed* **by** *auto*
**qed**

**context** *RModule*
**begin**

**abbreviation** *RSubmodule* :: *'m set* ⇒ *bool*
**where** *RSubmodule N* ≡ *RModule R smult N* ∧ *N* ⊆ *M*

**lemma** *Group* : *Group M*
**using** *RModule-axioms RModule.axioms*(*2*) **by** *fast*

**lemma** *Subgroup-RSubmodule* : *RSubmodule N* ⟹ *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* ∈ *R* ⟹ *set ms* ⊆ *M* ⟹ *set* (*map* ((·) *r*) *ms*) ⊆ *M*
**using** *smult-closed* **by** (*induct ms*) *auto*

**lemma** *zero-smult* : *m* ∈ *M* ⟹ *0* · *m* = *0*
**using** *R-scalars.zero-closed smult-distrib-right*[*of 0*] *add-left-imp-eq* **by** *simp*

**lemma** *smult-zero* : *r* ∈ *R* ⟹ *r* · *0* = *0*
**using** *zero-closed smult-distrib-left*[*of r 0*] **by** *simp*

**lemma** *neg-smult* : $r \in R \Longrightarrow m \in M \Longrightarrow (-r) \cdot m = -(r \cdot m)$
**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* : $m \in M \Longrightarrow (-1) \cdot m = -m$
**using** *one-closed neg-smult one-smult* **by** *fastforce*

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

**lemma** *smult-distrib-right-diff* :
$[\![ r \in R; s \in R; m \in M ]\!] \Longrightarrow (r - s) \cdot m = r \cdot m - s \cdot m$
**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** *finite A* $\Longrightarrow f$ ' $A \subseteq M \Longrightarrow r \cdot (\sum a{\in}A.\ f\ a) = (\sum a{\in}A.\ r \cdot f\ a)$
**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** *finite A* $\Longrightarrow f$ ' $A \subseteq R \Longrightarrow (\sum a{\in}A.\ f\ a) \cdot m = (\sum a{\in}A.\ (f\ a) \cdot m)$
**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* :
$r \in R \Longrightarrow set\ ms \subseteq M \Longrightarrow r \cdot (sum\text{-}list\ ms) = (\sum m{\leftarrow}ms.\ r \cdot m)$
**using** *smult-zero sum-list-closed*[*of id*] **by** (*induct ms*) *auto*

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

```
      ⟹ (∑ (x,y)←xys. f x y) · m = (∑ (x,y)←xys. f x y · m)
  using zero-smult R-scalars.sum-list-closed-prod[of f]
  by    (induct xys) auto

lemma RSubmoduleI :
  assumes Subgroup N ⋀r n. r ∈ R ⟹ n ∈ N ⟹ r · n ∈ N
  shows   RSubmodule N
proof
  show RModule R smult N
  proof (intro-locales, rule SubgroupD1[OF assms(1)], unfold-locales)
    from assms(2) show ⋀r m. r ∈ R ⟹ m ∈ N ⟹ r · m ∈ N by fast
    from assms(1)
      show  ⋀r m n. ⟦ r ∈ R; m ∈ N; n ∈ N ⟧ ⟹ r · (m + n) = r · m + r · n
      using smult-distrib-left
      by    blast
    from assms(1)
      show  ⋀r s m. ⟦ r ∈ R; s ∈ R; m ∈ N ⟧ ⟹ (r + s) · m = r · m + s · m
      using smult-distrib-right
      by    blast
    from assms(1)
      show  ⋀r s m. ⟦ r ∈ R; s ∈ R; m ∈ N ⟧ ⟹ r · s · m = (r * s) · m
      using smult-assoc
      by    blast
    from assms(1) show ⋀m. m ∈ N ⟹ 1 · m = m using one-smult by blast
  qed
  from assms(1) show N ⊆ M by fast
qed

end

lemma (in R-scalar-mult) listset-RModule-Rsmult-closed :
  ⟦ ∀ M∈set Ms. RModule R smult M; r ∈ R; ms ∈ listset Ms ⟧
      ⟹ [r · m. m←ms] ∈ listset Ms
proof−
  have ⟦ length ms = length Ms; ms ∈ listset Ms;
           ∀ M∈set Ms. RModule R smult M; r ∈ R ⟧
             ⟹ [r · m. m←ms] ∈ 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 · m M]
      by    fastforce
  qed simp
  thus ⟦ ∀ M∈set Ms. RModule R smult M; r ∈ R; ms ∈ listset Ms ⟧
           ⟹ [r · m. m←ms] ∈ listset Ms
    using listset-length[of ms Ms] by fast
qed

context Module
begin
```

**abbreviation** *Submodule* :: *'m set* ⇒ *bool*
  **where** *Submodule* ≡ *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* :
  (⋀*r*. *r* ∈ *R* ⟹ *r* · *0* = *0*) ⟹ *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* ⊆ *M*
    **using** *Group Group.set-plus-closed* **by** *auto*
  **ultimately show** *Subgroup* (*L* + *N*) **by** *fast*
**next**
  **fix** *r x* **assume** *rx*: *r* ∈ *R x* ∈ *L* + *N*
  **from** *rx*(*2*) **obtain** *m n* **where** *mn*: *m* ∈ *L n* ∈ *N x* = *m* + *n*
    **using** *set-plus-def*[*of L N*] **by** *fast*
  **with** *assms rx*(*1*) **show** *r* · *x* ∈ *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* :
  (∀ *N*∈*set Ns*. *RSubmodule N*) ⟹ *RSubmodule* (∑ *N*←*Ns*. *N*)
  **using** *trivial-RSubmodule RSubmodule-set-plus*
  **by** (*induct Ns*) *auto*

**lemma** *RSubmodule-inner-dirsum* :
  **assumes** (∀ *N*∈*set Ns*. *RSubmodule N*)
  **shows** *RSubmodule* (⨁ *N*←*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* :
$(\forall N{\in}set\ Ns.\ RSubmodule\ N) \Longrightarrow RModule\ R\ smult\ (\bigoplus N{\leftarrow}Ns.\ N)$
**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**
$\bigwedge r\ m.\ r \in S \Longrightarrow m \in M \Longrightarrow r \cdot m \in M$
$\bigwedge r\ m\ n.\ r \in S \Longrightarrow m \in M \Longrightarrow n \in M \Longrightarrow r \cdot (m + n) = r \cdot m + r \cdot n$
$\bigwedge m.\ m \in M \Longrightarrow 1 \cdot m = m$
**using** *smult-closed smult-distrib-left*
**by** *auto*
**from** *assms*
**show** $\bigwedge r\ s\ m.\ r \in S \Longrightarrow s \in S \Longrightarrow m \in M \Longrightarrow (r + s) \cdot m = r \cdot m + s \cdot m$
**using** *Ring1.add-closed smult-distrib-right*
**by** *fast*
**from** *assms*
**show** $\bigwedge r\ s\ m.\ r \in S \Longrightarrow s \in S \Longrightarrow m \in M \Longrightarrow r \cdot s \cdot m = (r * s) \cdot m$
**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* ⇒ *'m list* ⇒ *'m* (**infix** ·· *70*)
**where** $rs \cdot\cdot ms = (\sum (r,m){\leftarrow}zip\ rs\ ms.\ r \cdot m)$

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

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

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

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

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

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

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

**lemma** *lincomb-conv-take-right* : $rs \cdot\cdot ms = rs \cdot\cdot take\ (length\ rs)\ ms$
**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* : $set\ rs \subseteq R \Longrightarrow set\ ms \subseteq M \Longrightarrow rs \cdot\cdot ms \in M$
**proof** (*induct ms arbitrary*: *rs*)
**case** *Nil* **show** *?case* **using** *lincomb-Nil zero-closed* **by** *simp*
**next**
**case** (*Cons m ms*)
**hence** *Cons1*: $\bigwedge rs.\ set\ rs \subseteq R \Longrightarrow rs \cdot\cdot ms \in M\ m \in M\ set\ rs \subseteq R$ **by** *auto*
**show** $rs \cdot\cdot (m\#ms) \in M$
**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* :
$[\![\ set\ rs \subseteq R;\ s \in R;\ set\ ms \subseteq M\ ]\!] \Longrightarrow s \cdot (rs \cdot\cdot ms) = [s*r.\ r \leftarrow rs] \cdot\cdot ms$
**using** *lincomb-Nil smult-zero lincomb-Cons smult-closed lincomb-closed*
**by** (*induct rs ms rule*: *list-induct2'*) *auto*

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

*smult-lincomb*[*of rs − 1*] *map-times-neg1-eq-map-uminus*
**by** *auto*

**lemma** *lincomb-sum-left* :
⟦ *set rs* ⊆ *R*; *set ss* ⊆ *R*; *set ms* ⊆ *M*; *length rs* ≤ *length ss* ⟧
⟹ [*r* + *s*. (*r*,*s*)←*zip rs ss*] ·· *ms* = *rs* ·· *ms* + (*take* (*length rs*) *ss*) ·· *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** ⋀*ms*. *length* (*r*#*rs*) ≤ *length* []
⟹ [*a* + *b*. (*a*,*b*)←*zip* (*r*#*rs*) []] ·· *ms*
= (*r*#*rs*) ·· *ms* + (*take* (*length* (*r*#*rs*)) []) ·· *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*)←*zip* (*r*#*rs*) (*s*#*ss*)] ·· *ms*
= (*r*#*rs*) ·· *ms* + (*take* (*length* (*r*#*rs*)) (*s*#*ss*)) ·· *ms*
**using** *lincomb-Nil lincomb-Cons* **by** (*cases ms*) *auto*
**qed**

**lemma** *lincomb-sum* :
**assumes** *set rs* ⊆ *R set ss* ⊆ *R set ms* ⊆ *M length rs* ≤ *length ss*
**shows** *rs* ·· *ms* + *ss* ·· *ms*
= ([*a* + *b*. (*a*,*b*)←*zip rs ss*] @ (*drop* (*length rs*) *ss*)) ·· *ms*
**proof**−
**define** *zs fss bss*
**where** *zs* = [*a* + *b*. (*a*,*b*)←*zip rs ss*]
**and** *fss* = *take* (*length rs*) *ss*
**and** *bss* = *drop* (*length rs*) *ss*
**from** *assms*(*4*) *zs-def fss-def* **have** *length zs* = *length rs length fss* = *length rs*
**using** *length-concat-map-split-zip*[*of* λ*a b*. *a* + *b rs*] **by** *auto*
**hence** (*zs* @ *bss*) ·· *ms* = *rs* ·· *ms* + (*fss* @ *bss*) ·· *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* :
⟦ *set rs* ⊆ *R*; *set ss* ⊆ *R*; *set ms* ⊆ *M*; *length rs* ≤ *length ss* ⟧
⟹ [*r* − *s*. (*r*,*s*)←*zip rs ss*] ·· *ms* = *rs* ·· *ms* − (*take* (*length rs*) *ss*) ·· *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** ⋀*ms*. *length* (*r*#*rs*) ≤ *length* []
⟹ [*a* − *b*. (*a*,*b*)←*zip* (*r*#*rs*) []] ·· *ms*

```
            = (r#rs) ·· ms − (take (length (r#rs)) []) ·· 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)←zip (r#rs) (s#ss)] ·· ms
            = (r#rs) ·· ms − (take (length (r#rs)) (s#ss)) ·· ms
    using lincomb-Nil lincomb-Cons smult-distrib-right-diff by (cases ms) auto
qed

lemma lincomb-replicate-left :
  r ∈ R ⟹ set ms ⊆ M ⟹ (replicate k r) ·· ms = r · ( ∑ m←(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 ⊆ M ⟹ (replicate k 0) ·· ms = 0
proof−
  assume ms: set ms ⊆ M
  hence (replicate k 0) ·· ms = 0 · (∑ m←(take k ms). m)
    using R-scalars.zero-closed lincomb-replicate-left by fast
  moreover from ms have (∑ m←(take k ms). m) ∈ M
    using set-take-subset sum-list-closed by fastforce
  ultimately show (replicate k 0) ·· ms = 0 using zero-smult by simp
qed

lemma lincomb-0coeffs : set ms ⊆ M ⟹ ∀ s∈set rs. s = 0 ⟹ rs ·· ms = 0
  using lincomb-Nil lincomb-Cons zero-smult
  by    (induct rs ms rule: list-induct2') auto

lemma delta-scalars-lincomb-eq-nth :
  set ms ⊆ M ⟹ n < length ms
       ⟹ ((replicate (length ms) 0)[n := 1]) ·· 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* $\Longrightarrow$ *set ms* $\subseteq$ *M*
      $\Longrightarrow$ $\exists$ *ss. set ss* $\subseteq$ *R* $\wedge$ *length ss = length ms*
        $\wedge$ *take* (*length rs*) *ss = take* (*length ms*) *rs* $\wedge$ *rs* $\cdot\cdot$ *ms = ss* $\cdot\cdot$ *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*) [] $\cdot\cdot$ (*m*#*ms*) = *ss* $\cdot\cdot$ (*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* $\cdot\cdot$ *ms = ss* $\cdot\cdot$ *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*) $\cdot\cdot$ (*m*#*ms*) = (*r*#*ss*) $\cdot\cdot$ (*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*
      $\Longrightarrow$ (*concat rss*) $\cdot\cdot$ (*concat mss*) = ($\sum$ (*rs*,*ms*)$\leftarrow$*zip rss mss. rs* $\cdot\cdot$ *ms*)
  **using** *lincomb-Nil lincomb-append* **by** (*induct rss mss rule*: *list-induct2'*) *auto*

**lemma** *lincomb-snoc0* : *set ms* $\subseteq$ *M* $\Longrightarrow$ (*as*@[*0*]) $\cdot\cdot$ *ms = as* $\cdot\cdot$ *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*) $\cdot\cdot$ *ms = as* $\cdot\cdot$ *ms*
**proof** (*induct as rule*: *rev-induct*)
  **case** (*snoc a as*)
  **hence** *caseassm*: *strip-while* ((=) *0*) *as* $\cdot\cdot$ *ms = as* $\cdot\cdot$ *ms* **by** *fast*
  **show** *?case*
  **proof** (*cases a = 0*)

**case** *True*
**moreover with** *assms* **have** $(as@[a]) \cdot\cdot ms = as \cdot\cdot ms$
**using** *lincomb-snoc0* **by** *fast*
**ultimately show** *strip-while* $((=)\ 0)\ (as\ @\ [a]) \cdot\cdot ms = (as@[a]) \cdot\cdot ms$
**using** *caseassm* **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* ⇒ *'m set*
**where** *RSpan* [] = *0*
| *RSpan* (*m*#*ms*) = { *r* · *m* | *r*. *r* ∈ *R* } + *RSpan ms*

**lemma** *RSpan-single* : *RSpan* [*m*] = { *r* · *m* | *r*. *r* ∈ *R* }
**using** *add-0-right*[*of* { *r* · *m* | *r*. *r* ∈ *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* ∈ *RSpan ms* ⟹ ∃ *rs*. *set rs* ⊆ *R* ∧ *length rs* = *length ms* ∧ *n* = *rs* ·· *ms*
**proof** (*induct ms arbitrary*: *n*)
**case** *Nil*
**hence** *n* = *0* **by** *simp*
**thus** ∃ *rs*. *set rs* ⊆ *R* ∧ *length rs* = *length* [] ∧ *n* = *rs* ·· []
**using** *lincomb-Nil* **by** *simp*
**next**
**case** (*Cons m ms*)
**from** *this* **obtain** *r rs*
**where** *set* (*r*#*rs*) ⊆ *R length* (*r*#*rs*) = *length* (*m*#*ms*) *n* = (*r*#*rs*) ·· (*m*#*ms*)
**using** *set-plus-def*[*of* - *RSpan ms*] *lincomb-Cons*
**by** *fastforce*
**thus** ∃ *rs*. *set rs* ⊆ *R* ∧ *length rs* = *length* (*m*#*ms*) ∧ *n* = *rs* ·· (*m*#*ms*) **by** *fast*
**qed**

**lemma** *in-RSpan-Cons-obtain-same-length-coeffs* :
*n* ∈ *RSpan* (*m*#*ms*) ⟹ ∃ *r rs*. *set* (*r*#*rs*) ⊆ *R* ∧ *length rs* = *length ms*
∧ *n* = *r* · *m* + *rs* ·· *ms*
**proof**−
**assume** *n* ∈ *RSpan* (*m*#*ms*)

```
  from this obtain x y where x ∈ RSpan [m] y ∈ RSpan ms n = x + y
    using RSpan-Cons set-plus-def[of RSpan [m]] by auto
  thus ∃ r rs. set (r # rs) ⊆ R ∧ length rs = length ms ∧ n = r · m + rs ·· ms
    using RSpan-single in-RSpan-obtain-same-length-coeffs[of y ms] by auto
qed

lemma RSpanD-lincomb :
  RSpan ms = { rs ·· ms | rs. set rs ⊆ R ∧ length rs = length ms}
proof
  show RSpan ms ⊆ {rs ·· ms |rs. set rs ⊆ R ∧ length rs = length ms}
    using in-RSpan-obtain-same-length-coeffs by fast
  show {rs ·· ms |rs. set rs ⊆ R ∧ length rs = length ms} ⊆ RSpan ms
  proof
    fix x assume x ∈ {rs ·· ms |rs. set rs ⊆ R  ∧ length rs = length ms}
    from this obtain rs where rs: set rs ⊆ R length rs = length ms x = rs ·· ms
      by fast
    from rs(2) have set rs ⊆ R ⟹ rs ·· ms ∈ RSpan ms
      using lincomb-Nil lincomb-Cons by (induct rs ms rule: list-induct2) auto
    with rs(1,3) show x ∈ 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 ≡ R-scalar-mult.RSpan UNIV smult

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

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

end

context RModule
```

**begin**

**lemma** *RSpan-contains-spanset-single* : $m \in M \Longrightarrow m \in RSpan\ [m]$
**using** *one-closed RSpan-single* **by** *fastforce*

**lemma** *RSpan-single-nonzero* : $m \in M \Longrightarrow m \neq 0 \Longrightarrow RSpan\ [m] \neq 0$
**using** *RSpan-contains-spanset-single* **by** *auto*

**lemma** *Group-RSpan-single* :
**assumes** $m \in M$
**shows** *Group* (*RSpan* $[m]$)
**proof**
**from** *assms* **show** $RSpan\ [m] \neq \{\}$ **using** *RSpan-contains-spanset-single* **by** *fast*
**next**
**fix** $x\ y$ **assume** $x \in RSpan\ [m]\ y \in RSpan\ [m]$
**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** $x - y \in RSpan\ [m]$
**using** *R-scalars.diff-closed*[*of r s*] *RSpan-single*[*of m*] **by** *auto*
**qed**

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

```
    qed
  qed
qed

lemma RSpanI-lincomb-arb-len-coeffs :
  set rs ⊆ R ⟹ set ms ⊆ M ⟹ rs ·· ms ∈ RSpan ms
  using RSpanD-lincomb-arb-len-coeffs by fast

lemma RSpan-contains-RSpans-Cons-left :
  set ms ⊆ M ⟹ RSpan [m] ⊆ RSpan (m#ms)
  using RSpan-Cons Group-RSpan AbGroup.intro AbGroup.subset-plus-left by fast

lemma RSpan-contains-RSpans-Cons-right :
  m ∈ M ⟹ RSpan ms ⊆ 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 ⊆ M ⟹ RSpan ms ⊆ RSpan (ms@ns)
  using RSpan-append Group-RSpan AbGroup.intro AbGroup.subset-plus-left
  by    fast

lemma RSpan-contains-spanset : set ms ⊆ M ⟹ set ms ⊆ 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 ⊆ M ⟹ set ns ⊆ M ⟹ set ms ⊆ RSpan (ms@ns)
  using RSpan-contains-spanset[of ms@ns] by simp

lemma RSpan-contains-spanset-append-right :
  set ms ⊆ M ⟹ set ns ⊆ M ⟹ set ns ⊆ RSpan (ms@ns)
  using RSpan-contains-spanset[of ms@ns] by simp

lemma RSpan-zero-closed : set ms ⊆ M ⟹ 0 ∈ RSpan ms
  using Group-RSpan Group.zero-closed by fast

lemma RSpan-single-closed : m ∈ M ⟹ RSpan [m] ⊆ M
  using RSpan-single smult-closed by auto

lemma RSpan-closed : set ms ⊆ M ⟹ RSpan ms ⊆ 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** $r \in R$ *set ms* $\subseteq M$ $n \in$ *RSpan ms*
**shows** $r \cdot n \in$ *RSpan ms*
**proof**−
**from** *assms*(*2*,*3*) **obtain** *rs* **where** *rs*: *set rs* $\subseteq R$ $n = rs \cdot\cdot ms$
**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* $\subseteq M$ $n \in$ *RSpan ms* $n' \in$ *RSpan ms*
**shows** $n + n' \in$ *RSpan ms*
**proof**−
**from** *assms* (*2*,*3*) **obtain** *rs ss*
**where** *rs*: *set rs* $\subseteq R$ *length rs* = *length ms* $n = rs \cdot\cdot ms$
**and** *ss*: *set ss* $\subseteq R$ *length ss* = *length ms* $n' = ss \cdot\cdot ms$
**using** *RSpanD-lincomb* **by** *auto*
**with** *assms*(*1*) **have** $n + n' = [r + s.\ (r,s) \leftarrow zip\ rs\ ss] \cdot\cdot ms$
**using** *lincomb-sum-left* **by** *simp*
**moreover from** *rs*(*1*) *ss*(*1*) **have** *set* $[r + s.\ (r,s) \leftarrow zip\ rs\ ss] \subseteq R$
**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** $n + n' \in$ *RSpan ms*
**using** *assms*(*1*) *RSpanI-lincomb-arb-len-coeffs* **by** *simp*
**qed**

**lemma** *RSpan-lincomb-closed* :
⟦ *set rs* $\subseteq R$; *set ms* $\subseteq M$; *set ns* $\subseteq$ *RSpan ms* ⟧ $\Longrightarrow$ $rs \cdot\cdot ns \in$ *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* $\subseteq M \Longrightarrow M \subseteq$ *RSpan ms* $\Longrightarrow M =$ *RSpan ms*
**using** *RSpan-closed* **by** *fast*

**lemma** *RSpan-contains-RSpan-take* :
*set ms* $\subseteq M \Longrightarrow$ *RSpan* (*take k ms*) $\subseteq$ *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** $s$: $s \in R$ $n = s \cdot m$ **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* $\subseteq M \Longrightarrow$ *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* $\subseteq M \Longrightarrow$ *set ns* $\subseteq$ *RSpan ms* $\Longrightarrow$ *RSpan ns* $\subseteq$ *RSpan ms*
**using** *RSpanD-lincomb*[*of ns*] *RSpan-lincomb-closed* **by** *auto*

**lemma** *spanset-reduce-Cons* :
*set ms* $\subseteq M \Longrightarrow m \in$ *RSpan ms* $\Longrightarrow$ *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-rs*: $r \in R$ *set rs* $\subseteq R$ *length rs* = *length ms* $x = r \cdot m +$ *rs* ·· *ms*
**and** *s-ss*: $s \in R$ *set ss* $\subseteq R$ *length ss* = *length ms* $m = s \cdot n +$ *ss* ·· *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-rs*(*1*) *s-ss*(*2*) **have** *set1*: *set* [$r$∗$a$. $a$←*ss*] $\subseteq R$ **using** *mult-closed* **by** *auto*

**have** $x = ((r * s) \# [a + b.\ (a,b) \leftarrow zip\ [r*a.\ a \leftarrow ss]\ rs]) \cdot\cdot (n \# ms)$
**proof** −
**from** *r-rs*(2,3) *s-ss*(3) *assms*(2)
**have** $[r*a.\ a \leftarrow ss] \cdot\cdot ms + rs \cdot\cdot ms$
$= [a + b.\ (a,b) \leftarrow zip\ [r*a.\ a \leftarrow ss]\ rs] \cdot\cdot ms$
**using** *set1 lincomb-sum*
**by** *simp*
**moreover from** *assms*(1,2) *r-rs*(1,2,4) *s-ss*(1,2,4)
**have** $x = (r * s) \cdot n + ([r*a.\ a \leftarrow ss] \cdot\cdot ms + rs \cdot\cdot ms)$
**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* $((r * s) \# [a + b.\ (a,b) \leftarrow zip\ [r*a.\ a \leftarrow ss]\ rs]) \subseteq R$
**proof** −
**from** *r-rs*(2) **have** *set* $[a + b.\ (a,b) \leftarrow zip\ [r*a.\ a \leftarrow ss]\ rs] \subseteq R$
**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** $x \in RSpan\ (n \# ms)$
**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-fingen M* ≡ (∃ *ms. set ms* ⊆ *M* ∧ *RSpan ms* = *M*)

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

**definition** *dim-R* :: ′*m set* ⇒ *nat*
  **where** *dim-R M* = (*if R-fingen M then* (
        *LEAST n.* ∃ *ms. length ms* = *n* ∧ *set ms* ⊆ *M* ∧ *RSpan ms* = *M*
      ) *else 0*)

**lemma** *dim-R-nonzero* :
  **assumes** *dim-R M* > *0*
  **shows** *M* ≠ *0*
**proof**
  **assume** *M*: *M* = *0*
  **hence** *dim-R M*
      = (*LEAST n.* ∃ *ms. length ms* = *n* ∧ *set ms* ⊆ *M* ∧ *RSpan ms* = *M*)
    **using** *dim-R-def* **by** *simp*
  **moreover from** *M* **have** *length* [] = *0* ∧ *set* [] ⊆ *M* ∧ *RSpan* [] = *M* **by** *simp*
  **ultimately show** *False* **using** *assms* **by** *simp*
**qed**

**end**

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

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

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

### 3.2.4 *R*-linear independence

**context** *R-scalar-mult*
**begin**

**primrec** *R-lin-independent* :: ′*m list* ⇒ *bool* **where**
  *R-lin-independent-Nil*: *R-lin-independent* [] = *True* |
  *R-lin-independent-Cons*:
  *R-lin-independent* (*m*#*ms*) = (*R-lin-independent ms*
      ∧ (∀ *r rs.* (*set* (*r*#*rs*) ⊆ *R* ∧ (*r*#*rs*) ·· (*m*#*ms*) = *0*) ⟶ *r* = *0*))

**lemma** *R-lin-independent-ConsI* :
  **assumes** *R-lin-independent ms*
      ⋀*r rs. set* (*r*#*rs*) ⊆ *R* ⟹ (*r*#*rs*) ·· (*m*#*ms*) = *0* ⟹ *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)* $\Longrightarrow$ *R-lin-independent ms*
**by** *simp*

**lemma** *R-lin-independent-ConsD2* :
⟦ *R-lin-independent (m#ms)*; *set (r#rs)* $\subseteq$ *R*; *(r#rs)* $\cdot\cdot$ *(m#ms)* = *0* ⟧
$\Longrightarrow$ *r* = *0*
**by** *auto*

**end**

**context** *RModule*
**begin**

**lemma** *R-lin-independent-imp-same-scalars* :
⟦ *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* $\cdot\cdot$ *ms* = *ss* $\cdot\cdot$ *ms* ⟧ $\Longrightarrow$ *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*)←*zip* (*r#rs*) (*s#ss*)]) $\subseteq$ *R*
**using** *Ring1 Ring1.list-diff-closed* **by** *fast*
**hence** *set* ((*r*−*s*)#[*a*−*b*. (*a*,*b*)←*zip rs ss*]) $\subseteq$ *R* **by** *simp*
**moreover from** *Cons(1,4−6,8) ms*
**have** *1*: ((*r*−*s*)#[*a*−*b*. (*a*,*b*)←*zip rs ss*]) $\cdot\cdot$ (*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* $\cdot\cdot$ *ns* = *ss* $\cdot\cdot$ *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* :
⟦ *set ms* $\subseteq$ *M*; *R-lin-independent ms*; *n* $\in$ *RSpan ms* ⟧
$\Longrightarrow$ (∃! *rs*. *set rs* $\subseteq$ *R* ∧ *length rs* = *length ms* ∧ *n* = *rs* $\cdot\cdot$ *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 ⊆ M ⟹
       (∀ rs. set rs ⊆ R ∧ length rs = length ms ∧ rs ·· ms = 0 ⟶ set rs ⊆ 0)
         ⟹ R-lin-independent ms
proof (induct ms)
  case (Cons m ms) show ?case
  proof (rule R-lin-independent-ConsI)
    have ⋀rs. ⟦ set rs ⊆ R; length rs = length ms; rs ·· ms = 0 ⟧ ⟹ set rs ⊆ 0
    proof−
      fix rs assume rs: set rs ⊆ R length rs = length ms rs ·· ms = 0
      with Cons(2) have set (0#rs) ⊆ R length (0#rs)
                          = length (m#ms) (0#rs) ·· (m#ms) = 0
        using R-scalars.zero-closed lincomb-Cons zero-smult by auto
      with Cons(3) have set (0#rs) ⊆ 0 by fast
      thus set rs ⊆ 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) ·· (m#ms) = 0 using lincomb-Cons by simp
    moreover from r-rs(1) ss(1) have set (r#ss) ⊆ R by simp
    moreover from ss(2) have length (r#ss) = length (m#ms) by simp
    ultimately have set (r#ss) ⊆ 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) ⊆ M
          ⋀rss. set (concat rss) ⊆ R ⟹ list-all2 eq-len rss mss
              ⟹ (concat rss) ·· (concat mss) = 0 ⟹ (∀ rs ∈ set rss. set rs ⊆ 0)
  shows   R-lin-independent (concat mss)
  using   assms(2)
proof (rule R-lin-independentI-all-scalars)
  have ⋀rs. ⟦ set rs ⊆ R; length rs = length (concat mss); rs ·· concat mss = 0 ⟧
            ⟹ set rs ⊆ 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-all2 eq-len rss mss
      using match-concat by fast
    with rs(1,3) assms(3) show set rs ⊆ 0 by auto
  qed
  thus ∀ rs. set rs ⊆ R ∧ length rs = length (concat mss) ∧ rs ·· concat mss = 0
            ⟶ set rs ⊆ 0
```

**by** *auto*
**qed**

**lemma** *R-lin-independentD-all-scalars* :
⟦ *set rs* ⊆ *R*; *set ms* ⊆ *M*; *length rs* ≤ *length ms*; *R-lin-independent ms*;
*rs* ·· *ms* = *0* ⟧ ⟹ *set rs* ⊆ *0*
**proof** (*induct rs ms rule*: *list-induct2′*)
**case** (*4 r rs m ms*)
**from** *4*(*2*,*5*,*6*) **have** *r* = *0* **by** *auto*
**moreover with** *4* **have** *set rs* ⊆ *0* **using** *lincomb-Cons zero-smult* **by** *simp*
**ultimately show** *?case* **by** *simp*
**qed** *auto*

**lemma** *R-lin-independentD-all-scalars-nth* :
**assumes** *set rs* ⊆ *R set ms* ⊆ *M R-lin-independent ms rs* ·· *ms* = *0*
*k* < *min* (*length rs*) (*length ms*)
**shows** *rs*!*k* = *0*
**proof**−
**from** *assms*(*1*,*2*) **obtain** *ss*
**where** *ss*: *set ss* ⊆ *R length ss* = *length ms*
*take* (*length rs*) *ss* = *take* (*length ms*) *rs rs* ·· *ms* = *ss* ·· *ms*
**using** *lincomb-obtain-same-length-Rcoeffs*[*of rs ms*]
**by** *fast*
**from** *ss*(*1*,*2*,*4*) *assms*(*2*,*3*,*4*) **have** *set ss* ⊆ *0*
**using** *R-lin-independentD-all-scalars* **by** *auto*
**moreover from** *assms*(*5*) *ss*(*3*) **have** *rs*!*k* = (*take* (*length rs*) *ss*)!*k* **by** *simp*
**moreover from** *assms*(*5*) *ss*(*2*) **have** *k* < *length* (*take* (*length rs*) *ss*) **by** *simp*
**ultimately show** *?thesis* **using** *in-set-conv-nth* **by** *force*
**qed**

**lemma** *R-lin-dependent-dependence-relation* :
*set ms* ⊆ *M* ⟹ ¬ *R-lin-independent ms*
⟹ ∃ *rs*. *set rs* ⊆ *R* ∧ *set rs* ≠ *0* ∧ *length rs* = *length ms* ∧ *rs* ·· *ms* = *0*
**proof** (*induct ms*)
**case** (*Cons m ms*) **show** *?case*
**proof** (*cases R-lin-independent ms*)
**case** *True*
**with** *Cons*(*3*)
**have** ¬ (∀ *r rs*. (*set* (*r*#*rs*) ⊆ *R* ∧ (*r*#*rs*) ·· (*m*#*ms*) = *0*) ⟶ *r* = *0*)
**by** *simp*
**from** *this* **obtain** *r rs*
**where** *r-rs*: *set* (*r*#*rs*) ⊆ *R* (*r*#*rs*) ·· (*m*#*ms*) = *0 r* ≠ *0*
**by** *fast*
**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*
**from** *ss r-rs* **have** *set* (*r*#*ss*) ⊆ *R* ∧ *set* (*r*#*ss*) ≠ *0*
∧ *length* (*r*#*ss*) = *length* (*m*#*ms*) ∧ (*r*#*ss*) ·· (*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 ⊆ R set rs ≠ 0 length rs = length ms rs ·· ms = 0
   by   fastforce
  from False rs Cons(2)
   have  set (0#rs) ⊆ R ∧ set (0#rs) ≠ 0 ∧ length (0#rs) = length (m#ms)
           ∧ (0#rs) ·· (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 ⊆ M ⟹ R-lin-independent ms ⟹ distinct ms
proof (induct ms)
 case (Cons m ms)
 have ⋀n. n ∈ set ms ⟹ m ≠ n
 proof
  fix n assume n: n ∈ 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]) ·· (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 ⊆ M ⟹ R-lin-independent ms ⟹ 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) ⊆ R (r#rs)··(m # take k ms) = 0
```

```
    from r-rs(1) Cons(2) obtain ss
      where ss: set ss ⊆ R length ss = length (take k ms)
              rs ·· take k ms = ss ·· take k ms
      using set-take-subset[of k ms] lincomb-obtain-same-length-Rcoeffs
      by    force
    from r-rs(1) ss(1) have set (r#ss) ⊆ R by simp
    moreover from r-rs(2) ss have (r#ss) ·· (m#ms) = 0
      using lincomb-Cons lincomb-Nil
            lincomb-append-right[of ss take k ms drop k ms]
      by    simp
    ultimately show r = 0 using Cons(3) by auto
  qed
  ultimately show ?thesis by simp
 qed simp
qed simp

lemma R-lin-independent-Cons-imp-independent-RSpans :
 assumes m ∈ M R-lin-independent (m#ms)
 shows   add-independentS [RSpan [m], RSpan ms]
proof (rule add-independentS-doubleI)
 fix x y assume xy: x ∈ RSpan [m] y ∈ RSpan ms x + y = 0
 from xy(1,2) obtain r rs where r-rs: r ∈ R x = r · m set rs ⊆ R y = rs ·· ms
   using RSpan-single RSpanD-lincomb by fast
 with xy(3) have set (r#rs) ⊆ R (r#rs) ·· (m#ms) = 0
   using lincomb-Cons by auto
 with assms r-rs(2) show x = 0 using zero-smult by auto
qed

lemma hd0-imp-R-lin-dependent : ¬ R-lin-independent (0#ms)
 using lincomb-Cons[of 1 [] 0 ms] lincomb-Nil[of [] ms] smult-zero one-closed
       R-lin-independent-Cons
 by    fastforce

lemma R-lin-independent-imp-hd-n0 : R-lin-independent (m#ms) ⟹ m ≠ 0
 using hd0-imp-R-lin-dependent by fast

lemma R-lin-independent-imp-hd-independent-from-RSpan :
 assumes m ∈ M set ms ⊆ M R-lin-independent (m#ms)
 shows   m ∉ RSpan ms
proof
 assume m: m ∈ RSpan ms
 with assms(2) have (−1) · m ∈ RSpan ms
   using RSubmodule-RSpan[of ms]
         RModule.smult-closed[of R smult RSpan ms −1 m]
         one-closed R-scalars.neg-closed[of 1]
   by    simp
 moreover from assms(1) have m + (−1) · 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 ∈ M
 shows set ms ⊆ M ⟹ R-lin-independent (ms @ n # ns)
            ⟹ R-lin-independent (ms @ ns)
proof (induct ms)
 case (Cons m ms)
 moreover have ⋀r rs. set (r#rs) ⊆ R ⟹ (r#rs) ·· (m#ms@ns) = 0
                  ⟹ r = 0
 proof−
  fix r rs assume r-rs: set (r#rs) ⊆ R (r#rs) ·· (m # ms @ ns) = 0
  from Cons(2) r-rs(1) 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
  from assms ss(2,3) r-rs(2)
   have (r # ss @ 0 # drop (length ms) rs) ·· (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) ⊆ 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-lin-independent ( (m#ms) @ ns) by auto
qed simp

lemma R-lin-independent-vs-lincomb0 :
 assumes set (ms@n#ns) ⊆ M R-lin-independent (ms @ n # ns)
        set (rs@s#ss) ⊆ R length rs = length ms
        (rs@s#ss) ·· (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* ⊆ *M* ⟹ *R-lin-independent* (*ms*@*ns*)
⟹ *add-independentS* [*RSpan ms*, *RSpan ns*]
**proof** (*induct ms*)
**case** (*Cons m ms*)
**show** *?case*
**proof** (*rule add-independentS-doubleI*)
**fix** *x y* **assume** *xy*: *y* ∈ *RSpan nsx* ∈ *RSpan* (*m*#*ms*) *x* + *y* = *0*
**from** *xy*(*2*) **obtain** *x1 x2*
**where** *x1-x2*: *x1* ∈ *RSpan* [*m*] *x2* ∈ *RSpan 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*)) ⊆ *R length rs* = *length ms x1* = *r* · *m*
*x2* = *rs* ·· *ms y* = *ss* ·· *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*)) ·· (*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*
≡ *R-scalar-mult.R-lin-independent UNIV smult ms*

**lemmas** *lin-independent-ConsI*

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

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

**lemma** *R-finrankI* :
$(\bigwedge ms.\ set\ ms \subseteq M \Longrightarrow R\text{-}lin\text{-}independent\ ms \Longrightarrow length\ ms \leq n)$
$\Longrightarrow R\text{-}finrank\ M$
**unfolding** *R-finrank-def* **by** *blast*

**lemma** *R-finrankD* :
$R\text{-}finrank\ M \Longrightarrow \exists\, n.\ \forall\, ms.\ set\ ms \subseteq M \wedge R\text{-}lin\text{-}independent\ ms$
$\longrightarrow length\ ms \leq n$
**unfolding** *R-finrank-def* **by** *fast*

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

**end**

**context** *scalar-mult*
**begin**

**abbreviation** *finrank* :: *'m set* ⇒ *bool*
  **where** *finrank* ≡ *R-scalar-mult.R-finrank UNIV smult*

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

**end**

## 3.3 Module homomorphisms

### 3.3.1 Locales

**locale** *RModuleHom* = *Domain?*: *RModule R smult M*
+ *Codomain?*: *scalar-mult smult'*
+ *GroupHom?*: *GroupHom M T*
  **for** *R* :: *'r*::*ring-1 set*
  **and** *smult* :: *'r* ⇒ *'m*::*ab-group-add* ⇒ *'m* (**infixr** · *70*)
  **and** *M* :: *'m set*
  **and** *smult'* :: *'r* ⇒ *'n*::*ab-group-add* ⇒ *'n* (**infixr** ⋆ *70*)
  **and** *T* :: *'m* ⇒ *'n*
+ **assumes** *R-map*: ⋀*r m*. *r* ∈ *R* ⟹ *m* ∈ *M* ⟹ *T* (*r* · *m*) = *r* ⋆ *T m*

**abbreviation** (**in** *RModuleHom*) *lincomb'* :: *'r list* ⇒ *'n list* ⇒ *'n* (**infix** ·⋆ *70*)
  **where** *lincomb'* ≡ *Codomain.lincomb*

**lemma** (**in** *RModule*) *RModuleHomI* :
  **assumes** *GroupHom M T*
      ⋀*r m*. *r* ∈ *R* ⟹ *m* ∈ *M* ⟹ *T* (*r* · *m*) = *smult' r* (*T m*)
  **shows** *RModuleHom R smult M smult' T*
  **by** (
        *rule RModuleHom.intro*, *rule RModule-axioms*, *rule assms*(*1*), *unfold-locales*,
        *rule assms*(*2*)
      )

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

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

**lemmas** (**in** *ModuleHom*) *hom* = *hom*

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

**locale** *ModuleEnd* = *ModuleHom smult M smult T*
  **for** *smult* :: *'r*::*ring-1* ⇒ *'m*::*ab-group-add* ⇒ *'m* (**infixr** · *70*)
  **and** *M* :: *'m set* **and** *T* :: *'m* ⇒ *'m*
+ **assumes** *endomorph*: *ImG* ⊆ *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** *bijective*: *bij-betw T M N*

**lemma** (**in** *RModule*) *RModuleIsoI* :
  **assumes** *GroupIso M T N*
    ⋀*r m*. *r* ∈ *R* ⟹ *m* ∈ *M* ⟹ *T* (*r* · *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.bijective* **by** *unfold-locales*
**qed**

### 3.3.2 Basic facts

**lemma** (**in** *RModule*) *trivial-RModuleHom* :
  ∀ *r*∈*R*. *smult' r 0* = *0* ⟹ *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* : $r \in R \Longrightarrow r \star 0 = 0$
**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** $\bigwedge n.\ n \in T\ `\ M \Longrightarrow 1 \star n = n$ **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*: $m \in T\ `\ M$
**and** *n*: $n \in T\ `\ M$
**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** $r \star m \in T\ `\ M$ **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** $r \star s \star m = (r * s) \star m$
**using** *smult-closed R-map*[*of s*] *R-map*[*of r s · 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** $\bigwedge r\ n.\ r \in R \Longrightarrow n \in T\ `\ N \Longrightarrow r \star n \in T\ `\ N$
**using** *RModule.smult-closed* **by** *force*
**qed**

**lemma** *RModHom-composite-left* :
**assumes** $T\ `\ M \subseteq N$ *RModuleHom R smult' N smult'' S*

**shows** *RModuleHom R smult M smult'' (S ∘ T)*
**proof** (*rule RModule.RModuleHomI*, *rule RModule-axioms*)
**from** *assms(1)* **show** *GroupHom M (S ∘ T)*
**using** *RModuleHom.GroupHom*[*OF assms(2)*] *GroupHom-composite-left*
**by** *auto*
**from** *assms(1)*
**show** ⋀*r m*. *r* ∈ *R* ⟹ *m* ∈ *M* ⟹ (*S* ∘ *T*) (*r* · *m*) = *smult'' r* ((*S* ∘ *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 ↓ N)*
**proof** (*rule RModuleHom.intro*)
**from** *assms* **show** *RModule R* (·) *N* **by** *fast*
**from** *assms* **show** *GroupHom N (T ↓ N)*
**using** *RModule.Group GroupHom-restrict0-subgroup* **by** *fast*
**show** *RModuleHom-axioms R* (·) *N* (⋆) (*T ↓ N*)
**proof**
**fix** *r m* **assume** *r* ∈ *R m* ∈ *N*
**with** *assms* **show** (*T ↓ N*) (*r* · *m*) = *r* ⋆ (*T ↓ N*) *m*
**using** *RModule.smult-closed R-map* **by** *fastforce*
**qed**
**qed**

**lemma** *distrib-lincomb* :
*set rs* ⊆ *R* ⟹ *set ms* ⊆ *M* ⟹ *T* (*rs* ·· *ms*) = *rs* ·⋆ *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* ⊆ *M*
*M* = *Domain.R-scalars.RSpan ms* ∀ *m*∈*set ms*. *S m* = *T m*
**shows** *S* = *T*
**proof**
**fix** *m* **show** *S m* = *T m*
**proof** (*cases m* ∈ *M*)
**case** *True*
**with** *assms(2,3)* **obtain** *rs* **where** *rs*: *set rs* ⊆ *R m* = *rs* ·· *ms*
**using** *Domain.RSpanD-lincomb-arb-len-coeffs* **by** *fast*
**from** *rs(1) assms(2)* **have** *S* (*rs* ·· *ms*) = *rs* ·⋆ (*map S ms*)
**using** *RModuleHom.distrib-lincomb*[*OF assms(1)*] **by** *simp*
**moreover from** *rs(1) assms(2)* **have** *T* (*rs* ·· *ms*) = *rs* ·⋆ (*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-contra*[*of - S*] *suppI-contra*[*of - T*] **by** *fastforce*
**qed**
**qed**

**end**

**lemma** *RSubmodule-eigenspace* :
**fixes** *smult* :: *'r*::*ring-1* ⇒ *'m*::*ab-group-add* ⇒ *'m* (**infixr** · *70*)
**assumes** *RModHom*: *RModuleHom R smult M smult T*
**and** *r*: $r \in R \bigwedge s\ m.\ s \in R \Longrightarrow m \in M \Longrightarrow s \cdot r \cdot m = r \cdot s \cdot m$
**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** $\bigwedge s\ m.\ s \in R \Longrightarrow m \in E \Longrightarrow s \cdot m \in E$
**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-endomorph-is-RModuleEnd* :
**assumes** *grpend*: *GroupEnd M T*
**and** *Rmap* : $\bigwedge r\ m.\ r \in R \Longrightarrow m \in M \Longrightarrow T\ (r \cdot m) = r \cdot (T\ m)$
**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* ⟹ *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* ≡ (*the-inv-into M T*) ↓ *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* ∈ *R n* ∈ *N*
**thus** *invT* (*r* ⋆ *n*) = *r* · *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** $\forall N{\in}set\ Ns.\ RSubmodule\ N\ add\text{-}independentS\ Ns\ r \in R$
$x \in (\bigoplus N{\leftarrow}Ns.\ N)$
**shows** $(\bigoplus Ns{\leftarrow}(r \cdot x)) = [r \cdot m.\ m{\leftarrow}(\bigoplus Ns{\leftarrow}x)]$
**proof**−
**define** *xs* **where** $xs = (\bigoplus Ns{\leftarrow}x)$
**with** *assms* **have** *x*: $xs \in listset\ Ns\ x = sum\text{-}list\ xs$
**using** *RModule.AbGroup*[*of R*] *AbGroup-inner-dirsum-el-decompI*[*of Ns x*]
**by** *auto*
**from** *assms*(*1*,*2*,*4*) *xs-def* **have** *xs-M*: $set\ xs \subseteq M$
**using** *Subgroup-RSubmodule*
*AbGroup.abSubgroup-inner-dirsum-el-decomp-set*[*OF AbGroup*]
**by** *fast*
**from** *assms*(*1*,*3*) *x*(*1*) **have** $[r \cdot m.\ m{\leftarrow}xs] \in listset\ Ns$
**using** *listset-RModule-Rsmult-closed* **by** *fast*
**moreover from** *x assms*(*3*) *xs-M* **have** $r \cdot x = sum\text{-}list\ [r \cdot m.\ m{\leftarrow}xs]$
**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** $\forall N \in set\ Ns.\ RSubmodule\ N\ add\text{-}independentS\ Ns\ n < length\ Ns$
**shows** $RModuleEnd\ R\ smult\ (\bigoplus N{\leftarrow}Ns.\ N)\ (\bigoplus Ns{\downarrow}n)$
**proof** (*rule RModule.Rmap-endomorph-is-RModuleEnd*)
**from** *assms*(*1*) **show** $RModule\ R\ smult\ (\bigoplus N{\leftarrow}Ns.\ N)$
**using** *RSubmodule-inner-dirsum* **by** *fast*
**from** *assms* **show** $GroupEnd\ (\bigoplus N{\leftarrow}Ns.\ N)\ \bigoplus Ns{\downarrow}n$
**using** *RModule.AbGroup GroupEnd-inner-dirsum-el-decomp-nth*[*of Ns*] **by** *fast*
**show** $\bigwedge r\ m.\ r \in R \Longrightarrow m \in (\bigoplus N{\leftarrow}Ns.\ N)$
$\Longrightarrow (\bigoplus Ns{\downarrow}n)\ (r \cdot m) = r \cdot ((\bigoplus Ns{\downarrow}n)\ m)$
**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* ⇒ *'v*::*ab-group-add* ⇒ *'v* (**infixr** · *70*)

**abbreviation** (**in** *fscalar-mult*) *findim* ≡ *fingen*

**locale** *VectorSpace* = *Module smult V*
**for** *smult* :: *'f*::*field* ⇒ *'v*::*ab-group-add* ⇒ *'v* (**infixr** · *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* ⟹ *FinDimVectorSpace* (·) *V*
**by** *unfold-locales fast*

**context** *VectorSpace*
**begin**

**abbreviation** *Subspace* :: *'v set* ⇒ *bool* **where** *Subspace* ≡ *Submodule*

**lemma** *SubspaceD1* : *Subspace U* ⟹ *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*: ⟦ $a \neq 0$; $u \in V$; $v \in V$; $a \cdot u = a \cdot v$ ⟧ ⟹ $u = v$
**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* :
⟦ $v \in V$; *set vs* $\subseteq V$; $a \neq 0$; $(a\#as) \cdot\cdot (v\#vs) = 0$ ⟧ ⟹ $v \in$ *Span vs*
**using** *lincomb-Cons eq-neg-iff-add-eq-0*[*of* $a \cdot v$ *as* $\cdot\cdot$ *vs*]
*neg-lincomb smult-assoc*[*of* 1 / *a a v*] *smult-lincomb SpanD-lincomb-arb-len-coeffs*
**by** *auto*

**lemma** *lin-independent-Cons-conditions* :
⟦ $v \in V$; *set vs* $\subseteq V$; $v \notin$ *Span vs*; *lin-independent vs* ⟧
⟹ *lin-independent* $(v\#vs)$
**using** *lincomb-Cons-0-imp-in-Span lin-independent-ConsI* **by** *fast*

**lemma** *coeff-n0-imp-in-Span-others* :
**assumes** $v \in V$ *set us* $\subseteq V$ *set vs* $\subseteq V$ $b \neq 0$ *length as* = *length us*
$w = (as \mathbin{@} b \# bs) \cdot\cdot (us \mathbin{@} v \# vs)$
**shows** $v \in$ *Span* $(w \# us \mathbin{@} vs)$
**proof**−
**define** $x$ **where** $x = (1 \# [-\ c.\ c{\leftarrow}as@bs]) \cdot\cdot (w \# us \mathbin{@} vs)$

```
  from assms(1,4−6) have v = (1/b) · (w + − ( (as@bs) ·· (us@vs) ))
    using lincomb-append lincomb-Cons by simp
  moreover from assms(1,2,3,6) have w: w ∈ V using lincomb-closed by simp
  ultimately have v = (1/b) · 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 ⊆ V v ∈ V set vs ⊆ V lin-independent (us @ v # vs)
          length as = length us b ≠ 0
  shows   lin-independent ( ((as @ b # bs) ·· (us @ v # vs)) # us @ vs )
proof−
  define w where w = (as @ b # bs) ·· (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) ·· (w # us @ vs) = 0
    from assms(1,3) obtain ds es fs
      where dsesfs: length ds = length vs bs ·· vs = ds ·· vs
                    length es = length vs (drop (length us) cs) ·· vs = es ·· vs
                    length fs = length us cs ·· us = fs ·· 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)←zip [c∗a. a←as] fs]
        and ys = [x+y. (x,y)←zip es [c∗d. d←ds]]
    with assms(5) dsesfs(5) have len-xs: length xs = length us
      using length-concat-map-split-zip[of - [c∗a. a←as] fs] by simp
    from A w-def assms(1−3,5) dsesfs(2,4,6)
      have 0 = c · as ·· us + fs ·· us + (c ∗ b) · v + es ·· vs + c · ds ·· 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 ... = (xs @ (c ∗ b) # ys) ·· (us @ v # vs)
      using smult-lincomb lincomb-sum lincomb-Cons lincomb-append by simp
    finally have (xs @ (c ∗ b) # ys) ·· (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*. *set ws* $\subseteq$ *V* $\wedge$ *lin-independent* (*ws* @ *us*) $\wedge$ (*Span* (*ws* @ *us*) = *V* $\vee$ *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* $\vee$ *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 w* $\notin$ *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* $\equiv$ (*set vs* $\subseteq$ *V* $\wedge$ *V* = *Span vs* $\wedge$ *lin-independent vs*)

**context** *VectorSpace*
**begin**

**lemma** *spanset-contains-basis* :
*set vs* $\subseteq$ *V* $\Longrightarrow$ $\exists$ *us*. *set us* $\subseteq$ *set vs* $\wedge$ *basis-for* (*Span vs*) *us*
**proof** (*induct vs*)
**case** *Nil* **show** *?case* **using** *lin-independent-Nil* **by** *simp*
**next**
**case** (*Cons v vs*)
**from** *this* **obtain** *ws* **where** *ws*: *set ws* $\subseteq$ *set vs basis-for* (*Span vs*) *ws* **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-len-vs: n < length vs and len-acs: length acs = length ufvs
    by   auto
  from n-len-vs fvs-def y-def bvs-def have vs-decomp: vs = fvs @ y # bvs
    using id-take-nth-drop by simp
  with uv(3) as-bs(1,2,4) cs-ds acs-def ufvs-def
    have  w-decomp: w = (acs @ b # ds) ·· (ufvs @ y # bvs)
    using lincomb-append
    by    simp
  from closed(1) vs-decomp
    have y-V: y ∈ V and fvs-V: set fvs ⊆ V and bvs-V: set bvs ⊆ V
    by   auto
  from ufvs-def fvs-V closed(2) have ufvs-V: set ufvs ⊆ V by simp
  from w-decomp ufvs-V y-V bvs-V have w-V: w ∈ V
    using lincomb-closed by simp
  have Span (us@vs) = Span (w # ufvs @ bvs)
  proof
    from vs-decomp ufvs-def have 1: Span (us@vs) = Span (y # ufvs @ bvs)
      using Span-append Span-Cons[of y bvs] Span-Cons[of y ufvs]
            Span-append[of y#ufvs bvs]
      by    (simp add: algebra-simps)
    with new-w y-V ufvs-V bvs-V show Span (w # ufvs @ bvs) ⊆ Span (us@vs)
      using Span-replace-hd by simp
    from len-acs w-decomp y-V ufvs-V bvs-V have y ∈ Span (w # ufvs @ bvs)
      using b(2) coeff-n0-imp-in-Span-others by simp
    with w-V ufvs-V bvs-V have Span (y # ufvs @ bvs) ⊆ Span (w # ufvs @ bvs)
      using Span-replace-hd by simp
    with 1 show Span (us@vs) ⊆ Span (w # ufvs @ bvs) by fast
  qed
  moreover from ufvs-V y-V bvs-V ufvs-def indep vs-decomp w-decomp len-acs
b(2)
    have  lin-independent (w # ufvs @ bvs)
    using lin-independent-replace1-by-lincomb[of ufvs y bvs acs b ds]
    by    simp
  moreover have set (w # (us@fvs) @ bvs) ⊆ Span (us@vs)
  proof−
    from new-w have w ∈ Span (us@vs) by fast
    moreover from closed have set us ⊆ Span (us@vs)
      using Span-contains-spanset-append-left by fast
    moreover from closed fvs-def have set fvs ⊆ Span (us@vs)
      using Span-contains-spanset-append-right[of us] set-take-subset by fastforce
    moreover from closed bvs-def have set bvs ⊆ Span (us@vs)
      using Span-contains-spanset-append-right[of us] set-drop-subset by fastforce
    ultimately show ?thesis by simp
  qed
  ultimately show ?thesis using ufvs-def vs-decomp by auto
qed

lemma replace-basis :
  assumes closed: set vs ⊆ V and indep-vs: lin-independent vs
```

**shows** $[\![$ *length us* $\leq$ *length vs*; *set us* $\subseteq$ *Span vs*; *lin-independent us* $]\!]$
$\Longrightarrow \exists$ *pvs*. *length pvs* = *length vs* $\wedge$ *set pvs* = *set vs*
$\wedge$ *basis-for* (*Span vs*) (*take* (*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*: *length ppvs* = *length vs set ppvs* = *set vs*
*basis-for* (*Span vs*) (*take* (*length vs*) (*us* @ *ppvs*))
**using** *lin-independent-ConsD1*[*of u us*]
**by** *auto*
**define** *fppvs bppvs*
**where** *fppvs* = *take* (*length vs* − *length us*) *ppvs*
**and** *bppvs* = *drop* (*length vs* − *length us*) *ppvs*
**with** *ppvs*(*1*) *Cons*(*2*)
**have** *ppvs-decomp*: *ppvs* = *fppvs* @ *bppvs*
**and** *len-fppvs* : *length fppvs* = *length vs* − *length us*
**by** *auto*
**from** *closed Cons*(*3*) **have** *uus-V*: *u* ∈ *V set us* ⊆ *V*
**using** *Span-closed* **by** *auto*
**from** *closed ppvs*(*2*) **have** *set ppvs* ⊆ *V* **by** *fast*
**with** *fppvs-def* **have** *fppvs-V*: *set fppvs* ⊆ *V* **using** *set-take-subset*[*of - ppvs*] **by** *fast*
**from** *fppvs-def Cons*(*2*)
**have** *prev-basis-decomp*: *take* (*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 basis-for* (*Span vs*) (*u* # *us* @ *xs* @ *ys*)
**using** *lin-independent-imp-hd-independent-from-Span*
*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** *length pvs* = *length vs set pvs* = *set vs*
*basis-for* (*Span vs*) (*take* (*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* :
$[\![$ *set vs* $\subseteq$ *V*; *lin-independent vs*; *length us* = *length vs*;
*set us* $\subseteq$ *Span vs*; *lin-independent us* $]\!] \Longrightarrow$ *basis-for* (*Span vs*) *us*
**using** *replace-basis*[*of vs us*] **by** *auto*

**lemma** *basis-for-obtain-unique-scalars* :
*basis-for V vs* $\Longrightarrow$ *v* ∈ *V* $\Longrightarrow \exists!$ *as*. *length as* = *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*: $as \equiv (THE\ ds.\ length\ ds = length\ vs \wedge v = ds \cdot\cdot vs)$
**and** *bs*: $bs \equiv (THE\ ds.\ length\ ds = length\ vs \wedge v' = ds \cdot\cdot vs)$
**and** *cs*: $cs \equiv (THE\ ds.\ length\ ds = length\ vs \wedge v+v' = ds \cdot\cdot vs)$
**shows** $cs = [a+b.\ (a,b)\leftarrow zip\ as\ bs]$
**proof**−
**from** *vs v v' as bs*
**have** *as'*: $length\ as = length\ vs \wedge v = as \cdot\cdot vs$
**and** *bs'*: $length\ bs = length\ vs \wedge v' = bs \cdot\cdot vs$
**using** *basis-for-obtain-unique-scalars theI'*[
*of* $\lambda ds.\ length\ ds = length\ vs \wedge v = ds \cdot\cdot vs$
]
*theI'*[*of* $\lambda ds.\ length\ ds = length\ vs \wedge v' = ds \cdot\cdot vs$]
**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** $length\ [a+b.\ (a,b)\leftarrow zip\ as\ bs]$
$= length\ vs \wedge v + v' = [a + b.\ (a,b)\leftarrow zip\ as\ bs] \cdot\cdot vs$
**using** *lincomb-sum*
**by** *auto*
**moreover from** *vs v v'* **have** $\exists!\ ds.\ length\ ds = length\ vs \wedge v+v' = ds \cdot\cdot vs$
**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*: $as \equiv (THE\ cs.\ length\ cs = length\ vs \wedge v = cs \cdot\cdot vs)$
**and** *bs*: $bs \equiv (THE\ cs.\ length\ cs = length\ vs \wedge a \cdot v = cs \cdot\cdot vs)$
**shows** $bs = map\ ((*)\ a)\ as$
**proof**−
**from** *vs v as* **have** $length\ as = length\ vs \wedge v = as \cdot\cdot vs$
**using** *basis-for-obtain-unique-scalars theI'*[
*of* $\lambda cs.\ length\ cs = length\ vs \wedge v = cs \cdot\cdot vs$
]
**by** *auto*
**with** *vs* **have** $length\ (map\ ((*)\ a)\ as)$
$= length\ vs \wedge a \cdot v = (map\ ((*)\ a)\ as) \cdot\cdot vs$
**using** *smult-lincomb* **by** *auto*
**moreover from** *vs v* **have** $\exists!\ cs.\ length\ cs = length\ vs \wedge a \cdot v = cs \cdot\cdot vs$
**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* $\Longrightarrow$ $\neg$ *lin-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* $\Longrightarrow$ *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* $\Longrightarrow$ *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** ∀ *vs*. *set vs* ⊆ *V* ∧ *lin-independent vs* ⟶ *length vs* ≤ *n*
**using** *finrankD*
**by** *fastforce*
**moreover from** *build-lin-independent-seq*[*of* []] **obtain** *ws*
**where** *set ws* ⊆ *V lin-independent ws Span ws* = *V* ∨ *length ws* = *Suc n*
**by** *auto*
**ultimately show** *?thesis* **by** *auto*
**qed**

**lemma** *subspace-Span-is-findim* :
⟦ *set vs* ⊆ *V*; *Subspace W*; *W* ⊆ *Span vs* ⟧ ⟹ *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* ⟹ *findim U*
**using** *findim subspace-Span-is-findim* **by** *fast*

**lemma** *basis-ex* : ∃ *vs*. *basis-for V vs*
**using** *findim basis-for-Span-ex* **by** *auto*

**lemma** *lin-independent-length-le-dim* :
*set us* ⊆ *V* ⟹ *lin-independent us* ⟹ *length us* ≤ *dim V*
**using** *basis-ex max-lin-independent-set-in-Span dim-eq-size-basis*
**by** *force*

**lemma** *too-long-lin-dependent* :
*set us* ⊆ *V* ⟹ *length us* > *dim V* ⟹ ¬ *lin-independent us*
**using** *lin-independent-length-le-dim* **by** *fastforce*

**lemma** *extend-lin-independent-to-basis* :
**assumes** *set us* ⊆ *V lin-independent us*
**shows** ∃ *vs*. *basis-for V* (*vs* @ *us*)
**proof**−
**define** *n* **where** *n* = *Suc* (*dim V* − *length us*)
**from** *assms* **obtain** *vs*
**where** *vs*: *set vs* ⊆ *V lin-independent* (*vs* @ *us*)
*Span* (*vs* @ *us*) = *V* ∨ *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* :
$U \subseteq V \Longrightarrow$ *basis-for U us* $\Longrightarrow \exists$ *vs. basis-for V* (*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* (·) *U*]
*FinDimVectorSpace.basis-ex*[*of* (·) *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* (·) *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* (·) *U*]
*FinDimVectorSpace.basis-ex*[*of* (·) *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* $\Longrightarrow$ *U* $\neq$ *V* $\Longrightarrow$ *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.FinDimVectorSpaceI FinDimVectorSpace.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** ⋀*g g′*. *g* ∈ *V* ⟹ *g′* ∈ *V* ⟹ *T* (*g* + *g′*) = *T g* + *T g′*
*supp T* ⊆ *V*
⋀*r m*. *r* ∈ *UNIV* ⟹ *m* ∈ *V* ⟹ *T* (*r* · *m*) = *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** $smult :: 'f::field \Rightarrow 'v::ab\text{-}group\text{-}add \Rightarrow 'v$
**assumes** $VectorSpaceHom\ smult\ V\ smult\ T\ T\ `\ V \subseteq V$
**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** $smult :: 'f::field \Rightarrow 'v::ab\text{-}group\text{-}add \Rightarrow 'v$ (**infixr** $\cdot$ *70*)
**and** $V :: 'v\ set$
**and** $smult' :: 'f \Rightarrow 'w::ab\text{-}group\text{-}add \Rightarrow 'w$ (**infixr** $\star$ *70*)
**and** $T :: 'v \Rightarrow 'w$
+ **fixes** $W :: 'w\ set$
**assumes** *bijective*: $bij\text{-}betw\ T\ V\ W$

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

### 4.4.2 Basic facts

**lemma** (**in** *VectorSpace*) *trivial-VectorSpaceHom* :
$(\bigwedge a.\ smult'\ a\ 0 = 0) \Longrightarrow VectorSpaceHom\ smult\ V\ smult'\ 0$
**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 ⋀v v'. v ∈ V ⟹ v' ∈ V ⟹ a ⋆ T (v + v') = a ⋆ T v + a ⋆ T v'
    using additive VectorSpace.smult-distrib-left[OF VectorSpace-Im] by simp
  have ⋀v. v ∉ V ⟹ v ∉ supp (λv. a ⋆ T v)
  proof−
    fix v assume v ∉ V
    hence a ⋆ T v = 0
      using supp suppI-contra[of - T] codomain-smult-zero by fastforce
    thus v ∉ supp (λv. a ⋆ T v) using suppD-contra by fast
  qed
  thus supp (λv. a ⋆ T v) ⊆ V by fast
  show ⋀c v. v ∈ V ⟹ a ⋆ T (c · v) = c ⋆ a ⋆ T v
   using f-map VectorSpace.smult-assoc[OF VectorSpace-Im] by (simp add: field-simps)
qed

lemma VectorSpaceHom-composite-left :
  assumes ImG ⊆ 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: set vs ⊆ V scalar-mult.Span smult vs = V
    by fast
  define ws where ws = map T vs
  with vs(1) have 1: set ws ⊆ ImG by auto
  moreover have Span ws = ImG
```

```
  proof
    show Span ws ⊆ ImG
      using 1 VectorSpace.Span-closed[OF VectorSpace-Im] by fast
    from vs ws-def show Span ws ⊇ ImG
      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 ⇒ 'w::ab-group-add ⇒ 'w (infixr ⋆ 70)
  and     lincomb' :: 'f list ⇒ 'w list ⇒ 'w (infixr ·⋆ 70)
  defines lincomb' : lincomb' ≡ scalar-mult.lincomb smult'
  assumes VSpW     : VectorSpace smult' W
  and     basisV   : basis-for V vs
  and     basisV-im : set ws ⊆ W length ws = length vs
  shows   ∃! T. VectorSpaceHom smult V smult' T ∧ map T vs = ws
proof (rule ex-ex1I)
  define T where T = restrict0 (λv. (THE as. length as = length vs ∧ v = as ·· vs) ·⋆ ws) V
  have VectorSpaceHom (·) V smult' T
  proof
    fix v v' assume vv': v ∈ V v' ∈ 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 λds. length ds = length vs ∧ v = ds ·· vs
            ]
            theI'[of λds. length ds = length vs ∧ v' = ds ·· vs] add-closed
            add-unique-scalars VectorSpace.lincomb-sum[OF VSpW]
      by    auto
  next
    from T-def show supp T ⊆ V using supp-restrict0 by fast
  next
    fix a v assume v: v ∈ V
    with basisV basisV-im(1) T-def lincomb' show T (a · v) = a ⋆ 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 ⋀i. i<length (map T vs) ⟹ map T vs ! i = ws ! i
    proof−
      fix i assume i: i < length (map T vs)
      define zs where zs = (replicate (length vs) (0::'f))[i:=1]
```

**with** *basisV i* **have** *length zs = length vs ∧ vs!i = zs ·· vs*
**using** *delta-scalars-lincomb-eq-nth* **by** *auto*
**moreover from** *basisV i* **have** *vs!i ∈ V* **by** *auto*
**ultimately show** *(map T 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 λzs. length zs = length vs ∧ vs!i = zs ·· vs]*
*VectorSpace.delta-scalars-lincomb-eq-nth[OF VSpW, of ws]*
**by** *force*
**qed**
**thus** ⋀*i. i < length (map T vs) ⟹ map T vs ! i = ws ! i* **by** *fast*
**qed**
**ultimately have** *VectorSpaceHom (·) V smult′ T ∧ map T vs = ws* **by** *fast*
**thus** ∃ *T. VectorSpaceHom (·) V smult′ T ∧ map T vs = ws* **by** *fast*
**next**
**fix** *S T* **assume**
*VectorSpaceHom (·) V smult′ S ∧ map S vs = ws*
*VectorSpaceHom (·) V smult′ T ∧ map T 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::field ⇒ 'v::ab-group-add ⇒ 'v) ⇒ 'v set ⇒ ('f ⇒ 'w::ab-group-add ⇒ 'w)*
*⇒ 'w set ⇒ ('v ⇒ 'w) set*
**where** *VectorSpaceHomSet fsmult V fsmult′ W*
*≡ {T. VectorSpaceHom fsmult V fsmult′ T} ∩ {T. T ‘ V ⊆ W}*

**abbreviation** (**in** *VectorSpace*) *VectorSpaceEndSet ≡ {S. VEnd S}*

**lemma** *VectorSpaceHomSetI* :
*VectorSpaceHom fsmult V fsmult′ T ⟹ T ‘ V ⊆ W*
*⟹ T ∈ VectorSpaceHomSet fsmult V fsmult′ W*
**unfolding** *VectorSpaceHomSet-def* **by** *fast*

**lemma** *VectorSpaceHomSetD-VectorSpaceHom* :
*T ∈ VectorSpaceHomSet fsmult V fsmult′ N*
*⟹ VectorSpaceHom fsmult V fsmult′ T*
**unfolding** *VectorSpaceHomSet-def* **by** *fast*

**lemma** *VectorSpaceHomSetD-Im* :
*T ∈ VectorSpaceHomSet fsmult V fsmult′ W ⟹ T ‘ V ⊆ 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*
= (*GroupHomSet V W*) ∩ {*T*. ∀ *a*. ∀ *v*∈*V*. *T* (*a* · *v*) = *a* ⋆ (*T v*)}
**proof**
**show** *VectorSpaceHomSet smult V smult' W*
⊆ (*GroupHomSet V W*) ∩ {*T*. ∀ *a*. ∀ *v*∈*V*. *T* (*a* · *v*) = *a* ⋆ (*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*
⊇ (*GroupHomSet V W*) ∩ {*T*. ∀ *a*. ∀ *v*∈*V*. *T* (*a* · *v*) = *a* ⋆ (*T v*)}
**proof**
**fix** *T* **assume** *T*: *T* ∈ (*GroupHomSet V W*)
∩ {*T*. ∀ *a*. ∀ *v*∈*V*. *T* (*a* · *v*) = *a* ⋆ (*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*)
∩ {*T*. ∀ *a*. ∀ *v*∈*V*. *T* (*a* · *v*) = *a* ⋆ (*T v*)}
*T* ∈ (*GroupHomSet V W*) ∩ {*T*. ∀ *a*. ∀ *v*∈*V*. *T* (*a* · *v*) = *a* ⋆ (*T v*)}
**using** *VectorSpaceHomSet-is-fmaps-in-GroupHomSet*
**by** *auto*

```
  hence S − T ∈ GroupHomSet V W
    using VectorSpace.AbGroup[OF assms] Group-GroupHomSet Group.diff-closed
    by   fast
  moreover have ⋀a v. v ∈ V ⟹ (S − T) (a · v) = a ⋆ ((S−T) v)
  proof−
    fix a v assume v ∈ V
    with ST show (S − T) (a · v) = a ⋆ ((S − T) v)
      using GroupHomSetD-Im
            VectorSpace.smult-distrib-left-diff[OF assms, of a S v T v]
      by   fastforce
  qed
  ultimately show S − T ∈ VectorSpaceHomSet (·) V (⋆) W
    using VectorSpaceHomSet-is-fmaps-in-GroupHomSet[of smult′ W] by fast
qed

lemma VectorSpace-VectorSpaceHomSet :
  fixes   smult′   :: 'f ⇒ 'w::ab-group-add ⇒ 'w (infixr ⋆ 70)
  and     hom-smult :: 'f ⇒ ('v ⇒ 'w) ⇒ ('v ⇒ 'w) (infixr ⋆· 70)
  defines hom-smult: hom-smult ≡ λa T v. a ⋆ T v
  assumes VSpW: VectorSpace smult′ W
  shows   VectorSpace hom-smult (VectorSpaceHomSet smult V smult′ W)
proof (rule VectorSpace.intro, rule Module.intro, rule RModule.intro, rule R-scalar-mult)

  from VSpW show Group (VectorSpaceHomSet (·) V (⋆) W)
    using Group-VectorSpaceHomSet by fast

  show RModule-axioms UNIV hom-smult (VectorSpaceHomSet (·) V (⋆) W)
  proof
    fix a b S T
    assume S: S ∈ VectorSpaceHomSet (·) V (⋆) W
      and  T: T ∈ VectorSpaceHomSet (·) V (⋆) W
    show a ⋆· T ∈ VectorSpaceHomSet (·) V (⋆) W
    proof (rule VectorSpaceHomSetI)
      from assms T show VectorSpaceHom (·) V (⋆) (a ⋆· T)
        using VectorSpaceHomSetD-VectorSpaceHom VectorSpaceHomSetD-Im
              VectorSpaceHom.VectorSpaceHom-scalar-mul
        by   fast
      from hom-smult show (a ⋆· T) ‘ V ⊆ W
       using VectorSpaceHomSetD-Im[OF T] VectorSpace.smult-closed[OF VSpW]
        by   auto
    qed

    show a ⋆· (S + T) = a ⋆· S + a ⋆· T
    proof
      fix v from assms show (a ⋆· (S + T)) v = (a ⋆· S + a ⋆· T) v
        using VectorSpaceHomSetD-Im[OF S] VectorSpaceHomSetD-Im[OF T]
              VectorSpace.smult-distrib-left[OF VSpW, of a S v T v]
              VectorSpaceHomSetD-VectorSpaceHom[OF S]
              VectorSpaceHomSetD-VectorSpaceHom[OF S]
```

```
        VectorSpaceHom.supp suppI-contra[of v S] suppI-contra[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 VectorSpaceHomSetD-Im[OF T] VectorSpace.smult-distrib-right
        VectorSpaceHomSetD-VectorSpaceHom[OF T] VectorSpaceHom.supp
        suppI-contra[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 VectorSpaceHomSetD-Im[OF T] VectorSpace.smult-assoc
        VectorSpaceHomSetD-VectorSpaceHom[OF T]
        VectorSpaceHom.supp suppI-contra[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 VectorSpaceHomSetD-Im VectorSpace.one-smult
        VectorSpaceHomSetD-VectorSpaceHom VectorSpaceHom.supp
        suppI-contra[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** $T$ ‘ $V \subseteq\ V$ **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** $\cdot\cdot$ *70*)
**where** $a \cdot\cdot T \equiv (\lambda v.\ a \cdot T\ v)$

**abbreviation** *end-lincomb*
:: $'f$ *list* $\Rightarrow (('v \Rightarrow 'v)$ *list*$) \Rightarrow ('v \Rightarrow 'v)$ (**infixr** $\cdots$ *70*)
**where** *end-lincomb* $\equiv$ *scalar-mult.lincomb end-smult*

**lemma** *end-smult0*: $a \cdot\cdot 0 = 0$
**using** *smult-zero* **by** *auto*

**lemma** *end-0smult*: *range* $T \subseteq V \Longrightarrow 0 \cdot\cdot T = 0$
**using** *zero-smult* **by** *fastforce*

**lemma** *end-smult-distrib-left* :
**assumes** *range* $S \subseteq V$ *range* $T \subseteq V$
**shows** $a \cdot\cdot (S + T) = a \cdot\cdot S + a \cdot\cdot T$
**proof**
**fix** $v$ **from** *assms* **show** $(a \cdot\cdot (S + T))\ v = (a \cdot\cdot S\ + a \cdot\cdot T)\ v$
**using** *smult-distrib-left*[*of a S v T v*] **by** *fastforce*
**qed**

**lemma** *end-smult-distrib-right* :
**assumes** *range* $T \subseteq V$
**shows** $(a+b) \cdot\cdot T = a \cdot\cdot T + b \cdot\cdot T$
**proof**
**fix** $v$ **from** *assms* **show** $((a+b) \cdot\cdot T)\ v = (a \cdot\cdot T + b \cdot\cdot T)\ v$
**using** *smult-distrib-right*[*of a b T v*] **by** *fastforce*
**qed**

**lemma** *end-smult-assoc* :
**assumes** *range* $T \subseteq V$
**shows** $a \cdot\cdot b \cdot\cdot T = (a * b) \cdot\cdot T$
**proof**
**fix** $v$ **from** *assms* **show** $(a \cdot\cdot b \cdot\cdot T)\ v = ((a * b) \cdot\cdot T)\ v$
**using** *smult-assoc*[*of a b T v*] **by** *fastforce*
**qed**

**lemma** *end-smult-comp-comm-left* : $(a \cdot\cdot T) \circ S = a \cdot\cdot (T \circ S)$
**by** *auto*

**lemma** *end-idhom* : *VEnd* $(id{\downarrow}V)$
**by** (*rule VectorSpaceEnd.intro*, *rule VectorSpaceHom-idhom*, *unfold-locales*) *auto*

**lemma** *VectorSpaceEndSet-is-VectorSpaceHomSet* :
$VectorSpaceHomSet\ smult\ V\ smult\ V = \{T.\ VEnd\ T\}$
**proof**
**show** $VectorSpaceHomSet\ smult\ V\ smult\ V \subseteq \{T.\ VEnd\ T\}$
**using** *VectorSpaceHomSetD-VectorSpaceHom VectorSpaceHomSetD-Im VectorSpaceEndI*
**by** *fast*
**show** $VectorSpaceHomSet\ smult\ V\ smult\ V \supseteq \{T.\ VEnd\ T\}$
**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 VectorSpaceEnd-Set*
**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* :
$range\ S \subseteq V \Longrightarrow T \circ (a \cdot\cdot S) = a \cdot\cdot (T \circ S)$
**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* ⟹ *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* ⟧ ⟹ ¬ *inj-on S V* ∨ ¬ *inj-on T V*
**using** *inj-comp-end* **by** *fast*

### 4.4.5 Polynomials of endomorphisms

**context** *VectorSpaceEnd*
**begin**

**primrec** *endpow* :: *nat* ⇒ (*'v*⇒*'v*)
**where** *endpow0*: *endpow 0* = *id*↓*V*
| *endpowSuc*: *endpow* (*Suc n*) = *T* ∘ (*endpow n*)

**definition** *polymap* :: *'f poly* ⇒ (*'v*⇒*'v*)

**where** *polymap p* ≡ (*coeffs p*) ⋯ (*map endpow* [*0..<Suc* (*degree p*)])

**lemma** *VEnd-endpow* : *VEnd* (*endpow n*)
**proof** (*induct n*)
**case** *0* **show** *?case* **using** *end-idhom* **by** *simp*
**next**
**case** (*Suc k*)
**moreover have** *VEnd T* **..**
**ultimately have** *VEnd* (*T* ∘ (*endpow k*)) **using** *VEnd-composite-right* **by** *fast*
**moreover have** *endpow* (*Suc k*) = *T* ∘ (*endpow k*) **by** *simp*
**ultimately show** *VEnd* (*endpow* (*Suc k*)) **by** *simp*
**qed**

**lemma** *endpow-list-apply-closed* :
*v* ∈ *V* ⟹ *set* (*map* (λ*S*. *S v*) (*map endpow* [*0..<k*])) ⊆ *V*
**using** *VEnd-endpow VectorSpaceEnd.endomorph* **by** *fastforce*

**lemma** *map-endpow-Suc* :
*map endpow* [*0..<Suc n*] = (*id*↓*V*) # *map* ((∘) *T*) (*map endpow* [*0..<n*])
**proof** (*induct n*)
**case** (*Suc k*)
**hence** *map endpow* [*0..<Suc* (*Suc k*)] = *id* ↓ *V*
# *map* ((∘) *T*) (*map endpow* [*0..<k*]) @ *map* ((∘) *T*) [*endpow k*]
**by** *auto*
**also have** ... = *id* ↓ *V* # *map* ((∘) *T*) (*map endpow* ([*0..<Suc k*])) **by** *simp*
**finally show** *?case* **by** *fast*
**qed** *simp*

**lemma** *T-endpow-list-apply-commute* :
*map T* (*map* (λ*S*. *S v*) (*map endpow* [*0..<n*]))
= *map* (λ*S*. *S v*) (*map* ((∘) *T*) (*map endpow* [*0..<n*]))
**by** (*induct n*) *auto*

**lemma** *polymap0* : *polymap 0* = *0*
**using** *polymap-def scalar-mult.lincomb-Nil* **by** *force*

**lemma** *VEnd-polymap* : *VEnd* (*polymap p*)
**proof**−
**have** *set* (*map endpow* [*0..<Suc* (*degree p*)]) ⊆ {*S*. *VEnd S*}
**using** *VEnd-endpow* **by** *auto*
**thus** *?thesis*
**using** *polymap-def VectorSpace-VectorSpaceEndSet VectorSpace.lincomb-closed*
**by** *fastforce*
**qed**

**lemma** *polymap-pCons* : *polymap* (*pCons a p*) = *a* ·· (*id*↓*V*) + (*T* ∘ (*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 ≠ 0
    define zmap where zmap = (0::'v⇒'v)
    from a p have polymap (pCons a p) = a ·· (endpow 0)
      using polymap-def scalar-mult.lincomb-singles by simp
    moreover have a ·· (id↓V) + (T ∘ (polymap p)) = a ·· (id↓V)
      using p polymap0 comp-zero by simp
    ultimately show ?thesis by simp
  qed
next
  assume p ≠ 0
  hence polymap (pCons a p)
          = (a # (coeffs p)) ••• (map endpow [0..<Suc (Suc (degree p))])
    using polymap-def by simp
  also have ... = (a # (coeffs p))
                  ••• ((id↓V) # map ((∘) T) (map endpow [0..<Suc (degree p)]))
    using map-endpow-Suc[of Suc (degree p)] by fastforce
  also have ... = a ·· (id↓V) + (coeffs p)
                  ••• (map ((∘) T) (map endpow [0..<Suc (degree p)]))
    using scalar-mult.lincomb-Cons by simp
  also have ... = a ·· (id↓V) + (∑(c,S)
                  ←zip (coeffs p) (map ((∘) T) (map endpow [0..<Suc (degree p)])).
                     c ·· S)
    using scalar-mult.lincomb-def by simp
  finally have calc:
    polymap (pCons a p) = a ·· (id↓V)
      + (∑(c,k)←zip (coeffs p) [0..<Suc (degree p)]. c ·· (T ∘ (endpow k)))
    using sum-list-prod-map2[
          of λc S. c ·· S coeffs p (∘) T map endpow [0..<Suc (degree p)]
        ]
        sum-list-prod-map2[
          of λc S. c ·· (T ∘ S) coeffs p endpow [0..<Suc (degree p)]
        ]
    by    simp
  show ?thesis
  proof
    fix v show polymap (pCons a p) v = ((a ·· (id↓V)) + (T ∘ (polymap p))) v
    proof (cases v ∈ V)
      case True
      with calc
        have polymap (pCons a p) v = a · v + (∑(c,k)
                ←zip (coeffs p) [0..<Suc (degree p)]. c · T (endpow k v))
        using sum-list-prod-fun-apply[of λc k. c ·· (T ∘ (endpow k))] by simp
      hence polymap (pCons a p) v = a · v + (coeffs p) ·· (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-contra by fast
    moreover from False have ((a ·· (id↓V)) + (T ∘ (polymap p))) v = 0
      using smult-zero VEnd-polymap[of p] VectorSpaceEnd.supp suppI-contra
          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 : polymap (Polynomial.smult a p) = a ·· polymap p
proof (induct p)
  case 0 show polymap (Polynomial.smult a 0) = a ·· polymap 0
    using polymap0 end-smult0 by simp
next
  case (pCons b p)
  hence polymap (Polynomial.smult a (pCons b p))
            = a ·· b ·· (id↓V) + a ·· (T ∘ 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 polymap (Polynomial.smult a (pCons b p)) = a ·· (polymap (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 : polymap (p * q) = (polymap p) ∘ (polymap q)
proof (induct p)
  case 0 show ?case using polymap0 by auto
next
  case (pCons a p)
  have polymap (pCons a p * q) = a ·· polymap q + (T ∘ (polymap (p*q)))
    using polymap-plus polymap-polysmult polymap-pCons end-idhom
          end-0smult[OF VectorSpaceEnd.range]
    by    simp
  also from pCons(2)
    have ... = a ·· ((id↓V) ∘ polymap q) + (T ∘ polymap p ∘ polymap q)
    using VectorSpaceEnd.endomorph[OF VEnd-polymap]
          VectorSpaceEnd.idhom-left[OF VEnd-polymap]
    by    auto
  finally show polymap (pCons a p * q) = (polymap (pCons a p)) ∘ (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 ∈ V
  shows   polymap p v = (coeffs p)
              ⋅⋅ (map (λS. S v) (map endpow [0..<Suc (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 polymap (pCons a p) = a ⋅⋅ id↓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 polymap (pCons a p) v = a ⋅ v
                + T (coeffs p ⋅⋅ map (λS. S v) (map endpow [0..<Suc (degree p)]))
      using polymap-pCons by simp
    with assms pCons(1)
      have  1: polymap (pCons a p) v = (coeffs (pCons a p)) ⋅⋅ (v #
                    map T (map (λS. S v) (map endpow [0..<Suc (degree p)])))
    using endpow-list-apply-closed[of v Suc (degree p)] distrib-lincomb lincomb-Cons
      by  auto
    have 2: map T (map (λS. S v) (map endpow [0..<Suc (degree p)]))
              = map (λS. S v) (map ((∘) T) (map endpow [0..<Suc (degree p)]))
      using T-endpow-list-apply-commute[of v Suc (degree p)] by simp
    from 1 2
      have  polymap (pCons a p) v = (coeffs (pCons a p)) ⋅⋅ (v #
                map (λS. S v) (map ((∘) T) (map endpow [0..<Suc (degree p)])))
      using subst[
            OF 2, of λx. polymap (pCons a p) v = (coeffs (pCons a p)) ⋅⋅ (v # x)
          ]
      by  simp
    with assms
      have 3: polymap (pCons a p) v = (coeffs (pCons a p))
                  ⋅⋅ (map (λS. S v) (id↓V # map ((∘) T)
                    (map endpow [0..<Suc (degree p)])))
      by  simp
    from False pCons(1)
      have  4: id ↓ V # map ((∘) T) (map endpow [0..<Suc (degree p)])
                    = map endpow [0..<Suc (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 λx. polymap (pCons a p) v
                = (coeffs (pCons a p)) ⋅⋅ (map (λS. S v) x)
        ]
    by   simp
 qed
qed

lemma polymap-apply-linear : v ∈ V ⟹ polymap [:−c, 1:] v = T v − c ⋅ v
 using polymap-apply lincomb-def neg-smult endomorph by auto

lemma polymap-const-inj :
 assumes degree p = 0 p ≠ 0
 shows   inj-on (polymap p) V
proof (rule inj-onI)
 fix u v assume uv: u ∈ V v ∈ V polymap p u = polymap p v
 from assms have p: coeffs p = [coeff p 0] unfolding coeffs-def by simp
 from uv assms have (coeff p 0) ⋅ u = (coeff p 0) ⋅ 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 :
 ¬ inj-on (polymap (p ∗ q)) V
     ⟹ ¬ inj-on (polymap p) V ∨ ¬ inj-on (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: ⋀p::'f poly. degree p > 0 ⟹ ∃ c. poly p c = 0
 shows   p ≠ 0 ⟹ ¬ inj-on (polymap p) V
           ⟹ ∃ c. ¬ inj-on (polymap [:−c, 1:]) V
proof (induct n ≡ 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: poly p c = 0 by fastforce
 define q where q = 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 = degree q
      using degree-synthetic-div [of p c] q-def ‹Suc n = degree p›
      by auto
    moreover have q ≠ 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 FinDimVectorSpace) endomorph-has-eigenvector :
  assumes alg-closed: ⋀p::'a poly. degree p > 0 ⟹ ∃ c. poly p c = 0
  and     dim       : dim V > 0
  and     endo      : VectorSpaceEnd smult V T
  shows   ∃ c u. u ∈ V ∧ u ≠ 0 ∧ T u = c · u
proof−
  define Tpolymap where Tpolymap = VectorSpaceEnd.polymap smult V T
  from dim obtain v where v: v ∈ V v ≠ 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: set Tpows-v ⊆ V
    using VectorSpaceEnd.endpow-list-apply-closed by fast
 moreover from Tpows-v-def Tpows-def Tpows-v-V have ¬ lin-independent Tpows-v
    using too-long-lin-dependent by simp
  ultimately obtain as
    where as: set as ≠ 0 length as = length Tpows-v as •• 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 ≠ 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' = take (Suc (degree p)) Tpows
  proof−
    from Tpows-def
      have  1: take (Suc (degree p)) Tpows = 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 - λS. 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 ¬ 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 ≠ {}
  using VectorSpaceEnd.VEnd-polymap[OF endo] VectorSpaceEnd.Ker0-imp-inj-on
  by   fast
 from this obtain u where u ∈ V Tpolymap [:−c, 1:] u = 0 u ≠ 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 ⇒ 'v::ab-group-add ⇒ 'v (infixr · 70)
begin

definition fsmult :: 'r ⇒ 'v ⇒ 'v (infixr ♯· 70) where a ♯· v ≡ (a δδ 0) · v
abbreviation flincomb :: 'r list ⇒ 'v list ⇒ 'v (infixr ·♯· 70)
 where as ·♯· vs ≡ scalar-mult.lincomb fsmult as vs
abbreviation f-lin-independent :: 'v list ⇒ bool
 where f-lin-independent ≡ scalar-mult.lin-independent fsmult
```

**abbreviation** *fSpan* :: *'v list* ⇒ *'v set* **where** *fSpan* ≡ *scalar-mult.Span fsmult*
**definition** *Gmult* :: *'g* ⇒ *'v* ⇒ *'v* (**infixr** ∗· *70*) **where** *g* ∗· *v* ≡ (*1* δδ *g*) · *v*

**lemmas** *R-scalar-mult* = *R-scalar-mult*

**lemma** *fsmultD* : *a* ♯· *v* = (*a* δδ *0*) · *v*
**unfolding** *fsmult-def* **by** *fast*

**lemma** *GmultD* : *g* ∗· *v* = (*1* δδ *g*) · *v*
**unfolding** *Gmult-def* **by** *fast*

**definition** *negGorbit-list* :: *'g list* ⇒ (*'a* ⇒ *'v*) ⇒ *'a list* ⇒ *'v list list*
**where** *negGorbit-list gs T as* ≡ *map* (λ*g*. *map* (*Gmult* (−*g*) ∘ *T*) *as*) *gs*

**lemma** *negGorbit-Cons* :
*negGorbit-list* (*g*#*gs*) *T as*
= (*map* (*Gmult* (−*g*) ∘ *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* ∈ *set* (*negGorbit-list gs T as*) ⟹ *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* :
⋀*i*. *i* < *length gs* ⟹ (*negGorbit-list gs T as*)!*i* = *map* (*Gmult* (−*gs*!*i*) ∘ *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* ⇒ *'v*::*ab-group-add* ⇒ *'v* (**infixr** · *70*)
**and** *V* :: *'v set*

**sublocale** *FGModule* < *aezfun-scalar-mult* **proof**− **qed**

**lemma** (**in** *Group*) *trivial-FGModule* :
  **fixes** *smult* :: ('*f*::*field*, '*g*) *aezfun* ⇒ '*v*::*ab-group-add* ⇒ '*v*
  **assumes** *smult-zero*: ∀ *a*∈*group-ring*. *smult a* (*0*::'*v*) = *0*
  **shows** *FGModule G smult* (*0*::'*v set*)
**proof** (*rule FGModule.intro*)
  **from** *assms* **show** *RModule group-ring smult 0*
    **using** *Ring1-RG trivial-RModule* **by** *fast*
**qed** (*unfold-locales*)

**context** *FGModule*
**begin**

**abbreviation** *FG* :: ('*f*,'*g*) *aezfun set* **where** *FG* ≡ *ActingGroup.group-ring*
**abbreviation** *FGSubmodule* ≡ *RSubmodule*
**abbreviation** *FG-proj* ≡ *ActingGroup.RG-proj*

**lemma** *GroupG*: *Group G* **..**

**lemmas** *zero-closed* = *zero-closed*
**lemmas** *neg-closed* = *neg-closed*
**lemmas** *diff-closed* = *diff-closed*
**lemmas** *zero-smult* = *zero-smult*
**lemmas** *smult-zero* = *smult-zero*
**lemmas** *AbGroup* = *AbGroup*
**lemmas** *sum-closed* = *AbGroup.sum-closed*[*OF AbGroup*]
**lemmas** *FGSubmoduleI* = *RSubmoduleI*
**lemmas** *FG-proj-mult-leftdelta* = *ActingGroup.RG-proj-mult-leftdelta*
**lemmas** *FG-proj-mult-right* = *ActingGroup.RG-proj-mult-right*
**lemmas** *FG-el-decomp* = *ActingGroup.RG-el-decomp-aezdeltafun*

**lemma** *FG-n0*: *FG* ≠ *0* **using** *ActingGroup.RG-n0* **by** *fast*

**lemma** *FG-proj-in-FG* : *FG-proj x* ∈ *FG*
  **using** *ActingGroup.RG-proj-in-RG* **by** *fast*

**lemma** *FG-fddg-closed* : *g* ∈ *G* ⟹ *a* *δδ* *g* ∈ *FG*
  **using** *ActingGroup.RG-aezdeltafun-closed* **by** *fast*

**lemma** *FG-fdd0-closed* : *a* *δδ* *0* ∈ *FG*
  **using** *ActingGroup.RG-aezdelta0fun-closed* **by** *fast*

**lemma** *Gmult-closed* : *g* ∈ *G* ⟹ *v* ∈ *V* ⟹ *g* ∗· *v* ∈ *V*
  **using** *FG-fddg-closed smult-closed GmultD* **by** *simp*

**lemma** *map-Gmult-closed* :
  *g* ∈ *G* ⟹ *set vs* ⊆ *V* ⟹ *set* (*map* ((∗·) *g*) *vs*) ⊆ *V*
  **using** *Gmult-def FG-fddg-closed map-smult-closed*[*of 1* *δδ* *g vs*] **by** *auto*

**lemma** *Gmult0* :
**assumes** $v \in V$
**shows** $0 \ast\cdot v = v$
**proof**−
**have** $0 \ast\cdot v = (1 \ \delta\delta \ 0) \cdot v$ **using** *GmultD* **by** *fast*
**moreover have** $1 \ \delta\delta \ 0 = (1::('f,'g) \ aezfun)$ **using** *one-aezfun-transfer* **by** *fast*
**ultimately have** $0 \ast\cdot v = (1::('f,'g) \ aezfun) \cdot v$ **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** $g \ast\cdot h \ast\cdot v = (g + h) \ast\cdot v$
**proof**−
**define** $n$ **where** $n = (1::'f)$
**with** *assms* **have** $g \ast\cdot h \ast\cdot v = ((n \ \delta\delta \ g) * (n \ \delta\delta \ h)) \cdot v$
**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* :
$[\![ g \in G; \ v \in V; \ v' \in V ]\!] \Longrightarrow g \ast\cdot (v + v') = g \ast\cdot v + g \ast\cdot v'$
**using** *GmultD FG-fddg-closed* **by** *simp*

**lemma** *neg-Gmult* : $g \in G \Longrightarrow v \in V \Longrightarrow g \ast\cdot (- \ v) = - \ (g \ast\cdot v)$
**using** *GmultD FG-fddg-closed smult-neg* **by** *simp*

**lemma** *Gmult-neg-left* : $g \in G \Longrightarrow v \in V \Longrightarrow (- \ g) \ast\cdot g \ast\cdot v = v$
**using** *ActingGroup.neg-closed Gmult-assoc*[*of* $-$ *g g*] *Gmult0* **by** *simp*

**lemma** *fddg-smult-decomp* : $g \in G \Longrightarrow v \in V \Longrightarrow (f \ \delta\delta \ g) \cdot v = f \ \sharp\cdot \ g \ast\cdot v$
**using** *aezdeltafun-decomp*[*of f g*] *FG-fddg-closed FG-fdd0-closed GmultD*
*fsmult-def*
**by** *simp*

**lemma** *sum-list-aezdeltafun-smult-distrib* :
**assumes** $v \in V \ set \ (map \ snd \ fgs) \subseteq G$
**shows** $(\sum (f,g)\leftarrow fgs. \ f \ \delta\delta \ g) \cdot v = (\sum (f,g)\leftarrow fgs. \ f \ \sharp\cdot \ g \ast\cdot v)$
**proof**−
**from** *assms*(*2*) **have** *set* (*map* (*case-prod aezdeltafun*) *fgs*) $\subseteq$ *FG*
**using** *FG-fddg-closed* **by** *auto*
**with** *assms*(*1*) **have** $(\sum (f,g)\leftarrow fgs. \ f \ \delta\delta \ g) \cdot v = \ (\sum (f,g)\leftarrow fgs. \ (f \ \delta\delta \ g) \cdot v)$
**using** *sum-list-prod-map-smult-distrib* **by** *auto*
**also have** $\ldots = (\sum (f,g)\leftarrow fgs. \ f \ \sharp\cdot \ g \ast\cdot v)$
**using** *assms fddg-smult-decomp*
*sum-list-prod-cong*[*of fgs* $\lambda \ f \ g. \ (f \ \delta\delta \ g) \cdot v \ \lambda \ f \ g. \ f \ \sharp\cdot \ g \ast\cdot v$]

**by** *fastforce*
**finally show** *?thesis* **by** *fast*
**qed**

**abbreviation** *GSubspace* ≡ *RSubmodule*
**abbreviation** *GSpan* ≡ *RSpan*
**abbreviation** *G-fingen* ≡ *R-fingen*

**lemma** *GSubspaceI* : *FGModule G smult U* ⟹ *U* ⊆ *V* ⟹ *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* (·) *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*) (·) *V*
**using** *ActingGroup.Subgroup-imp-Subring SModule-restrict-scalars* **by** *fast*
**qed**

**lemma** *negGorbit-list-V* :
**assumes** *set gs* ⊆ *G T* ‘ (*set as*) ⊆ *V*
**shows** *set* (*concat* (*negGorbit-list gs T as*)) ⊆ *V*
**proof**−
**from** *assms*(*2*)
**have** *set* (*concat* (*negGorbit-list gs T as*)) ⊆ (⋃ *g*∈*set gs*. (*Gmult* (−*g*)) ‘ *V*)
**using** *set-concat negGorbit-list-def*[*of gs T as*]
**by** *force*
**moreover from** *assms*(*1*) **have** ⋀*g*. *g* ∈ *set gs* ⟹ (*Gmult* (−*g*)) ‘ *V* ⊆ *V*
**using** *ActingGroup.neg-closed Gmult-closed* **by** *fast*
**ultimately show** *?thesis* **by** *fast*
**qed**

**lemma** *negGorbit-list-Cons0* :
*T* ‘ (*set as*) ⊆ *V*
⟹ *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** $\bigwedge v.\ v \in V \Longrightarrow a \sharp\cdot v \in V$
**using** *fsmult-def smult-closed FG-fdd0-closed* **by** *simp*
**next**
**fix** *a*::*'f* **show** $\bigwedge u\ v.\ u \in V \Longrightarrow v \in V \Longrightarrow a \sharp\cdot (u + v) = a \sharp\cdot u + a \sharp\cdot v$
**using** *fsmult-def FG-fdd0-closed* **by** *simp*
**next**
**fix** *a b* :: *'f* **and** *v* :: *'v* **assume** *v*: $v \in V$
**have** $(a+b) \sharp\cdot v = (a\ \delta\delta\ 0 + b\ \delta\delta\ 0) \cdot v$
**using** *aezdeltafun-plus*[*of a b 0*] *arg-cong*[*of - - λr. r · v*] *fsmult-def* **by** *fastforce*
**with** *v* **show** $(a+b) \sharp\cdot v = a \sharp\cdot v + b \sharp\cdot v$
**using** *fsmult-def FG-fdd0-closed* **by** *simp*
**next**
**fix** *a b* :: *'f* **show** $\bigwedge v.\ v \in V \Longrightarrow a \sharp\cdot (b \sharp\cdot v) = (a * b) \sharp\cdot v$
**using** *times-aezdeltafun-aezdeltafun*[*of a 0 b 0*] *arg-cong fsmult-def FG-fdd0-closed*
**by** *simp*
**next**
**fix** *v* :: *'v* **assume** $v \in V$ **thus** $1 \sharp\cdot v = v$
**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* : $v \in V \Longrightarrow a \sharp\cdot v \in V$
**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*

$= VectorSpace.in\text{-}Span\text{-}obtain\text{-}same\text{-}length\text{-}coeffs\ [OF\ fVectorSpace]$

**lemma** *fsmult-smult-comm* : $r \in FG \Longrightarrow v \in V \Longrightarrow a \sharp\cdot r \cdot v = r \cdot a \sharp\cdot v$
**using** *fsmultD FG-fdd0-closed smult-assoc aezdelta0fun-commutes*[*of r*] **by** *simp*

**lemma** *fsmult-Gmult-comm* : $g \in G \Longrightarrow v \in V \Longrightarrow a \sharp\cdot g *\cdot v = g *\cdot a \sharp\cdot v$
**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** $g *\cdot as \cdot\sharp\cdot vs = as \cdot\sharp\cdot (map\ (Gmult\ g)\ vs)$
**proof**−
**have** $g *\cdot as \cdot\sharp\cdot vs = (1\ \delta\delta\ g) \cdot (\sum (a,v) \leftarrow zip\ as\ vs.\ a \sharp\cdot v)$
**using** *Gmult-def scalar-mult.lincomb-def*[*of fsmult*] **by** *simp*
**with** *assms* **have** $g *\cdot as \cdot\sharp\cdot vs$
$= sum\text{-}list\ (map\ ((\cdot)\ (1\ \delta\delta\ g) \circ (\lambda(x,\ y).\ x \sharp\cdot y))\ (zip\ as\ vs))$
**using** *set-zip-rightD fsmult-closed FG-fddg-closed*[*of g 1::'f*]
*smult-sum-list-distrib*[*of 1 δδ g map* (*case-prod* (♯·)) (*zip as vs*)]
*map-map*[*of* (·) (*1 δδ g*) *case-prod* (♯·) *zip as vs*]
**by** *fastforce*
**moreover have** $(\cdot)\ (1\ \delta\delta\ g) \circ (\lambda(x,\ y).\ x \sharp\cdot y) = (\lambda(x,y).\ (1\ \delta\delta\ g) \cdot (x \sharp\cdot y))$
**by** *auto*
**ultimately have** $g *\cdot as \cdot\sharp\cdot vs = sum\text{-}list\ (map\ (\lambda(x,y).\ g *\cdot x \sharp\cdot y)\ (zip\ as\ vs))$
**using** *Gmult-def* **by** *simp*
**moreover from** *assms* **have** $\forall (x,y) \in set\ (zip\ as\ vs).\ g *\cdot x \sharp\cdot y = x \sharp\cdot g *\cdot y$
**using** *set-zip-rightD fsmult-Gmult-comm* **by** *fastforce*
**ultimately have** $g *\cdot as \cdot\sharp\cdot vs$
$= sum\text{-}list\ (map\ (\lambda(x,y).\ x \sharp\cdot y)\ (zip\ as\ (map\ (Gmult\ g)\ vs)))$
**using** *sum-list-prod-cong sum-list-prod-map2*[*of* $\lambda x\ y.\ x \sharp\cdot y$ *as Gmult g*]
**by** *force*
**thus** *?thesis* **using** *scalar-mult.lincomb-def*[*of fsmult*] **by** *simp*
**qed**

**lemma** *GSubspace-is-Subspace* :
*GSubspace U* $\Longrightarrow$ *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** ⋀$v$ $v'$. $v \in V \Longrightarrow v' \in V \Longrightarrow T\ (v + v') = T\ v + T\ v'$
*supp* $T \subseteq V$ ⋀$r$ $m$. $r \in FG \Longrightarrow m \in V \Longrightarrow T\ (r \cdot m) = smult'\ r\ (T\ m)$
**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*: *ImG* ⊆ $V$

**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** *bijective*: *bij-betw T V W*

**abbreviation** (**in** *FGModule*) *isomorphic* ::
(($'f$,$'g$) *aezfun* ⇒ $'w$::*ab-group-add* ⇒ $'w$) ⇒ $'w$ *set* ⇒ *bool*
**where** *isomorphic smult' W* ≡ (∃ $T$. *FGModuleIso G smult V smult' T W*)

### 5.4.2 Basic facts

**context** *FGModule*
**begin**

**lemma** *trivial-FGModuleHom* :
**assumes** ⋀$r$. $r \in FG \Longrightarrow smult'\ r\ 0 = 0$
**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* : ⋀*g v*. *g* ∈ *G* ⟹ *v* ∈ *V* ⟹ *T* (*g* ⋆· *v*) = *g* ⋆⋆ (*T v*)
  **shows** *FGModuleHom G smult V smult' T*
**proof**

  **show** ⋀*v v'*. *v* ∈ *V* ⟹ *v'* ∈ *V* ⟹ *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** ⋀*r v*. *r* ∈ *FG* ⟹ *v* ∈ *V* ⟹ *T* (*r* · *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* = (∑ (*f*,*g*)←*fgs*. *f* *δδ* *g*)
      **using** *FG-el-decomp*
      **by** *fast*
    **from** *fgs v* **have** *r* · *v* = (∑ (*f*,*g*)←*fgs*. *f* ♯· *g* ⋆· *v*)
      **using** *sum-list-aezdeltafun-smult-distrib* **by** *simp*
    **moreover from** *v fgs*(*1*) **have** *set* (*map* (λ(*f*,*g*). *f* ♯· *g* ⋆· *v*) *fgs*) ⊆ *V*
      **using** *Gmult-closed fsmult-closed* **by** *auto*
    **ultimately have** *T* (*r* · *v*) = (∑ (*f*,*g*)←*fgs*. *T* (*f* ♯· *g* ⋆· *v*))
      **using** *hom VectorSpaceHom.im-sum-list-prod* **by** *auto*
    **moreover from** *hom G-map fgs*(*1*) *v*
      **have** ∀(*f*,*g*) ∈ *set fgs*. *T* (*f* ♯· *g* ⋆· *v*) = *f* ♯⋆ *g* ⋆⋆ *T v*
      **using** *Gmult-closed VectorSpaceHom.f-map*[*of fsmult V fsmult' T*]
      **by** *auto*
    **ultimately have** *T* (*r* · *v*) = (∑ (*f*,*g*)←*fgs*. *f* ♯⋆ *g* ⋆⋆ *T v*)
      **using** *sum-list-prod-cong* **by** *simp*
    **with** *v fgs fsmult' Gmult' Im-W*(*2*) **show** *T* (*r* · *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-im-imp-diff-in-Ker* = *eq-im-imp-diff-in-Ker*
**lemmas** *im-submodule* = *im-submodule*
**lemmas** *fsmultD'* = *aezfun-scalar-mult.fsmultD*[*of smult'*]
**lemmas** *GmultD'* = *aezfun-scalar-mult.GmultD*[*of smult'*]

**lemma** *f-map* : $v \in V \Longrightarrow T\ (a\ \sharp\cdot\ v) = a\ \sharp\star\ T\ v$
**using** *fsmultD ActingGroup.RG-aezdelta0fun-closed*[*of a*] *FG-map fsmultD'*
**by** *simp*

**lemma** *G-map* : $g \in G \Longrightarrow v \in V \Longrightarrow T\ (g\ *\cdot\ v) = g\ **\star\ T\ v$
**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* ⊆ *W*
**shows** *FGModuleHom G smult V smult'' (S* ∘ *T)*
**proof** (*rule FGModuleHom.intro*)
**from** *assms*(*2*) **show** *RModuleHom FG smult V smult'' (S* ∘ *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* (·) *V*
**using** *FGModule.restriction-to-subgroup-is-module* **by** *fast*

**from** *supp* **show** *supp T* ⊆ *V* **by** *fast*
**from** *assms*
**show** ⋀*r m*. ⟦ *r* ∈ (*Group.group-ring H*); *m* ∈ *V* ⟧ ⟹ *T* (*r* · *m*) = *r* ⋆ *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*′ (λ*v*. *a* ♯⋆ *T v*)
**proof**
**have** *vsphom*: *VectorSpaceHom fsmult V fsmult*′ (λ*v*. *a* ♯⋆ *T v*)
**using** *VectorSpaceHom.VectorSpaceHom-scalar-mul*[*OF VectorSpaceHom*]
**by** *fast*
**thus** ⋀*v v*′. *v* ∈ *V* ⟹ *v*′ ∈ *V* ⟹ *a* ♯⋆ *T* (*v* + *v*′) = *a* ♯⋆ *T v* + *a* ♯⋆ *T v*′
**using** *VectorSpaceHom.additive*[*of fsmult V*] **by** *auto*
**from** *vsphom* **show** *supp* (λ*v*. *a* ♯⋆ *T v*) ⊆ *V*
**using** *VectorSpaceHom.supp* **by** *fast*
**next**
**fix** *r v* **assume** *rv*: *r* ∈ *FG v* ∈ *V*
**thus** *a* ♯⋆ *T* (*r* · *v*) = *r* ⋆ *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*) *aezfun* ⇒ ′*v* ⇒ ′*v* (**infixr** · *70*)
**assumes** *FGModHom*: *FGModuleHom G smult V smult T*
**defines** *E* : *E* ≡ {*v* ∈ *V*. *T v* = *aezfun-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* δδ *0* ∈ *FGModule.FG G*
**using** *FGModuleHom.FG-fdd0-closed*[*OF FGModHom*] **by** *fast*
**show** ⋀*s v*. *s* ∈ *FGModule.FG G* ⟹ *v* ∈ *V* ⟹ *s* · (*e* δδ *0*) · *v* = (*e* δδ *0*) · *s* · *v*
**using** *FGModuleHom.fsmult-smult-domain-comm*[*OF FGModHom*]

*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* ⇒ *'v*::*ab-group-add* ⇒ *'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*: ⋀*g v*. *g* ∈ *G* ⟹ *v* ∈ *V* ⟹ *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* ⊆ *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* :
⟦ ∀ *U*∈*set Us*. *GSubspace U*; *add-independentS Us*; *n* < *length Us* ⟧
⟹ *FGModuleEnd G smult* (⨁ *U*←*Us*. *U*) (⨁ *Us*↓*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* ≡ (*the-inv-into V T*) ↓ *W*

**lemma** *RModuleIso* : *RModuleIso FG smult V smult′ T W*
**proof** (*rule RModuleIso.intro*)
  **show** *RModuleHom FG* (·) *V* (⋆) *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* ↓ *U*) (*T* ‘ *U*)
**proof** (*rule FGModuleIso.intro*)
  **from** *assms* **show** *FGModuleHom G* (·) *U* (⋆) (*T* ↓ *U*)
    **using** *FGModuleHom-restrict0-GSubspace* **by** *fast*
  **show** *FGModuleIso-axioms U* (*T* ↓ *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* ↓ *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* (⋆) *W* (·) *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 ∘ T) X*
**proof** (*rule FGModuleIso.intro*)
**from** *assms* **show** *FGModuleHom G (·) V smult″ (S ∘ T)*
**using** *FGModuleIso.axioms*(*1*) *ImG FGModHom-composite-left* **by** *fast*
**show** *FGModuleIso-axioms V (S ∘ T) X*
**using** *bijective FGModuleIso.bijective*[*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*
*⟹ FGModule.isomorphic G smult V smult″ X*
**using** *FGModIso-composite-left* **by** *fast*

**lemma** *isomorphic-to-zero-left* : *V = 0 ⟹ W = 0*
**using** *bijective bij-betw-imp-surj-on im-zero* **by** *fastforce*

**lemma** *isomorphic-to-zero-right* : *W = 0 ⟹ V = 0*
**using** *isomorphic-sym FGModuleIso.isomorphic-to-zero-left* **by** *fast*

**lemma** *isomorphic-to-irr-right′* :
**assumes** *⋀U. FGModule.GSubspace G smult′ W U ⟹ U = 0 ∨ U = W*
**shows** *⋀U. GSubspace U ⟹ U = 0 ∨ U = V*
**proof**−
**fix** *U* **assume** *U*: *GSubspace U*
**have** *U ≠ V ⟹ U = 0*
**proof**−
**assume** *UV*: *U ≠ V*
**from** *U bijective* **have** *T ‘ U = T ‘ V ⟹ U = V*
**using** *bij-betw-imp-inj-on*[*of T V W*] *inj-onD*[*of T V*] **by** *fast*
**with** *UV bijective* **have** *T ‘ U ≠ 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 ∨ U = V* **by** *fast*
**qed**

**end**

**context** *FGModule*
**begin**

**lemma** *isomorphic-sym* :
*isomorphic smult' W* $\Longrightarrow$ *FGModule.isomorphic G smult' W smult V*
**using** *FGModuleIso.inv* **by** *fast*

**lemma** *isomorphic-trans* :
*isomorphic smult' W* $\Longrightarrow$ *FGModule.isomorphic G smult' W smult'' X*
$\Longrightarrow$ *isomorphic smult'' X*
**using** *FGModuleIso.FGModIso-composite-left* **by** *fast*

**lemma** *isomorphic-to-zero-left* : *V = 0* $\Longrightarrow$ *isomorphic smult' W* $\Longrightarrow$ *W = 0*
**using** *FGModuleIso.isomorphic-to-zero-left* **by** *fast*

**lemma** *isomorphic-to-zero-right* : *isomorphic smult' 0* $\Longrightarrow$ *V = 0*
**using** *FGModuleIso.isomorphic-to-zero-right* **by** *fast*

**lemma** *FGModIso-idhom* : *FGModuleIso G smult V smult* (*id*$\downarrow$*V*) *V*
**using** *FGModHom-idhom*
**proof** (*rule FGModuleIso.intro*)
**show** *FGModuleIso-axioms V* (*id*$\downarrow$*V*) *V*
**using** *bij-betw-id bij-betw-restrict0* **by** *unfold-locales fast*
**qed**

**lemma** *isomorphic-refl* : *isomorphic smult V* **using** *FGModIso-idhom* **by** *fast*

**end**

### 5.4.5 Hom-sets

**definition** *FGModuleHomSet* ::
*'g*::*group-add set* $\Rightarrow$ ((*'f*::*field*,*'g*) *aezfun* $\Rightarrow$ *'v*::*ab-group-add* $\Rightarrow$ *'v*) $\Rightarrow$ *'v set*
$\Rightarrow$ ((*'f*,*'g*) *aezfun* $\Rightarrow$ *'w*::*ab-group-add* $\Rightarrow$ *'w*) $\Rightarrow$ *'w set*
$\Rightarrow$ (*'v* $\Rightarrow$ *'w*) *set*
**where** *FGModuleHomSet G fgsmult V fgsmult' W*
$\equiv$ {*T*. *FGModuleHom G fgsmult V fgsmult' T*} $\cap$ {*T*. *T* ‘ *V* $\subseteq$ *W*}

**lemma** *FGModuleHomSetI* :
*FGModuleHom G fgsmult V fgsmult' T* $\Longrightarrow$ *T* ‘ *V* $\subseteq$ *W*
$\Longrightarrow$ *T* $\in$ *FGModuleHomSet G fgsmult V fgsmult' W*
**unfolding** *FGModuleHomSet-def* **by** *fast*

**lemma** *FGModuleHomSetD-FGModuleHom* :
*T* $\in$ *FGModuleHomSet G fgsmult V fgsmult' W*
$\Longrightarrow$ *FGModuleHom G fgsmult V fgsmult' T*
**unfolding** *FGModuleHomSet-def* **by** *fast*

**lemma** *FGModuleHomSetD-Im* :
*T* $\in$ *FGModuleHomSet G fgsmult V fgsmult' W* $\Longrightarrow$ *T* ‘ *V* $\subseteq$ *W*
**unfolding** *FGModuleHomSet-def* **by** *fast*

**context** *FGModule*
**begin**

**lemma** *FGModuleHomSet-is-Gmaps-in-VectorSpaceHomSet* :
**fixes** *smult′* :: (*′f*, *′g*) *aezfun* ⇒ *′w*::*ab-group-add* ⇒ *′w* (**infixr** ⋆ *70*)
**and** *fsmult′* :: *′f* ⇒ *′w* ⇒ *′w* (**infixr** ♯⋆ *70*)
**and** *Gmult′* :: *′g* ⇒ *′w* ⇒ *′w* (**infixr** ∗⋆ *70*)
**defines** *fsmult′* : *fsmult′* ≡ *aezfun-scalar-mult.fsmult smult′*
**and** *Gmult′* : *Gmult′* ≡ *aezfun-scalar-mult.Gmult smult′*
**assumes** *FGModW* : *FGModule G smult′ W*
**shows** *FGModuleHomSet G smult V smult′ W*
= (*VectorSpaceHomSet fsmult V fsmult′ W*)
∩ {*T*. ∀ *g*∈*G*. ∀ *v*∈*V*. *T* (*g* ∗· *v*) = *g* ∗⋆ (*T v*)}
**proof**
**from** *fsmult′ Gmult′*
**show** *FGModuleHomSet G smult V smult′ W*
⊆ (*VectorSpaceHomSet fsmult V fsmult′ W*)
∩ {*T*. ∀ *g*∈*G*. ∀ *v*∈*V*. *T* (*g* ∗· *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*
⊇ (*VectorSpaceHomSet fsmult V fsmult′ W*)
∩ {*T*. ∀ *g*∈*G*. ∀ *v*∈*V*. *T* (*g* ∗· *v*) = *g* ∗⋆ *T v*}
**proof**
**fix** *T*
**assume** *T*: *T* ∈ (*VectorSpaceHomSet fsmult V fsmult′ W*)
∩ {*T*. ∀ *g*∈*G*. ∀ *v*∈*V*. *T* (*g* ∗· *v*) = *g* ∗⋆ *T v*}
**show** *T* ∈ *FGModuleHomSet G smult V smult′ W*
**proof** (*rule FGModuleHomSetI*, *rule VecHom-GMap-is-FGModuleHom*)
**from** *T fsmult′*
**show** *VectorSpaceHom* (♯·) *V* (*aezfun-scalar-mult.fsmult smult′*) *T*
**using** *VectorSpaceHomSetD-VectorSpaceHom*
**by** *fast*
**from** *T* **show** *T* ‘ *V* ⊆ *W* **using** *VectorSpaceHomSetD-Im* **by** *fast*
**from** *T Gmult′*
**show** ⋀*g v*. *g* ∈ *G* ⟹ *v* ∈ *V*
⟹ *T* (*g* ∗· *v*) = *aezfun-scalar-mult.Gmult* (⋆) *g* (*T v*)
**by** *fast*
**from** *T* **show** *T* ‘ *V* ⊆ *W* **using** *VectorSpaceHomSetD-Im* **by** *fast*
**qed** (*rule FGModW*)
**qed**
**qed**

**lemma** *Group-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)
  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* :: $'f \Rightarrow ('v \Rightarrow 'w) \Rightarrow ('v \Rightarrow 'w)$ (**infixr** $\sharp\star\cdot$ *70*)
**defines** *fsmult'* : *fsmult'* $\equiv$ *aezfun-scalar-mult.fsmult smult'*
**and** *Gmult'* : *Gmult'* $\equiv$ *aezfun-scalar-mult.Gmult smult'*
**defines** *hom-fsmult* : *hom-fsmult* $\equiv \lambda a\ T\ v.\ a\ \sharp\star\ T\ v$
**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* ($\sharp\star\cdot$) (*VectorSpaceHomSet* ($\sharp\cdot$) *V* ($\sharp\star$) *W*)
**using** *FGModule.fVectorSpace*[*OF FGModW*]
*VectorSpace.VectorSpace-VectorSpaceHomSet*[*OF fVectorSpace*]
**by** *fast*
**from** *fsmult' Gmult' FGModW*
**show** *Group* (*FGModuleHomSet G* ($\cdot$) *V* ($\star$) *W*)
$\wedge$ *FGModuleHomSet G* ($\cdot$) *V* ($\star$) *W*
$\subseteq$ *VectorSpaceHomSet* ($\sharp\cdot$) *V* ($\sharp\star$) *W*
**using** *Group-FGModuleHomSet FGModuleHomSet-is-Gmaps-in-VectorSpaceHomSet*
**by** *fast*
**next**
**fix** *a T* **assume** *T*: *T* $\in$ *FGModuleHomSet G* ($\cdot$) *V* ($\star$) *W*
**from** *hom-fsmult fsmult'* **have** *FGModuleHom G smult V smult'* (*a* $\sharp\star\cdot$ *T*)
**using** *FGModuleHomSetD-FGModuleHom*[*OF T*]
*FGModuleHomSetD-Im*[*OF T*]
*FGModuleHom.FGModuleHom-fscalar-mul*
**by** *simp*
**moreover from** *hom-fsmult fsmult'* **have** (*a* $\sharp\star\cdot$ *T*) ‘ *V* $\subseteq$ *W*
**using** *FGModuleHomSetD-Im*[*OF T*] *FGModule.fsmult-closed*[*OF FGModW*]
**by** *auto*
**ultimately show** *a* $\sharp\star\cdot$ *T* $\in$ *FGModuleHomSet G* ($\cdot$) *V* ($\star$) *W*
**using** *FGModuleHomSetI* **by** *fastforce*
**qed**

**lemma** *VectorSpace-FGModuleHomSet* :
**fixes** *smult'* :: ($'f$, $'g$) *aezfun* $\Rightarrow$ $'w$::*ab-group-add* $\Rightarrow$ $'w$ (**infixr** $\star$ *70*)
**and** *fsmult'* :: $'f \Rightarrow 'w \Rightarrow 'w$ (**infixr** $\sharp\star$ *70*)
**and** *hom-fsmult* :: $'f \Rightarrow ('v \Rightarrow 'w) \Rightarrow ('v \Rightarrow 'w)$ (**infixr** $\sharp\star\cdot$ *70*)
**defines** *fsmult'* $\equiv$ *aezfun-scalar-mult.fsmult smult'*
**defines** *hom-fsmult* $\equiv \lambda a\ T\ v.\ a\ \sharp\star\ T\ v$
**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* ⇒ $'m$::*monoid-add set* ⇒ ($'a$ ⇒ $'m$) *set*
  **where** *addfunset A M* ≡ {*f*. *supp f* ⊆ *A* ∧ *range f* ⊆ *M*
    ∧ (∀ *x*∈*A*. ∀ *y*∈*A*. *f* (*x*+*y*) = *f x* + *f y*) }

**lemma** *addfunsetI* :
  ⟦ *supp f* ⊆ *A*; *range f* ⊆ *M*; ∀ *x*∈*A*. ∀ *y*∈*A*. *f* (*x*+*y*) = *f x* + *f y* ⟧
    ⟹ *f* ∈ *addfunset A M*
  **unfolding** *addfunset-def* **by** *fast*

**lemma** *addfunsetD-supp* : *f* ∈ *addfunset A M* ⟹ *supp f* ⊆ *A*
  **unfolding** *addfunset-def* **by** *fast*

**lemma** *addfunsetD-range* : *f* ∈ *addfunset A M* ⟹ *range f* ⊆ *M*
  **unfolding** *addfunset-def* **by** *fast*

**lemma** *addfunsetD-range'* : *f* ∈ *addfunset A M* ⟹ *f x* ∈ *M*
  **using** *addfunsetD-range* **by** *fast*

**lemma** *addfunsetD-add* :
  ⟦ *f* ∈ *addfunset A M*; *x*∈*A*; *y*∈*A* ⟧ ⟹ *f* (*x*+*y*) = *f x* + *f y*
  **unfolding** *addfunset-def* **by** *fast*

**lemma** *addfunset0* : *addfunset A* (*0*::$'m$::*monoid-add set*) = *0*
**proof**
  **show** *addfunset A 0* ⊆ *0* **using** *addfunsetD-range'* **by** *fastforce*
  **have** (*0*::$'a$⇒$'m$) ∈ *addfunset A 0*
    **using** *supp-zerofun-subset-any* **by** (*rule addfunsetI*) *auto*
  **thus** *addfunset A* (*0*::$'m$::*monoid-add set*) ⊇ *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** *addfunset R M* ≠ {}
    **using** *addfunsetI*[*of 0 R M*] *supp-zerofun-subset-any Group.zero-closed*
    **by** *fastforce*
**next**
  **fix** *g h* **assume** *gh*: *g* ∈ *addfunset R M h* ∈ *addfunset R M*
  **show** *g* − *h* ∈ *addfunset R M*
  **proof** (*rule addfunsetI*)
    **from** *gh* **show** *supp* (*g* − *h*) ⊆ *R*
      **using** *addfunsetD-supp supp-diff-subset-union-supp* **by** *fast*
    **from** *gh* **show** *range* (*g* − *h*) ⊆ *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* :: $'g\ set \Rightarrow ('r,'g)\ aezfun\ set \Rightarrow (('r,'g)\ aezfun \Rightarrow 'v)\ set$
**where** *smultfunset G FH* $\equiv \{f.\ (\forall a{::}'r.\ \forall g{\in}G.\ \forall x{\in}FH.$
$f\ (\ a\ \delta\delta\ g * x\ ) = (a\ \delta\delta\ g) \cdot (f\ x))\}$

**lemma** *smultfunsetD* :
$[\![f \in smultfunset\ G\ FH;\ g{\in}G;\ x{\in}FH\ ]\!] \Longrightarrow f\ (\ a\ \delta\delta\ g * x\ ) = (a\ \delta\delta\ g) \cdot (f\ x)$
**unfolding** *smultfunset-def* **by** *fast*

**lemma** *smultfunsetI* :
$\forall a{::}'r.\ \forall g{\in}G.\ \forall x{\in}FH.\ f\ (\ a\ \delta\delta\ g * x\ ) = (a\ \delta\delta\ g) \cdot (f\ x)$
$\Longrightarrow f \in smultfunset\ G\ FH$
**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* ::
$'g\ set \Rightarrow ('r,'g)\ aezfun\ set \Rightarrow 'v\ set \Rightarrow (('r,'g)\ aezfun \Rightarrow 'v)\ set$
**where** *indspace G FH V* = *addfunset FH V* $\cap$ *smultfunset G FH*

**lemma** *indspaceD* :
$f \in indspace\ G\ FH\ V \Longrightarrow f \in addfunset\ FH\ V \cap smultfunset\ G\ FH$
**using** *indspace-def* **by** *fast*

**lemma** *indspaceD-supp* : $f \in indspace\ G\ FH\ V \Longrightarrow supp\ \ f \subseteq FH$
**using** *indspace-def addfunsetD-supp* **by** *fast*

**lemma** *indspaceD-supp'* : $f \in indspace\ G\ FH\ V \Longrightarrow x \notin FH \Longrightarrow f\ x = 0$
**using** *indspaceD-supp suppI-contra* **by** *fast*

**lemma** *indspaceD-range* : $f \in indspace\ G\ FH\ V \Longrightarrow range\ f \subseteq V$
**using** *indspace-def addfunsetD-range* **by** *fast*

**lemma** *indspaceD-range'*: $f \in indspace\ G\ FH\ V \Longrightarrow f\ x \in V$
**using** *indspaceD-range* **by** *fast*

**lemma** *indspaceD-add* :
$[\![ f \in indspace\ G\ FH\ V;\ x{\in}FH;\ y{\in}FH\ ]\!] \Longrightarrow f\ (x{+}y) = f\ x + f\ y$
**using** *indspace-def addfunsetD-add* **by** *auto*

**lemma** *indspaceD-transform* :
$[\![ f \in indspace\ G\ FH\ V;\ g{\in}G;\ x{\in}FH\ ]\!] \Longrightarrow f\ (\ a\ \delta\delta\ g * x\ ) = (a\ \delta\delta\ g) \cdot (f\ x)$
**using** *indspace-def smultfunsetD* **by** *auto*

**lemma** *indspaceI* :
$f \in addfunset\ FH\ V \Longrightarrow f \in smultfunset\ G\ FH \Longrightarrow f \in indspace\ G\ FH\ V$
**using** *indspace-def* **by** *fast*

**lemma** *indspaceI'* :
$[\![ supp\ f \subseteq FH;\ range\ f \subseteq V;\ \forall\, x{\in}FH.\ \forall\, y{\in}FH.\ f\ (x{+}y) = f\ x + f\ y;$
$\forall\, a{::}'r.\ \forall\, g{\in}G.\ \forall\, x{\in}FH.\ f\ (\ a\ \delta\delta\ g * x\ ) = (a\ \delta\delta\ g) \cdot (f\ x)\ ]\!]$
$\Longrightarrow f \in indspace\ G\ FH\ V$
**using** *smultfunsetI addfunsetI*[*of f*] *indspaceI* **by** *simp*

**lemma** *mono-indspace* : *mono* (*indspace G FH*)
**proof** (*rule monoI*)
**fix** *U V* :: *'v set* **assume** *U-V*: $U \subseteq V$
**show** *indspace G FH U* $\subseteq$ *indspace G FH V*
**proof**
**fix** *f* **assume** *f*: $f \in indspace\ G\ FH\ U$
**show** $f \in indspace\ G\ FH\ V$ **using** *indspaceD-supp*[*OF f*]
**proof** (*rule indspaceI'*)
**from** *f U-V* **show** *range f* $\subseteq$ *V* **using** *indspaceD-range*[*of f G FH*] **by** *auto*
**from** *f* **show** $\forall\, x{\in}FH.\ \forall\, y{\in}FH.\ f\ (x{+}y) = f\ x + f\ y$
**using** *indspaceD-add* **by** *auto*
**from** *f* **show** $\forall\, a{::}'r.\ \forall\, g{\in}G.\ \forall\, x{\in}FH.\ f\ (\ a\ \delta\delta\ g * x\ ) = (a\ \delta\delta\ g) \cdot (f\ x)$
**using** *indspaceD-transform* **by** *auto*
**qed**
**qed**
**qed**

**end**

**context** *FGModule*
**begin**

**lemma** *zero-transforms* : $0 \in smultfunset\ G\ FH$
**using** *smultfunsetI FG-fddg-closed smult-zero* **by** *simp*

**lemma** *indspace0* : $indspace\ G\ FH\ 0 = 0$
**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* ⊆ *V* **by** *simp*
**with** *mono-indspace* [*of G FH*]
**have** *indspace G FH 0* ⊆ *indspace G FH V*
**by** (*auto dest*!: *monoD* [*of - 0 V*])
**then show** *indspace G FH V* ≠ {}
**using** *indspace0* [*of FH*] **by** *auto*
**next**
**fix** *f1 f2* **assume** *ff*: *f1* ∈ *indspace G FH V f2* ∈ *indspace G FH V*
**hence** *f1* − *f2* ∈ *addfunset FH V*
**using** *assms indspaceD indspaceD Group Group-addfunset Group.diff-closed*
**by** *fast*
**moreover from** *ff* **have** *f1* − *f2* ∈ *smultfunset G FH*
**using** *indspaceD-transform FG-fddg-closed indspaceD-range*′ *smult-distrib-left-diff*
*smultfunsetI*
**by** *simp*
**ultimately show** *f1* − *f2* ∈ *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* ⇒ (′*r* ⇒ ′*m*::*ab-group-add*) ⇒ (′*r* ⇒ ′*m*) (**infixr** ¤ *70*)
**where** *rightreg-scalar-mult r f* = (λ*x*. *if x* ∈ *R then f* (*x*∗*r*) *else 0*)

**lemma** *rightreg-scalar-multD1* : *x* ∈ *R* ⟹ (*r* ¤ *f*) *x* = *f* (*x*∗*r*)
**unfolding** *rightreg-scalar-mult-def* **by** *simp*

**lemma** *rightreg-scalar-multD2* : *x* ∉ *R* ⟹ (*r* ¤ *f*) *x* = *0*
**unfolding** *rightreg-scalar-mult-def* **by** *simp*

**lemma** *rrsmult-supp* : *supp* (*r* ¤ *f*) ⊆ *R*
**using** *rightreg-scalar-multD2 suppD-contra* **by** *force*

**lemma** *rrsmult-range* : *range* (*r* ¤ *f*) ⊆ {*0*} ∪ *range f*
**proof** (*rule image-subsetI*)
**fix** *x* **show** (*r* ¤ *f*) *x* ∈ {*0*} ∪ *range f*
**using** *rightreg-scalar-multD1*[*of x r f*] *image-eqI*
*rightreg-scalar-multD2*[*of x r f*]
**by** (*cases x* ∈ *R*) *auto*
**qed**

**lemma** *rrsmult-distrib-left* : $r \mathbin{¤} (f + g) = r \mathbin{¤} f + r \mathbin{¤} g$
**proof**
  **fix** $x$ **show** $(r \mathbin{¤} (f + g))\ x = (r \mathbin{¤} f + r \mathbin{¤} g)\ x$
    **unfolding** *rightreg-scalar-mult-def* **by** (*cases* $x \in R$) *auto*
**qed**

**lemma** *rrsmult-distrib-right* :
  **assumes** $\bigwedge x\ y.\ x \in R \Longrightarrow y \in R \Longrightarrow f\ (x+y) = f\ x + f\ y\ r \in R\ s \in R$
  **shows** $(r + s) \mathbin{¤} f = r \mathbin{¤} f + s \mathbin{¤} f$
**proof**
  **fix** $x$ **show** $((r + s) \mathbin{¤} f)\ x = (r \mathbin{¤} f + s \mathbin{¤} f)\ x$
    **using** *assms mult-closed*
    **unfolding** *rightreg-scalar-mult-def*
    **by** (*cases* $x \in R$) (*auto simp add*: *distrib-left*)
**qed**

**lemma** *RModule-addfunset* :
  **fixes** $M::'g::ab\text{-}group\text{-}add\ set$
  **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* (¤) (*addfunset R M*)
  **proof**
    **fix** $r\ f$ **assume** $r$: $r \in R$ **and** $f$: $f \in addfunset\ R\ M$
    **show** $r \mathbin{¤} f \in addfunset\ R\ M$
    **proof** (*rule addfunsetI*)
      **show** $supp\ (r \mathbin{¤} f) \subseteq R$
        **using** *rightreg-scalar-multD2 suppD-contra* **by** *force*
      **show** $range\ (r \mathbin{¤} f) \subseteq M$
        **using** *addfunsetD-range*[*OF f*] *Group.zero-closed*[*OF assms*]
        **unfolding** *rightreg-scalar-mult-def*
        **by** *fastforce*
      **from** $r$ **show** $\forall x{\in}R.\ \forall y{\in}R.\ (r \mathbin{¤} f)\ (x + y) = (r \mathbin{¤} f)\ x + (r \mathbin{¤} f)\ y$
        **using** *mult-closed add-closed addfunsetD-add*[*OF f*]
        **unfolding** *rightreg-scalar-mult-def*
        **by** (*simp add*: *distrib-right*)
    **qed**
  **next**
    **show** $\bigwedge r\ f\ g.\ r \mathbin{¤} (f + g) = r \mathbin{¤} f + r \mathbin{¤} g$ **using** *rrsmult-distrib-left* **by** *fast*
  **next**
    **fix** $r\ s\ f$ **assume** $r \in R\ s \in R\ f \in addfunset\ R\ M$
    **thus** $(r + s) \mathbin{¤} f = r \mathbin{¤} f + s \mathbin{¤} f$
      **using** *addfunsetD-add*[*of f*] *rrsmult-distrib-right*[*of f*] **by** *simp*
  **next**
    **fix** $r\ s\ f$ **assume** $r$: $r \in R$ **and** $s$: $s \in R$ **and** $f$: $f \in addfunset\ R\ M$

```
    show r ¤ s ¤ f = (r * s) ¤ f
    proof
      fix x from r show (r ¤ s ¤ f) x = ((r * s) ¤ f) x
        using mult-closed unfolding rightreg-scalar-mult-def
        by    (cases x ∈ R) (auto simp add: mult.assoc)
    qed
  next
    fix f assume f: f ∈ addfunset R M
    show 1 ¤ f = f
    proof
      fix x show (1 ¤ f) x = f x
        unfolding rightreg-scalar-mult-def
        using     addfunsetD-supp[OF f] suppI-contra[of x f]
                  contra-subsetD[of supp f]
        by        (cases x ∈ 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 *indV* will be shown to be a module over the group ring for $H$ under the right regular scalar multiplication *rrsmult*.

```
locale InducedFHModule = Supgroup?: Group H
+ BaseFGMod?    : FGModule G smult V
+ induced-smult?: aezfun-scalar-mult rrsmult
  for   H       :: 'g::group-add set
  and   G       :: 'g set
  and   FG      :: ('f::field, 'g) aezfun set
  and   smult   :: ('f, 'g) aezfun ⇒ '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)
+ fixes FH      :: ('f, 'g) aezfun set
  and   indV    :: (('f, 'g) aezfun ⇒ 'v) set
  defines FH    : FH ≡ Supgroup.group-ring
  and     indV  : indV ≡ BaseFGMod.indspace G FH V
  assumes rrsmult : rrsmult = Ring1.rightreg-scalar-mult FH
  and     Subgroup: Supgroup.Subgroup G
begin

abbreviation indfsmult ::
  'f ⇒ (('f, 'g) aezfun ⇒ 'v) ⇒ (('f, 'g) aezfun ⇒ 'v) (infixl ¤¤ 70)
  where indfsmult ≡ induced-smult.fsmult
abbreviation indflincomb ::
```

*'f list* ⇒ (('*f*, '*g*) *aezfun* ⇒ '*v*) *list* ⇒ (('*f*, '*g*) *aezfun* ⇒ '*v*) (**infixl** ·¤¤ *70*)
**where** *indflincomb* ≡ *induced-smult.flincomb*
**abbreviation** *Hmult* ::
'*g* ⇒ (('*f*, '*g*) *aezfun* ⇒ '*v*) ⇒ (('*f*, '*g*) *aezfun* ⇒ '*v*) (**infixl** ∗¤ *70*)
**where** *Hmult* ≡ *induced-smult.Gmult*

**lemma** *Ring1-FH* : *Ring1 FH* **using** *FH Supgroup.Ring1-RG* **by** *fast*

**lemma** *FG-subring-FH* : *Ring1.Subring1 FH BaseFGMod.FG*
**using** *FH Supgroup.Subgroup-imp-Subring*[*OF Subgroup*] **by** *fast*

**lemma** *rrsmultD1* : *x* ∈ *FH* ⟹ (*r* ¤ *f*) *x* = *f* (*x*∗*r*)
**using** *Ring1.rightreg-scalar-multD1*[*OF Ring1-FH*] *rrsmult* **by** *simp*

**lemma** *rrsmultD2* : *x* ∉ *FH* ⟹ (*r* ¤ *f*) *x* = *0*
**using** *Ring1.rightreg-scalar-multD2*[*OF Ring1-FH*] *rrsmult* **by** *fast*

**lemma** *rrsmult-supp* : *supp* (*r* ¤ *f*) ⊆ *FH*
**using** *Ring1.rrsmult-supp*[*OF Ring1-FH*] *rrsmult* **by** *auto*

**lemma** *rrsmult-range* : *range* (*r* ¤ *f*) ⊆ {*0*} ∪ *range f*
**using** *Ring1.rrsmult-range*[*OF Ring1-FH*] *rrsmult* **by** *auto*

**lemma** *FHModule-addfunset* : *FGModule H rrsmult* (*addfunset FH V*)
**proof** (*rule FGModule.intro*)
**from** *FH rrsmult* **show** *RModule Supgroup.group-ring* (¤) (*addfunset FH V*)
**using** *Group Supgroup.Ring1-RG Ring1.RModule-addfunset* **by** *fast*
**qed** (*unfold-locales*)

**lemma** *FHSubmodule-indspace* :
*FGModule.FGSubmodule H rrsmult* (*addfunset FH V*) *indV*
**proof** (*rule FGModule.FGSubmoduleI*[*of H*], *rule FHModule-addfunset*, *rule conjI*)
**from** *Ring1-FH indV* **show** *Group indV* **using** *Group-indspace* **by** *fast*
**from** *indV* **show** *indV* ⊆ *addfunset FH V*
**using** *BaseFGMod.indspaceD* **by** *fast*
**next**
**fix** *r f* **assume** *rf*: *r* ∈ (*Supgroup.group-ring* :: ('*f*,'*g*) *aezfun set*) *f* ∈ *indV*
**from** *rf*(*2*) *indV* **have** *rf2'*: *f* ∈ *BaseFGMod.indspace G FH V* **by** *fast*
**show** *r* ¤ *f* ∈ *indV*
**unfolding** *indV*
**proof** (*rule BaseFGMod.indspaceI'*, *rule rrsmult-supp*)
**show** *range* (*r* ¤ *f*) ⊆ *V*
**using** *rrsmult-range BaseFGMod.indspaceD-range*[*OF rf2'*] *zero-closed*
**by** *force*
**from** *FH rf*(*1*) *rf2'*
**show** ∀ *x*∈*FH*. ∀ *y*∈*FH*. (*r* ¤ *f*) (*x* + *y*) = (*r* ¤ *f*) *x* + (*r* ¤ *f*) *y*
**using** *Ring1.add-closed*[*OF Ring1-FH*] *rrsmultD1*[*of - r f*]
*Ring1.mult-closed*[*OF Ring1-FH*] *BaseFGMod.indspaceD-add*
**by** (*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 (λy. 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 λy. (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-setspan-proj-in-setspan[of G y]
```

```
        smult-closed ActingGroup.group-ringD
    by  auto
qed

{
  fix x y assume xy: x ∈ FH y ∈ FH
  with v have (induced-vector v) (x + y)
                = (induced-vector v) x + (induced-vector v) y
   using Ring1-FH Ring1.add-closed aezfun-setspan-proj-add[of G x y] FG-proj-in-FG
         smult-distrib-left induced-vector-def
    by  auto
}
thus ∀ x∈FH. ∀ y∈FH. induced-vector v (x + y)
          = induced-vector v x + induced-vector v y
  by fast

{
  fix a g x assume g: g ∈ G and x: x ∈ FH
  with v FH
    have (induced-vector v) (a δδ g ∗ x) = a δδ g · (induced-vector v) x
    using FG-subring-FH FG-fddg-closed Ring1-FH
          Ring1.mult-closed[of FH a δδ g] FG-proj-mult-leftdelta[of g a]
          FG-fddg-closed FG-proj-in-FG smult-assoc induced-vector-def
    by  fastforce
}
thus ∀ a. ∀ g∈G. ∀ x∈FH. induced-vector v (a δδ g ∗ x)
          = a δδ g · induced-vector v x
  by fast

qed

lemma induced-vector-additive :
  v ∈ V ⟹ v′ ∈ V
      ⟹ induced-vector (v+v′) = induced-vector v + induced-vector v′
  using add-closed induced-vector-def FG-proj-in-FG smult-distrib-left by auto

lemma hom-induced-vector : FGModuleHom G smult V rrsmult induced-vector
proof

  show ⋀v v′. v ∈ V ⟹ v′ ∈ V
          ⟹ induced-vector (v + v′) = induced-vector v + induced-vector v′
    using induced-vector-additive by fast

  have induced-vector = (λv. if v∈V then λy. if y ∈ FH
          then (FG-proj y) · v else 0 else 0)
    using induced-vector-def by fast
  thus supp induced-vector ⊆ V using supp-restrict0[of V] by fastforce

  show ⋀x v. x ∈ BaseFGMod.FG ⟹ v ∈ V
```

$\Longrightarrow$ *induced-vector* $(x \cdot v) = x \natural$ *induced-vector* $v$
**proof** −
**fix** $x$ $v$ **assume** *xv*: $x \in$ *BaseFGMod.FG* $v \in V$
**show** *induced-vector* $(x \cdot v) = x \natural$ *induced-vector* $v$
**proof**
**fix** $a$
**from** *xv FH* **show** *induced-vector* $(x \cdot v)$ $a = (x \natural$ *induced-vector* $v)$ $a$
**using** *smult-closed induced-vector-def FG-proj-in-FG smult-assoc rrsmultD1 FG-subring-FH Ring1.mult-closed*[*OF Ring1-FH*] *induced-vector-apply1 FG-proj-mult-right*[*of x*] *smult-closed induced-vector-apply2 rrsmultD2*
**by** *auto*
**qed**
**qed**

**qed**

**lemma** *indspace-sum-list-fddh*:
⟦ *fhs* $\neq$ []; *set* (*map snd fhs*) $\subseteq H$; $f \in$ *indV* ⟧
$\Longrightarrow f\ (\sum (a,h) \leftarrow$ *fhs*. $a\ \delta\delta\ h) = (\sum (a,h) \leftarrow$ *fhs*. $f\ (a\ \delta\delta\ h))$
**proof** (*induct fhs rule*: *list-nonempty-induct*)
**case** (*single fh*) **show** *?case*
**using** *split-beta*[*of* $\lambda a\ h.\ a\ \delta\delta\ h$ *fh*] *split-beta*[*of* $\lambda a\ h.\ f\ (a\ \delta\delta\ h)$ *fh*] **by** *simp*
**next**
**case** (*cons fh fhs*)
**hence** *prevcase*: *snd fh* $\in H$ *set* (*map snd fhs*) $\subseteq H$ $f \in$ *indV*
$f\ (\sum (a,h) \leftarrow$ *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*. $a\ \delta\delta\ h)$
$= f\ ((fst\ fh)\ \delta\delta\ (snd\ fh) + (\sum ah \leftarrow$ *fhs*. *case-prod* $(\lambda a\ h.\ a\ \delta\delta\ h)\ ah))$
**using** *split-beta*[*of* $\lambda a\ h.\ a\ \delta\delta\ h$ *fh*] **by** *simp*
**moreover from** *prevcase*(*1*) *FH* **have** (*fst fh*) $\delta\delta$ (*snd fh*) $\in$ *FH*
**using** *Supgroup.RG-aezdeltafun-closed* **by** *fast*
**moreover from** *prevcase*(*2*) *FH*
**have** $(\sum ah \leftarrow$ *fhs*. *case-prod* $(\lambda a\ h.\ a\ \delta\delta\ h)\ ah) \in$ *FH*
**using** *Supgroup.RG-aezdeltafun-closed*
*Ring1.sum-list-closed*[*OF Ring1-FH*, *of* $\lambda ah.$ *case-prod* $(\lambda a\ h.\ a\ \delta\delta\ h)\ ah$ *fhs*]
**by** *fastforce*
**ultimately have** $f\ (\sum (a,h) \leftarrow$ *fh*#*fhs*. $a\ \delta\delta\ h)$
$= f\ ((fst\ fh)\ \delta\delta\ (snd\ fh)) + f\ (\sum (a,h) \leftarrow$ *fhs*. $a\ \delta\delta\ h)$
**using** *indV prevcase*(*3*) *BaseFGMod.indspaceD-add* **by** *simp*
**with** *prevcase*(*4*) **show** *?case* **using** *split-beta*[*of* $\lambda a\ h.\ f\ (a\ \delta\delta\ h)$ *fh*] **by** *simp*
**qed**

**lemma** *induced-fsmult-conv-fsmult-1ddh* :
$f \in$ *indV* $\Longrightarrow h \in H \Longrightarrow (r \natural\natural f)\ (1\ \delta\delta\ h) = r \sharp\cdot (f\ (1\ \delta\delta\ h))$
**using** *FH indV induced-smult.fsmultD Supgroup.RG-aezdeltafun-closed*[*of h 1*::*'f*]
*rrsmultD1 aezdeltafun-decomp'*[*of r h*]
*aezdeltafun-decomp*[*of r h*] *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 HmodG ⊆ H H = (⋃h∈HmodG. G + {h}) f ∈ indV f' ∈ indV
      ∀h∈HmodG. f (1 δδ h) = f' (1 δδ h) h ∈ H
  shows  f (r δδ h) = f' (r δδ h)
proof−
  from assms(2,6) obtain h' where h': h' ∈ HmodG h ∈ G + {h'} by fast
  from h'(2) obtain g where g: g ∈ G h = g + h'
    using set-plus-def[of G] by auto
  from g(2) have r δδ h = r δδ 0 ∗ (1 δδ (g+h'))
    using aezdeltafun-decomp by simp
  moreover have (1::'f) δδ (g+h') = 1 δδ g ∗ (1 δδ h')
    using times-aezdeltafun-aezdeltafun[of 1::'f, THEN sym] by simp
  ultimately have r δδ h = r δδ g ∗ (1 δδ h')
    using aezdeltafun-decomp[of r g]
    by  (simp add: algebra-simps)
  with indV FH assms(1,3,4) g(1) h'(1)
    have f (r δδ h) = r δδ g · f (1 δδ h') f' (r δδ h) = r δδ g · f' (1 δδ 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 δδ h) = f' (r δδ h) using h'(1) assms(5) by simp
qed

lemma indspace-el-eq :
  assumes HmodG ⊆ H H = (⋃h∈HmodG. G + {h}) f ∈ indV f' ∈ indV
      ∀h∈HmodG. f (1 δδ h) = f' (1 δδ h)
  shows  f = f'
proof
  fix x show f x = f' x
  proof (cases x = 0 x ∈ FH rule: conjcases)
    case BothTrue
    hence x = 0 δδ 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: set (map snd rhs) ⊆ H x = (∑(r,h)←rhs. r δδ 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 = (∑ (r,h)←rhs. f (r δδ h)) f' x = (∑ (r,h)←rhs. f' (r δδ h))
      using indspace-sum-list-fddh
      by    auto
    moreover from rhs(1) assms have ∀ (r,h) ∈ set rhs. f (r δδ h) = f' (r δδ 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 λr h. f (r δδ h)] by simp
  next
    case BothFalse
    with indV assms(3,4) show ?thesis
      using BaseFGMod.indspaceD-supp'[of f] BaseFGMod.indspaceD-supp'[of f']
      by    simp
  qed
qed

lemma indspace-el-eq' :
  assumes set hs ⊆ H H = (⋃ h∈set hs. G + {h}) f ∈ indV f' ∈ indV
        ∀ i<length hs. f (1 δδ (hs!i)) = f' (1 δδ (hs!i))
  shows   f = f'
  using assms(1−4)
proof (rule indspace-el-eq[of set hs])
  have ⋀h. h∈set hs ⟹ f (1 δδ h) = f' (1 δδ h)
  proof−
    fix h assume h ∈ set hs
    from this obtain i where i < length hs h = hs!i
      using in-set-conv-nth[of h] by fast
    with assms(5) show f (1 δδ h) = f' (1 δδ h) by simp
  qed
  thus ∀ h∈set hs. f (1 δδ h) = f' (1 δδ 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 ⇒ 'v::ab-group-add ⇒ 'v (infixl · 70)
```

```
  and V     :: 'v set
+
  assumes good-char: of-nat (card G) ≠ (0::'f)
  and     findim   : fscalar-mult.findim fsmult V

lemma (in Group) trivial-FinGroupRep :
  fixes   smult     :: ('f::field, 'g) aezfun ⇒ 'v::ab-group-add ⇒ 'v
  assumes good-char  : of-nat (card G) ≠ (0::'f)
  and     smult-zero : ∀ a∈group-ring. smult a (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 ≡ of-nat (card G)
abbreviation GRepHom ≡ FGModuleHom G smult V
abbreviation GRepIso ≡ FGModuleIso G smult V
abbreviation GRepEnd ≡ 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* ⟹ *U* = *0* ∨ *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* ∈ *V* ⟹ *v* ≠ *0* ⟹ *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* ⇒ *′v*::*ab-group-add* ⇒ *′v*
**assumes** *of-nat* (*card G*) ≠ (*0*::*′f*)

**and** $\forall a{\in}$*group-ring. smult a* $(0::'v) = 0$
**shows** *IrrFinGroupRepresentation G smult* $(0::'v\ set)$
**proof** (*rule IrrFinGroupRepresentation.intro*)
**from** *assms* **show** *FinGroupRepresentation G smult 0*
**using** *trivial-FinGroupRep* **by** *fast*
**show** *IrrFinGroupRepresentation-axioms G smult 0*
**using** *RModule.zero-closed* **by** *unfold-locales auto*
**qed**

**lemma** (**in** *Group*) *trivial-IrrFinGroupRepresentation-in-FG* :
*of-nat* (*card G*) $\neq$ $(0::'f::field)$
$\Longrightarrow$ *IrrFinGroupRepresentation G* $(*)$ $(0::('f,'g)\ aezfun\ set)$
**using** *trivial-IrrFinGroupRepI*[*of* $(*)$] **by** *simp*

**context** *FinGroupRepresentation*
**begin**

**lemma** *IrrFinGroupRep-trivialGSubspace* :
*IrrFinGroupRepresentation G smult* $(0::'v\ set)$
**proof**−
**have** *ordG* $\neq$ $(0::'f)$ **using** *good-char* **by** *fast*
**moreover have** $\forall a{\in}FG.\ a \cdot 0 = 0$ **using** *smult-zero* **by** *simp*
**ultimately show** *?thesis*
**using** *ActingGroup.trivial-IrrFinGroupRepI*[*of smult*] **by** *fast*
**qed**

**lemma** *IrrI* :
**assumes** $\bigwedge U.$ *FGModule.GSubspace G smult V U* $\Longrightarrow$ $U = 0 \vee U = V$
**shows** *IrrFinGroupRepresentation G smult V*
**using** *assms IrrFinGroupRepresentation.intro* **by** *unfold-locales*

**lemma** *notIrr* :
$\neg$ *IrrFinGroupRepresentation G smult V*
$\Longrightarrow$ $\exists U.$ *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** *FinGroupRepresentation*
**begin**

**lemma** *avg-proj-eq-id-on-right* :
**assumes** *VectorSpace fsmult W add-independentS* $[W,V]$ $v \in V$
**defines** $P : P \equiv (\bigoplus [W,V]{\downarrow}1)$
**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** $T\ v = v$
**proof**−
**from** *P assms(2,3)* **have** $\bigwedge g.\ g \in G \Longrightarrow P\ (g *\cdot v) = g *\cdot v$
**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** $\bigwedge g.\ g \in G \Longrightarrow CP\ g\ v = v$
**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 λx. 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** $\bigwedge g.\ g \in G \Longrightarrow P\ (g *\cdot v) \in U$
**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** $\bigwedge g.\ g \in G \Longrightarrow CP\ g\ v \in U$
**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 λg. 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* ⊕ *U*
**defines** *P* : *P* ≡ (⨁ [*W*,*U*]↓*1*)
**defines** *CP*: *CP* ≡ (λ*g*. *Gmult* (− *g*) ∘ *P* ∘ *Gmult g*)
**defines** *T* : *T* ≡ (*fsmult* (*1*/*ordG*) ∘ (∑ *g*∈*G*. *CP g*))↓*V*
**shows** *FGModuleEnd G smult V T*
**proof** (*rule VecEnd-GMap-is-FGModuleEnd*)

**from** *T* **have** *T-apply*: ⋀*v*. *v*∈*V* ⟹ *T v* = (*1*/*ordG*) ♯· (∑ *g*∈*G*. *CP g v*)
**using** *finiteG sum-fun-apply*[*of G CP*] **by** *simp*

**from** *assms*(*1*−*4*) *P* **have** *Pgv*: ⋀*g v*. *g* ∈ *G* ⟹ *v* ∈ *V* ⟹ *P* ( *g* ∗· *v* ) ∈ *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*: ⋀*v*. *v* ∈ *V* ⟹ (λ*g*. *CP g v*) ‘ *G* ⊆ *V*
**proof**−
**fix** *v* **assume** *v* ∈ *V* **thus** (λ*g*. *CP g v*) ‘ *G* ⊆ *V*
**using** *CP Pgv*[*of* - *v*] *ActingGroup.neg-closed Gmult-closed finiteG* **by** *auto*
**qed**

**have** *sumCP-V*: ⋀*v*. *v* ∈ *V* ⟹ (∑ *g*∈*G*. *CP g v*) ∈ *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* ∈ *V v*′ ∈ *V*
**with** *T-apply* **have** *T* (*v* + *v*′) = (*1*/*ordG*) ♯· (∑ *g*∈*G*. *CP g* (*v* + *v*′))
**using** *add-closed* **by** *fast*
**moreover have** ⋀*g*. *g* ∈ *G* ⟹ *CP g* (*v* + *v*′) = *CP g v* + *CP g v*′
**proof**−
**fix** *g* **assume** *g*: *g* ∈ *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 ⊆ V using supp-restrict0 by fast
 qed

 show ⋀a v. v ∈ V ⟹ T (a ♯· v) = a ♯· T v
 proof−
  fix a::′f and v assume v: v ∈ V
  with T-apply have T (a ♯· v) = (1/ordG) ♯· (∑ g∈G. CP g (a ♯· v))
    using fsmult-closed by fast
  moreover have ⋀g. g ∈ G ⟹ CP g (a ♯· v) = a ♯· CP g v
  proof−
   fix g assume g ∈ G
   with assms(1−4) CP P v show CP g (a ♯· v) = a ♯· 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 ♯· v) = a ♯· 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 ‘ V ⊆ V using sumCP-V fsmult-closed T-apply image-subsetI by auto

qed

show ⋀g v. g ∈ G ⟹ v ∈ V ⟹ T (g ∗· v) = g ∗· T v
proof−
 fix g v assume g: g ∈ G and v: v ∈ V
 with T-apply have T (g ∗· v) = (1/ordG) ♯· (∑ h∈G. CP h (g ∗· v))
   using Gmult-closed by fast
 with g have T (g ∗· v) = (1/ordG) ♯· (∑ h∈G. CP (h + − g) (g ∗· v))
   using GroupG Group.neg-closed
         Group.right-translate-sum[of G − g λh. CP h (g ∗· v)]
   by    fastforce
 moreover from CP
   have ⋀h. h ∈ G ⟹ CP (h + − g) (g ∗· v) = g ∗· 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 δδ g G]
          fsmult-Gmult-comm[of g ∑ h∈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 ⊕ U v ∈ V
 defines P : P ≡ (⨁ [W,U]↓1)
 defines CP: CP ≡ (λg. Gmult (− g) ∘ P ∘ Gmult g)
 defines T : T ≡ fsmult (1/ordG) ∘ (∑ g∈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   ∃ W. GSubspace W ∧ V = W ⊕ 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 ⊕ U using inner-dirsum-doubleI by fast
 moreover have GSubspace 0 using trivial-RSubmodule zero-closed by auto
 ultimately show ∃ W. GSubspace W ∧ V = W ⊕ U by fast
next
 case False
 from assms obtain W'
   where W': VectorSpace.Subspace fsmult V W' ∧ V = W' ⊕ U
  using GSubspace-is-Subspace FinDimVectorSpace FinDimVectorSpace.semisimple
   by    force
 hence vsp-W': VectorSpace fsmult W' and dirsum: V = W' ⊕ 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* = (⨁ [*W'*,*U*]↓*1*)
**define** *CP* **where** *CP* = (λ*g*. *Gmult* (− *g*) ∘ *P* ∘ *Gmult g*)
**define** *S* **where** *S* = *fsmult* (*1*/*ordG*) ∘ (∑ *g*∈*G*. *CP g*)
**define** *W* **where** *W* = *GroupHom.Ker V* (*S*↓*V*)
**from** *assms W' indS S-def CP-def P-def* **have** *endo*: *GRepEnd* (*S*↓*V*)
**using** *FGModuleEnd-avg-proj-right* **by** *fast*
**moreover from** *S-def CP-def P-def* **have** ⋀*v*. *v* ∈ *V* ⟹ (*S*↓*V*) ((*S*↓*V*) *v*) = (*S*↓*V*) *v*
**using** *endo FGModuleEnd.endomorph*
*avg-proj-is-proj-right*[*OF vsp-W' assms indS dirsum*]
**by** *fastforce*
**moreover have** (*S*↓*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*↓*V*) ‘ *V* = *S* ‘ *V* **by** *auto*
**ultimately show** *?thesis* **by** *fast*
**qed**
**ultimately have** *V* = *W* ⊕ *U GSubspace W*
**using** *W-def FGModuleEnd.proj-decomp*[*of G smult V S*↓*V*]
*FGModuleEnd.GSubspace-Ker*
**by** *auto*
**thus** *?thesis* **by** *fast*
**qed**

**corollary** *Maschke-proper* :
**assumes** *GSubspace U U* ≠ *0 U* ≠ *V*
**shows** ∃ *W*. *GSubspace W* ∧ *W* ≠ *0* ∧ *W* ≠ *V* ∧ *V* = *W* ⊕ *U*
**proof**−
**from** *assms*(*1*) **obtain** *W* **where** *W*: *GSubspace W V* = *W* ⊕ *U*
**using** *Maschke* **by** *fast*
**from** *assms*(*3*) *W*(*2*) **have** *W* ≠ *0* **using** *inner-dirsum-double-left0* **by** *fast*
**moreover from** *assms*(*1*,*2*) *W* **have** *W* ≠ *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* :
¬ *IrrFinGroupRepresentation G smult V*
⟹ ∃ *U W*. *GSubspace U* ∧ *U* ≠ *0* ∧ *U* ≠ *V* ∧ *GSubspace W* ∧ *W* ≠ *0*

```
        ∧ W ≠ V ∧ V = U ⊕ 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 *AbGroup-subset-inner-dirsum*.)

```
lemma FinGroupRepresentation-reducible' :
  fixes n::nat
  shows ⋀V. FinGroupRepresentation G fgsmult V
            ∧ n = FGModule.fdim fgsmult V
              ⟹ (∃ Us. Ball (set Us) (IrrFinGroupRepresentation G fgsmult)
                ∧ (0 ∉ set Us) ∧ V = (⨁ U←Us. U) )
proof (induct n rule: full-nat-induct)
  fix n V
  define GRep IGRep GSubspace GSpan fdim
    where GRep = FinGroupRepresentation G fgsmult
      and IGRep = IrrFinGroupRepresentation G fgsmult
      and GSubspace = FGModule.GSubspace G fgsmult V
      and GSpan = FGModule.GSpan G fgsmult
      and fdim = FGModule.fdim fgsmult
  assume ∀ m. Suc m ≤ n ⟶ (∀ x. GRep x ∧ m = fdim x ⟶ ( ∃ Us.
            Ball (set Us) IGRep ∧ (0 ∉ set Us) ∧ x = (⨁ U←Us. U)) )
  hence prev-case:
    ⋀m. m < n ⟹ (⋀W. GRep W ⟹ m = fdim W ⟹ ( ∃ Us.
          Ball (set Us) IGRep ∧ (0 ∉ set Us) ∧ W = (⨁ U←Us. U)))
    using Suc-le-eq by auto
  show GRep V ∧ n = fdim V ⟹ ( ∃ Us.
        Ball (set Us) IGRep ∧ (0 ∉ set Us) ∧ V = (⨁ U←Us. U) )
  proof−
    assume V: GRep V ∧ n = fdim V
    show (∃ Us. Ball (set Us) IGRep ∧ (0 ∉ set Us) ∧ V = (⨁ U←Us. U))
    proof (cases IGRep V V = 0 rule: conjcases)
      case BothTrue
      moreover have Ball (set []) IGRep ∀ U∈set []. U ≠ 0 by auto
      ultimately show ?thesis using inner-dirsum-Nil by fast
    next
      case OneTrue
      with V GRep-def obtain v where v: v ∈ V v ≠ 0
        using FinGroupRepresentation.Group[of G fgsmult] Group.obtain-nonzero
        by    auto
       from v(1) V GRep-def GSpan-def GSubspace-def have GSub: GSubspace
(GSpan [v])
        using FinGroupRepresentation.GSubmodule-GSpan-single by fast
      moreover from v V GRep-def GSpan-def have nzero: GSpan [v] ≠ 0
        using FinGroupRepresentation.GSpan-single-nonzero by fast
      ultimately have V = GSpan [v]
        using OneTrue GSpan-def GSubspace-def IGRep-def IrrFinGroupRepresen-
tation.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*) ((∈) *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* [(∑ *X*←*Us*. *X*),(∑ *X*←*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* ∈ (∑ *X*←*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* :
∃ *Us*. (∀ *U*∈*set Us*. *IrrFinGroupRepresentation G smult U*) ∧ (*0* ∉ *set Us*)
∧ *V* = (⨁ *U*←*Us*. *U*)
**using** *FinGroupRepresentation-axioms FinGroupRepresentation-reducible'* **by** *fast*

### 6.3.4 Consequence: decomposition relative to a homomomorphism

**lemma** (**in** *FinGroupRepresentation*) *GRepHom-decomp* :
**fixes** *T* :: *'v* ⇒ *'w*::*ab-group-add*
**defines** *KerT* : *KerT* ≡ (*ker T* ∩ *V*)
**assumes** *hom* : *GRepHom smult' T* **and** *nonzero*: *V* ≠ *0*
**shows** ∃ *U*. *GSubspace U* ∧ *V* = *U* ⊕ *KerT*
∧ *FGModule.isomorphic G smult U smult'* (*T* ‘ *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* ⊕ *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* ↓ *U*) (*T* ‘ *V*)
**proof** (*rule FGModuleIso.intro*)
**from** *U*(*1*) **show** *FGModuleHom G* (·) *U smult'* (*T* ↓ *U*)
**using** *FGModuleHom.FGModuleHom-restrict0-GSubspace*[*OF hom*] **by** *fast*

```
    show FGModuleIso-axioms U (T ↓ U) (T ‘ V)
      unfolding FGModuleIso-axioms-def bij-betw-def
    proof (rule conjI, rule inj-onI)
      show (T ↓ U) ‘ U = T ‘ V
      proof
        from U(1) show (T ↓ U) ‘ U ⊆ T ‘ V by auto
        show (T ↓ U) ‘ U ⊇ T ‘ V
        proof (rule image-subsetI)
          fix v assume v ∈ V
          with U(2) obtain u k where uk: u ∈ U k ∈ KerT v = u + k
            using inner-dirsum-doubleD[OF indep] set-plus-def[of U KerT] by fast
          with KerT U(1) have T v = (T ↓ U) u
            using kerD FGModuleHom.additive[OF hom] by force
          with uk(1) show T v ∈ (T ↓ U) ‘ U by fast
        qed
      qed
    next
      fix x y assume xy: x ∈ U y ∈ U (T ↓ U) x = (T ↓ U) y
      with U(1) KerT have x − y ∈ U ∩ KerT
        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 KerT' have AbGroup KerT
        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 ⟹ T ‘ V ≠ 0 ⟹ 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 ‘ V ⊆ W
          T ‘ V ≠ 0
  shows   T ‘ V = W
proof−
  have FGModule.GSubspace G smult' W (T ‘ 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* ⊆ *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* ⊆ *W T ‘ V* ≠ *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*: ⋀*p*::*′b poly*. *degree p > 0* ⟹ ∃ *c*. *poly p c* = *0*
**shows** ∃ *c*. ∀ *v* ∈ *V*. *T v* = *c* ♯· *v*
**proof**−
**from** *fdim alg-closed* **obtain** *e u* **where** *eu*: *u* ∈ *V u* ≠ *0 T u* = *e* ♯· *u*
**using** *FGModuleEnd.VectorSpaceEnd*[*OF GRep*]
*FinDimVectorSpace.endomorph-has-eigenvector*[
*OF FinDimVectorSpace*, *of T*
]
**by** *fast*
**define** *U* **where** *U* = {*v* ∈ *V*. *T v* = *e* ♯· *v*}
**moreover from** *eu U-def* **have** *U* ≠ *0* **by** *auto*
**ultimately have** ∀ *v* ∈ *V*. *T v* = *e* ♯· *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 ≡ group-ring :: ('f::field, 'g) 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 ≡ group-ring :: ('f::field, 'g) aezfun set
  assumes good-char: of-nat (card G) ≠ (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
              ∀f∈ FG. ∃ bs. length bs = length gs
                    ∧ f = (∑ (b,g)←zip bs gs. (b δδ 0) ∗ (1 δδ g))
      using good-card-imp-finite FinGroupI FinGroup.group-ring-spanning-set
      by    fast
    define xs where xs = map ((δδ) (1::'f)) gs
    with FG gs(1) have 1: set xs ⊆ 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 ⊆ FG
        using FGModule-FG FGModule.fSpan-closed by fast
      show aezfun-scalar-mult.fSpan (∗) xs ⊇ FG
      proof
        fix x assume x ∈ FG
        from this gs(2) obtain bs
          where bs: length bs = length gs
                  x = (∑ (b,g)←zip bs gs. (b δδ 0) ∗ (1 δδ g))
          by    fast
        from bs(2) xs-def have x = (∑ (b,a)←zip bs xs. (b δδ 0) ∗ a)
          using sum-list-prod-map2[THEN sym] by fast
        with bs(1) xs-def show x ∈ aezfun-scalar-mult.fSpan (∗) xs
          using aezfun-scalar-mult.fsmultD[of (∗), THEN sym]
                sum-list-prod-cong[
                  of zip bs xs λb a. (b δδ 0) ∗ a
                     λb a. 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** $\exists xs.\ set\ xs \subseteq FG \wedge aezfun\text{-}scalar\text{-}mult.fSpan\ (*)\ xs = FG$
**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*) ≠ (*0*::*'f*::*field*)
**shows** ∃ *Us*::(*'f*,*'g*) *aezfun set list*.
(∀ *U*∈*set Us*. *IrrFinGroupRepresentation G* (∗) *U*) ∧ *0* ∉ *set Us*
∧ *group-ring* = (⨁ *U*←*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 smult' W*
**shows** *IrrFinGroupRepresentation G smult V*
**proof** (*rule FinGroupRepresentation.IrrI*)
**from** *assms* **show** *FinGroupRepresentation G* (·) *V*
**using** *IrrFinGroupRepresentation.axioms*(*1*) *isomorphic-sym*
*FinGroupRepresentation.isomorphic-imp-GRep*
**by** *fast*
**from** *assms* **show** ⋀*U*. *GSubspace U* ⟹ *U* = *0* ∨ *U* = *V*
**using** *IrrFinGroupRepresentation.irr isomorphic-to-irr-right'* **by** *fast*
**qed**

**lemma** (**in** *FinGroupRepresentation*) *IrrGSubspace-iso-constituent* :
**assumes** *nonzero* : *V* ≠ *0*
**and** *subsp* : *W* ⊆ *V W* ≠ *0 IrrFinGroupRepresentation G smult W*
**and** *V-decomp*: ∀ *U*∈*set Us*. *IrrFinGroupRepresentation G smult U*
*0* ∉ *set Us V* = (⨁ *U*←*Us*. *U*)
**shows** ∃ *U*∈*set Us*. *FGModule.isomorphic G smult W smult U*
**proof**−
**from** *V-decomp*(*1*) **have** *abGrp*: ∀ *U*∈*set Us*. *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** ∀ *U*∈*set Us*. *U* ⊆ *V*
**using** *abGrp AbGroup-subset-inner-dirsum* **by** *fast*
**with** *subsp*(*1*,*3*) *V-decomp*(*1*)
**have** *GSubspaces*: *GSubspace W* ∀ *U*∈ *set Us*. *GSubspace U*
**using** *IrrFinGroupRepresentation.axioms*(*1*)[*of G smult*]

```
        FinGroupRepresentation.axioms(1)[of G smult] GSubspaceI
   by   auto
  have ⋀i. i < length Us ⟹ (⨁ Us↓i) ‘ W ≠ 0
           ⟹ FGModuleIso G smult W smult ( (⨁ Us↓i) ↓ W ) (Us!i)
  proof−
   fix i assume i: i < length Us (⨁ Us↓i) ‘ W ≠ 0
   from i(1) V-decomp(3) have GRepEnd (⨁ Us↓i)
    using GSubspaces(2) indep GEnd-inner-dirsum-el-decomp-nth by fast
   hence FGModuleHom G smult W smult ( (⨁ Us↓i) ↓ W )
    using GSubspaces(1) FGModuleEnd.FGModuleHom-restrict0-GSubspace
    by   fast
   moreover have ( (⨁ Us↓i) ↓ W ) ‘ W ⊆ Us!i
   proof−
    from V-decomp(3) i(1) subsp(1,3) have (⨁ Us↓i) ‘ W ⊆ 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 ( (⨁ Us↓i) ↓ W ) (Us!i)
    using i(2)
          IrrFinGroupRepresentation.Schur1[
           OF subsp(3), of smult Us!i (⨁ Us↓i) ↓ W
          ]
    by   auto
  qed
  moreover from nonzero V-decomp(3)
   have ∀ i<length Us. (⨁ Us↓i) ‘ W = 0 ⟹ W = 0
   using inner-dirsum-Nil abGrp subsp(1) indep
         AbGroup-inner-dirsum-subset-proj-eq-0[of Us W]
   by   fastforce
  ultimately have ∃ i<length Us.  FGModuleIso G smult W smult
                      ( (⨁ Us↓i) ↓ 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 ≠ 0
  and     FG-decomp: ∀ U∈set Us. IrrFinGroupRepresentation G (∗) U
                     0 ∉ set Us FG = (⨁ U←Us. U)
  shows   ∃ U∈set Us. isomorphic (∗) U
proof−
  from nonzero obtain v where v: v ∈ V v ≠ 0 using nonempty by auto
  define T where T = (λx. x · v)↓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 ⋀v v'. v ∈ FG ⟹ v' ∈ FG ⟹ T (v + v') = T v +
T v'
    using Ring1.add-closed[OF Ring1] smult-distrib-right by auto
  from T-def show supp T ⊆ FG using supp-restrict0 by fast
  from T-def v(1) show ⋀r m. r ∈ FG ⟹ m ∈ FG ⟹ T (r * m) = r · T m
    using ActingGroup.RG-mult-closed by auto
 qed
 then obtain W
  where W: FGModule.GSubspace G (*) FG W FG = W ⊕ (ker T ∩ 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 ≠ 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 ∃ U∈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 ⇒ ('f,'g) aezfun set list where
```

```
  remisodups [] = []
| remisodups (U # Us) = (if
      (∃ W∈set Us. FGModule.isomorphic G (*) U (*) W)
       then remisodups Us else U # remisodups Us)

lemma set-remisodups : set (remisodups Us) ⊆ set Us
  by (induct Us) auto

lemma isodistinct-remisodups :
  ⟦ ∀ U∈set Us. FGModule G (*) U; V ∈ set (remisodups Us);
      W ∈ set (remisodups Us); V ≠ W ⟧
       ⟹ ¬ (FGModule.isomorphic G (*) V (*) W)
proof (induct Us arbitrary: V W)
  case (Cons U Us)
  show ?case
  proof (cases ∃ X∈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 ≡ SOME Us.
                (∀ U∈set Us. IrrFinGroupRepresentation G (*) U)
                  ∧ 0 ∉ set Us ∧ group-ring = (⨁ U←Us. U)

lemma FG-constituents-irr :
  of-nat (card G) ≠ (0::'f::field)
       ⟹ ∀ U∈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-consitutents-n0:
  of-nat (card G) ≠ (0::'f::field)
       ⟹ 0 ∉ 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*) ≠ (*0*::*'f*::*field*)
      ⟹ (*group-ring*::(*'f*,*'g*) *aezfun set*) = (⨁ *U*←*FG-constituents*. *U*)
  **using** *someI-ex*[*OF FG-reducible*] **unfolding** *FG-constituents-def* **by** *fast*

**definition** *GIrrRep-repset* ≡ *0* ∪ *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*) ≠ (*0*::*'f*::*field*)
  **shows** ∀ *U*∈(*GIrrRep-repset*::(*'f*,*'g*) *aezfun set set*).
            *IrrFinGroupRepresentation G* (∗) *U*
**proof**
  **fix** *U* :: (*'f*,*'g*) *aezfun set* **assume** *U* ∈ *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* ≡ *GIrrRep-repset* :: (*'f*::*field*,*'g*) *aezfun set set*
  **assumes** *of-nat* (*card G*) ≠ (*0*::*'f*) *V* ∈ *GIRRS W* ∈ *GIRRS V* ≠ *W*
  **shows**  ¬ (*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-consitutents-n0*
        *trivial-FGModule*[*of* (∗)] *FGModule.isomorphic-to-zero-left*[*of G* (∗)]
    **by**  *fastforce*
**next**
  **case** *OtherTrue*
  **moreover with** *assms*(*1*,*3*) **have** *V* ∈ *set FG-constituents*
    **using** *GIrrRep-repset-def set-remisodups* **by** *auto*
  **ultimately show** *?thesis*
    **using** *assms*(*2*) *FG-consitutents-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* ∈ *set* (*remisodups FG-constituents*)
                    *W* ∈ *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* :
$\exists\, U{\in}set\ Us.\ isomorphic\ (*)\ U$
$\Longrightarrow \exists\, U{\in}set\ (ActingGroup.remisodups\ Us).\ isomorphic\ (*)\ U$
**proof** (*induct Us*)
**case** (*Cons U Us*)
**from** *Cons(2)* **obtain** *W* **where** *W*: $W \in set\ (U\#Us)$ *isomorphic* $(*)$ *W*
**by** *fast*
**show** *?case*
**proof** (
*cases* $W = U\ \exists\, X{\in}set\ Us.$ *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* :
$\exists\, U{\in}ActingGroup.GIrrRep\text{-}repset.\ isomorphic\ (*)\ U$
**using** *zero-isomorphic-to-FG-zero ActingGroup.GIrrRep-repset-def*
*good-char ActingGroup.FG-constituents-irr*
*ActingGroup.FG-consitutents-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** $GIRRS \equiv GIrrRep\text{-}repset :: ('f{::}field, 'g)\ aezfun\ set\ set$
**assumes** *of-nat* $(card\ G) \neq (0{::}'f)$
**shows** *finite GIRRS*
$\forall\, U{\in}GIRRS.\ IrrFinGroupRepresentation\ G\ (*)\ U$
$\bigwedge U\ W.\ [\![\ U \in GIRRS;\ W \in GIRRS;\ U \neq W\ ]\!]$
$\Longrightarrow \neg\ (FGModule.isomorphic\ G\ (*)\ U\ (*)\ W)$
$\bigwedge fgsmult\ V.\ IrrFinGroupRepresentation\ G\ fgsmult\ V$
$\Longrightarrow \exists\, U{\in}GIRRS.\ FGModule.isomorphic\ G\ fgsmult\ V\ (*)\ U$
**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* ⇒ *'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*)
+ **fixes** *FH* :: (*'f*, *'g*) *aezfun set*
**and** *indV* :: ((*'f*, *'g*) *aezfun* ⇒ *'v*) *set*
**defines** *FH* : *FH* ≡ *Supgroup.group-ring*
**and** *indV* : *indV* ≡ *BaseRep.indspace G FH V*
**assumes** *rrsmult* : *rrsmult* = *Ring1.rightreg-scalar-mult FH*
**and** *good-ordSupgrp*: *of-nat* (*card H*) ≠ (*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* ≡ *of-nat* (*card H*)
**abbreviation** *is-Vfbasis* ≡ *fbasis-for V*
**abbreviation** *GRepHomSet* ≡ *FGModuleHomSet G smult V*
**abbreviation** *HRepHom* ≡ *FGModuleHom H rrsmult indV*
**abbreviation** *HRepHomSet* ≡ *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* ≡ *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** *hfvss* : *hfvss* ≡ *negHorbit-list hs induced-vector vs*
**assumes** *vs* : *set vs* ⊆ *V* **and** *i*: *i* < *length hs*
**and** *scalars* : *list-all2* (λ*rs ms*. *length rs* = *length ms*) *css hfvss*
**and** *rcoset-reps* : *Supgroup.is-rcoset-replist G hs*
**shows** ((*concat css*) •¤¤ (*concat hfvss*)) (*1* δδ (*hs*!*i*)) = (*css*!*i*) •♯· *vs*
**proof**−
**have** *mostly-zero*:
⋀*k j*. *k* < *length hs* ⟹ *j* < *length hs*
⟹ ((*css*!*j*) •¤¤ (*hfvss*!*j*)) (*1* δδ *hs*!*k*)
= (*if j* = *k then* (*css*!*k*) •♯· *vs else 0*)
**proof**−
**fix** *k j* **assume** *k*: *k* < *length hs* **and** *j*: *j* < *length hs*
**hence** *hsk-H*: *hs*!*k* ∈ *H* **and** *hsj-H*: *hs*!*j* ∈ *H*
**using** *Supgroup.is-rcoset-replistD-set*[*OF rcoset-reps*] **by** *auto*
**define** *LHS* **where** *LHS* = ((*css*!*j*) •¤¤ (*hfvss*!*j*)) (*1* δδ *hs*!*k*)
**with** *hfvss*
**have** *LHS* = (∑(*r*,*m*)←*zip* (*css*!*j*) (*hfvss*!*j*). (*r* ¤¤ *m*) (*1* δδ *hs*!*k*))
**using** *length-negHorbit-list scalar-mult.lincomb-def*[*of induced-smult.fsmult*]
*sum-list-prod-fun-apply*
**by** *simp*
**moreover have** ∀(*r*,*m*) ∈ *set* (*zip* (*css*!*j*) (*hfvss*!*j*)).
(*induced-smult.fsmult r m*) (*1* δδ *hs*!*k*) = *r* ♯· *m* (*1* δδ *hs*!*k*)
**proof** (*rule prod-ballI*)
**fix** *r m* **assume** (*r*,*m*) ∈ *set* (*zip* (*css*!*j*) (*hfvss*!*j*))
**with** *k j vs hfvss*
**show** (*induced-smult.fsmult r m*) (*1* δδ *hs*!*k*) = *r* ♯· *m* (*1* δδ *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 hfvss*] *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 = (∑(r,v)←zip (css!j) vs.
        r ♯· (induced-vector v) (1 δδ hs!k ∗ (1 δδ − hs!j)))
  using FH j hfvss induced-smult.negGorbit-list-def[of hs induced-vector vs]
        sum-list-prod-cong[of - λr m. (induced-smult.fsmult r m) (1 δδ hs!k)]
        sum-list-prod-map2[of
          λr m. r ♯· m (1 δδ hs!k) - Hmult (− hs!j) map induced-vector vs
        ]
        sum-list-prod-map2[of λr v. r ♯· (Hmult (−hs!j) v) (1 δδ hs!k)]
        induced-smult.GmultD hsk-H
        Supgroup.RG-aezdeltafun-closed[of hs!k 1::'f]
        rrsmultD1[of 1 δδ (hs!k)]
  by    force
moreover have (1::'f) δδ hs!k ∗ (1 δδ − hs!j) = 1 δδ (hs!k − hs!j)
  using times-aezdeltafun-aezdeltafun[of 1::'f hs!k 1 −(hs!j)]
  by (simp add: algebra-simps)
ultimately have LHS = (∑(r,v)←zip (css!j) vs.
                    r ♯· (induced-vector v) (1 δδ (hs!k − hs!j)))
  using sum-list-prod-map2 by simp
moreover from FH vs
 have ∀(r,v) ∈ set (zip (css!j) vs). r ♯· (induced-vector v) (1 δδ (hs!k − hs!j))
            = r ♯· (FG-proj (1 δδ (hs!k − hs!j)) · v)
  using set-zip-rightD induced-vector-def hsk-H hsj-H Supgroup.diff-closed
        Supgroup.RG-aezdeltafun-closed[of - 1::'f]
  by    fastforce
ultimately have calc: LHS = (∑(r,v)←zip (css!j) vs.
                         r ♯· (FG-proj (1 δδ (hs!k − hs!j)) · v) )
  using sum-list-prod-cong by force
show LHS = (if j = k then (css!k) ·♯· vs else 0)
proof (cases j = k)
  case True
  with calc have LHS = (∑(r,v)←zip (css!k) vs. r ♯· 1 ♯· v)
    using Group.zero-closed[OF GroupG]
          aezfun-setspan-proj-aezdeltafun[of G 1::'f]
          BaseRep.fsmult-def
    by    simp
  moreover from vs have ∀(r,v) ∈ set (zip (css!k) vs). r ♯· 1 ♯· v = r ♯· v
    using set-zip-rightD BaseRep.fsmult-assoc by fastforce
  ultimately show ?thesis
    using True sum-list-prod-cong[of - λr v. r ♯· 1 ♯· v]
          scalar-mult.lincomb-def[of BaseRep.fsmult]
    by    simp
next
  case False
  with k j calc have LHS = (∑(r,v)←zip (css!j) vs. r ♯· (0 · v))
    using Supgroup.is-rcoset-replist-imp-nrelated-nth[OF Subgroup rcoset-reps]
          aezfun-setspan-proj-aezdeltafun[of G 1::'f]
    by    simp
  moreover from vs have ∀(r,v) ∈ set (zip (css!j) vs). r ♯· (0 · v) = 0
    using set-zip-rightD BaseRep.zero-smult by fastforce
```

```
    ultimately have LHS = (∑ (r,v)←zip (css!j) vs. (0::'v))
      using sum-list-prod-cong[of - λr v. r ♯· (0 · v)] by simp
    hence LHS = (∑ rv←zip (css!j) vs. case-prod (λr v. (0::'v)) rv) by fastforce
    with False show ?thesis by simp
  qed
qed

define terms LHS
  where terms = map (λa. case-prod (λcs hfvs. (cs ·¤ ¤ hfvs) (1 δδ hs!i)) a)
(zip css hfvss)
    and LHS = ((concat css) ·¤ ¤ (concat hfvss)) (1 δδ (hs!i))
hence LHS = sum-list terms
  using scalars
        VectorSpace.lincomb-concat[OF fVectorSpace-indspace, of css hfvss]
        sum-list-prod-fun-apply
  by    simp
hence LHS = (∑ j∈{0..<length terms}. terms!j)
  using sum-list-sum-nth[of terms] by simp
moreover from terms-def
  have ∀ j∈{0..<length terms}. terms!j = ((css!j) ·¤ ¤ (hfvss!j)) (1 δδ hs!i)
  by   simp
ultimately show LHS = (css!i) ·♯· vs
  using terms-def sum.cong scalars list-all2-lengthD[of - css hfvss] hfvss
        length-negHorbit-list[of hs induced-vector vs] i mostly-zero
        sum-single-nonzero[
          of {0..<length hs} λi j. ((css!j) ·¤ ¤ (hfvss!j)) (1 δδ (hs!i))
             λi. (css!i) ·♯· vs
        ]
  by    simp

qed

lemma indspace-fspanning-set :
  fixes   vs          :: 'v list
  and     hs          :: 'g list
  defines hfvss       : hfvss ≡ negHorbit-list hs induced-vector vs
  assumes base-spset  : set vs ⊆ V V = BaseRep.fSpan vs
  and     rcoset-reps : Supgroup.is-rcoset-replist G hs
  shows   indV = induced-smult.fSpan (concat hfvss)
proof (rule VectorSpace.SpanI[OF fVectorSpace-indspace])
  from base-spset(1) hfvss show set (concat hfvss) ⊆ indV
    using Supgroup.is-rcoset-replistD-set[OF rcoset-reps]
          induced-vector-indV negHorbit-list-indV
    by    fast
  show indV ⊆ R-scalar-mult.RSpan UNIV (aezfun-scalar-mult.fsmult (¤))
            (concat hfvss)
  proof

    fix f assume f: f ∈ indV
```

**hence** $\forall\, h \in$ *set hs*. $f\ (1\ \delta\delta\ h) \in V$
**using** *indV BaseRep.indspaceD-range* **by** *fast*
**with** *base-spset*(*2*)
**have** *coeffs-exist*: $\forall\, h \in$ *set hs*. $\exists\, ahs.$ *length ahs* = *length vs*
$\wedge\ f\ (1\ \delta\delta\ h) = ahs\ \cdot\sharp\cdot\ vs$
**using** *BaseRep.in-fSpan-obtain-same-length-coeffs*
**by** *fast*
**define** *f-coeffs*
**where** *f-coeffs h* = (*SOME ahs*. *length ahs* = *length vs* $\wedge$ *f* (*1* $\delta\delta$ *h*) = *ahs* $\cdot\sharp\cdot$ *vs*) **for** *h*
**define** *ahss* **where** *ahss* = *map f-coeffs hs*
**hence** *len-ahss*: *length ahss* = *length hs* **by** *simp*
**with** *hfvss* **have** *len-zip-ahss-hfvss*: *length* (*zip ahss hfvss*) = *length hs*
**using** *length-negHorbit-list*[*of hs induced-vector vs*] **by** *simp*
**have** *len-ahss-el*: $\forall$ *ahs*$\in$*set ahss*. *length ahs* = *length vs*
**proof**
**fix** *ahs* **assume** *ahs* $\in$ *set ahss*
**from** *this ahss-def* **obtain** *h* **where** *h*: *h* $\in$ *set hs ahs* = *f-coeffs h*
**using** *set-map* **by** *auto*
**from** *h*(*1*) **have** $\exists\, ahs.$ *length ahs* = *length vs* $\wedge$ *f* (*1* $\delta\delta$ *h*) = *ahs* $\cdot\sharp\cdot$ *vs*
**using** *coeffs-exist* **by** *fast*
**with** *h*(*2*) **show** *length ahs* = *length vs*
**unfolding** *f-coeffs-def*
**using** *someI-ex*[*of* $\lambda$*ahs*. *length ahs* = *length vs* $\wedge$ *f* (*1* $\delta\delta$ *h*) = *ahs* $\cdot\sharp\cdot$ *vs*]
**by** *fast*
**qed**
**have** $\forall$ (*ahs*,*hfvs*)$\in$*set* (*zip ahss hfvss*). *length ahs* = *length hfvs*
**proof**
**fix** *x* **assume** *x*: *x* $\in$ *set* (*zip ahss hfvss*)
**show** *case x of* (*ahs*, *hfvs*) $\Rightarrow$ *length ahs* = *length hfvs*
**proof**
**fix** *ahs hfvs* **assume** *x* = (*ahs*,*hfvs*)
**with** *x hfvss* **have** *length ahs* = *length vs length hfvs* = *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** *length ahs* = *length hfvs* **by** *simp*
**qed**
**qed**
**with** *hfvss* **have** *list-all2-len-ahss-hfvss*:
*list-all2* ($\lambda$*rs ms*. *length rs* = *length ms*) *ahss hfvss*
**using** *len-ahss length-negHorbit-list*[*of hs induced-vector vs*]
*list-all2I*[*of ahss hfvss*]
**by** *auto*

**define** *f'* **where** *f'* = (*concat ahss*) $\cdot\,\overline{\times}\,\overline{\times}$ (*concat hfvss*)
**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 hfvss base-spset(1) show f' ∈ indV
        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 ⋀i. i<length hs ⟹ f (1 δδ (hs!i)) = f' (1 δδ (hs!i))
      proof−
        fix i assume i: i < length hs
        with f-coeffs-def have f (1 δδ (hs!i)) = (f-coeffs (hs!i)) ·♯· vs
          using coeffs-exist
                someI-ex[of λahs. length ahs = length vs ∧ f (1 δδ hs!i) = ahs ·♯· vs]
          by    auto
        moreover from i hfvss f'-def base-spset(1) rcoset-reps ahss-def
          have  f' (1 δδ (hs!i)) = (f-coeffs (hs!i)) ·♯· vs
          using list-all2-len-ahss-hfvss flincomb-Horbit-induced-veclist-reduce
          by    simp
        ultimately show f (1 δδ (hs!i)) = f' (1 δδ (hs!i)) by simp
      qed
      thus ∀ i<length hs. f (1 δδ (hs!i)) = f' (1 δδ (hs!i)) by fast
    qed
    with f'-def hfvss base-spset(1) show f ∈ induced-smult.fSpan (concat hfvss)
      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 hfvss       : hfvss ≡ negHorbit-list hs induced-vector vs
  assumes base-spset  : BaseRep.fbasis-for V vs
  and     rcoset-reps : Supgroup.is-rcoset-replist G hs
  shows   fscalar-mult.basis-for induced-smult.fsmult indV (concat hfvss)
proof−
  from assms
    have  1: set (concat hfvss) ⊆ indV
      and     indV = induced-smult.fSpan (concat hfvss)
    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 hfvss)
  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 hfvss
               (concat rss) •¤ ¤ (concat hfvss) = 0
    from rss(1) have len-rss-hfvsss: length rss = length hfvss
      using list-all2-lengthD by fast
    with hfvss 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 hfvss rss(1) have length rs = length vs
        using list-all2-nthD len-rss-hfvsss in-set-conv-nth[of - hfvss]
              length-negHorbit-list-sublist
        by    fastforce
      moreover from hfvss 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 hfvss      : hfvss ≡ 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 hfvss)
proof−
  from hfvss base-spset
    have  hfvss = (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 hfvss)
    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* ⊆ *V V* = *BaseRep.fSpan vs*
**by** *fast*
**obtain** *hs* **where** *hs*: *Supgroup.is-rcoset-replist G hs*
**using** *ex-rcoset-replist* **by** *fast*
**define** *hfvss* **where** *hfvss* = *negHorbit-list hs induced-vector vs*
**with** *vs hs*
**have** *set* (*concat hfvss*) ⊆ *indV indV* = *induced-smult.fSpan* (*concat hfvss*)
**using** *Supgroup.is-rcoset-replistD-set*[*OF hs*] *induced-vector-indV*
*negHorbit-list-indV*[*of hs induced-vector vs*] *indspace-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* (¤) *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 FinDimVectorSpace.basis-ex[OF GRep.FinDimVectorSpace] 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** $\varphi$ :: (((*'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** $\varphi$*-im* : $\varphi$ ‘ *HRepHomSet* ($\star$) *W* $\subseteq$ *GRepHomSet* ($\star$) *W*
**proof** (*rule image-subsetI*)

**fix** *T* **assume** *T*: *T* $\in$ *HRepHomSet* ($\star$) *W*
**show** $\varphi$ *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, $\varphi$.

**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** ∃ *S*. *ψ-condition T S*
**proof**−
**from** *assms* **obtain** *S*
**where** *S*: *VectorSpaceHom induced-smult.fsmult indV fsmult′ S*
*map S* (*concat indVfbasis*) = *concat* (*negHorbit-homVfbasis T*)
**using** *inverse-im-exists′*
**by** *fast*
**from** *S*(*2*) **have** *concat* (*map* (*map S*) *indVfbasis*)
= *concat* (*negHorbit-homVfbasis T*)
**using** *map-concat*[*of S*] **by** *simp*
**moreover have** *list-all2* (λ*xs ys*. *length xs* = *length ys*)
(*map* (*map S*) *indVfbasis*) (*negHorbit-homVfbasis T*)
**proof** (*rule iffD2*[*OF list-all2-iff*], *rule conjI*)
**show** *length* (*map* (*map S*) *indVfbasis*) = *length* (*negHorbit-homVfbasis T*)
**using** *indVfbasis-def induced-smult.length-negGorbit-list*
*HRep.length-negGorbit-list*[*of H-rcoset-reps T*]
**by** *auto*
**show** ∀(*xs*,*ys*)∈*set* (*zip* (*map* (*map S*) *indVfbasis*)
(*negHorbit-homVfbasis T*)). *length xs* = *length ys*
**proof** (*rule prod-ballI*)
**fix** *xs ys*
**assume** *xsys*: (*xs*,*ys*) ∈ *set* (*zip* (*map* (*map S*) *indVfbasis*)
(*negHorbit-homVfbasis T*))
**from** *this* **obtain** *zs* **where** *zs*: *zs* ∈ *set indVfbasis xs* = *map S zs*
**using** *set-zip-leftD* **by** *fastforce*
**with** *xsys* **show** *length xs* = *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** *map* (*map S*) *indVfbasis* = *negHorbit-homVfbasis T*
**using** *concat-eq*[*of map* (*map S*) *indVfbasis*] **by** *fast*
**with** *S*(*1*) **show** *?thesis* **using** *ψ-condition-def* **by** *fast*
**qed**
**moreover have** ⋀*S U*. *ψ-condition T S* ⟹ *ψ-condition T U* ⟹ *S* = *U*
**proof**−
**fix** *S U* **assume** *ψ-condition T S ψ-condition T U*
**hence** *VectorSpaceHom induced-smult.fsmult indV fsmult′ S*
*map S* (*concat indVfbasis*) = *concat* (*negHorbit-homVfbasis T*)
*VectorSpaceHom induced-smult.fsmult indV fsmult′ U*
*map U* (*concat indVfbasis*) = *concat* (*negHorbit-homVfbasis T*)
**using** *ψ-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** $\psi :: ('v \Rightarrow 'w) \Rightarrow (((('f, 'g)\ aezfun \Rightarrow 'v) \Rightarrow 'w)$
  **where** $\psi \equiv restrict0\ (\lambda T.\ THE\ S.\ \psi\text{-}condition\ T\ S)\ (GRepHomSet\ (\star)\ W)$

**lemma** $\psi D : T \in GRepHomSet\ (\star)\ W \Longrightarrow \psi\text{-}condition\ T\ (\psi\ T)$
  **using** $\psi\text{-}def\ inverse\text{-}im\text{-}exists[of\ T]\ theI'[of\ \lambda S.\ \psi\text{-}condition\ T\ S]$ **by** *simp*

**lemma** $\psi D\text{-}VectorSpaceHom$ :
  $T \in GRepHomSet\ (\star)\ W$
      $\Longrightarrow VectorSpaceHom\ induced\text{-}smult.fsmult\ indV\ fsmult'\ (\psi\ T)$
  **using** $\psi D\ \psi\text{-}condition\text{-}def$ **by** *fast*

**lemma** $\psi D\text{-}im$ :
  $T \in GRepHomSet\ (\star)\ W \Longrightarrow map\ (map\ (\psi\ T))\ indVfbasis$
      $= aezfun\text{-}scalar\text{-}mult.negGorbit\text{-}list\ (\star)\ H\text{-}rcoset\text{-}reps\ T\ Vfbasis$
  **using** $\psi D\ \psi\text{-}condition\text{-}def$ **by** *fast*

**lemma** $\psi D\text{-}im\text{-}single$ :
  **assumes** $T \in GRepHomSet\ (\star)\ W\ h \in set\ H\text{-}rcoset\text{-}reps\ v \in set\ Vfbasis$
  **shows** $\psi\ T\ ((-\ h) *¤\ (induced\text{-}vector\ v)) = (-h) *\star\ (T\ v)$
**proof**−
  **from** $assms(2,3)$ **obtain** $i\ j$
    **where** $i$: $i < length\ H\text{-}rcoset\text{-}reps\ h = H\text{-}rcoset\text{-}reps!i$
    **and** $j$: $j < length\ Vfbasis\ v = Vfbasis!j$
    **using** $set\text{-}conv\text{-}nth[of\ H\text{-}rcoset\text{-}reps]\ set\text{-}conv\text{-}nth[of\ Vfbasis]$ **by** *auto*
  **moreover**
    **hence** $map\ (map\ (\psi\ T))\ indVfbasis\ !i\ !j = \psi\ T\ ((-h) *¤\ (induced\text{-}vector\ v))$
    **using** $indVfbasis\text{-}def$
          $aezfun\text{-}scalar\text{-}mult.length\text{-}negGorbit\text{-}list[$
            $of\ rrsmult\ H\text{-}rcoset\text{-}reps\ induced\text{-}vector$
          $]$
          $aezfun\text{-}scalar\text{-}mult.negGorbit\text{-}list\text{-}nth[$
            $of\ i\ H\text{-}rcoset\text{-}reps\ rrsmult\ induced\text{-}vector$
          $]$
    **by** *auto*
  **ultimately show** *?thesis*
    **using** $assms(1)\ HRep.negGorbit\text{-}list\text{-}nth[of\ i\ H\text{-}rcoset\text{-}reps\ T]\ \psi D\text{-}im$ **by** *simp*
**qed**

**lemma** $\psi T\text{-}W$ :
  **assumes** $T \in GRepHomSet\ (\star)\ W$
  **shows** $\psi\ T\ `\ indV \subseteq W$
**proof** $(rule\ image\text{-}subsetI)$
  **from** *assms* **have** $T$: $VectorSpaceHom\ induced\text{-}smult.fsmult\ indV\ fsmult'\ (\psi\ T)$
    **using** $\psi D\text{-}VectorSpaceHom$ **by** *fast*
  **fix** $f$ **assume** $f \in indV$
  **from** *this* **obtain** $cs$
    **where** $cs$:$length\ cs = length\ (concat\ indVfbasis)\ f = cs \cdot ¤\ ¤\ (concat\ indVfbasis)$
    **using** $indVfbasis\ scalar\text{-}mult.in\text{-}Span\text{-}obtain\text{-}same\text{-}length\text{-}coeffs$

```
  by   fast
 from cs(1) obtain css
  where css: cs = concat css list-all2 (λxs ys. length xs = length ys) css indVfbasis
  using match-concat
  by   fast
 from assms cs(2) css
  have ψ T f = ψ T (∑ (cs,hfvs)←zip css indVfbasis. cs ·¤¤ hfvs)
  using VectorSpace.lincomb-concat[OF fVectorSpace-indspace] by simp
 also have ... = (∑ (cs,hfvs)←zip css indVfbasis. ψ T (cs ·¤¤ 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 ψ T f = (∑ (cs,ψ Thfvs)←zip css (map (map (ψ T)) indVfbasis).
                       cs ·♯⋆ ψ Thfvs)
  using set-zip-rightD[of - - css indVfbasis] indVfbasis-indV
        VectorSpaceHom.distrib-lincomb[OF T]
        sum-list-prod-cong[of
          zip css indVfbasis λcs hfvs. ψ T (cs ·¤¤ hfvs)
          λcs hfvs. cs ·♯⋆ (map (ψ T) hfvs)
        ]
        sum-list-prod-map2[of λcs ψ Thfvs. cs ·♯⋆ ψ Thfvs css map (ψ T)]
  by   fastforce
 moreover from css(2)
  have list-all2 (λxs ys. length xs = length ys) css (map (map (ψ T)) indVfbasis)
  using list-all2-iff[of - css indVfbasis] set-zip-map2
        list-all2-iff[of - css map (map (ψ T)) indVfbasis]
  by   force
 ultimately have ψ T f = (concat css) ·♯⋆ (concat (negHorbit-homVfbasis T))
  using HRep.flincomb-concat map-concat[of ψ T] ψD-im[OF assms]
  by   simp
 thus ψ T f ∈ W
  using assms negHorbit-HomSet-indVfbasis-W HRep.flincomb-closed by simp
qed

lemma ψT-Hmap-on-indVfbasis :
 assumes T ∈ GRepHomSet (⋆) W
 shows   ⋀x f. x ∈ H ⟹ f ∈ set (concat indVfbasis)
           ⟹ ψ T (x ∗¤ f) = x ∗⋆ (ψ T f)
proof−
 fix x f assume x: x ∈ H and f: f ∈ set (concat indVfbasis)
 from f obtain i where i: i < length indVfbasis f ∈ set (indVfbasis!i)
  using set-concat set-conv-nth[of indVfbasis] by auto
 from i(1) have i': i < length H-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 ∗¤ 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** $\psi\ T$ ‘ $indV \subseteq W$ **using** *indVfbasis* $\psi T$-$W$[*OF assms*] **by** *fast*
**show** $\bigwedge g\ v.\ g \in H \Longrightarrow v \in set\ (concat\ indVfbasis)$
$\Longrightarrow \psi\ T\ (g *¤\ v) = g *\star \psi\ T\ v$
**using** $\psi T$-*Hmap-on-indVfbasis*[*OF assms*] **by** *fast*
**qed**

**lemma** $\psi$-*im* : $\psi$ ‘ *GRepHomSet* $(\star)$ $W \subseteq$ *HRepHomSet* $(\star)$ $W$
**using** $\psi T$-$W$ $\psi T$-*hom FGModuleHomSetI* **by** *fastforce*

**end**

### 6.8.4 Demonstration of bijectivity

Now we demonstrate that $\varphi$ is bijective via the inverse $\psi$.

**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* ·♯· *Vfbasis*
**using** *Vfbasis*
*VectorSpace.in-Span-obtain-same-length-coeffs*[*OF GRep.fVectorSpace*]
**by** *fast*
**define** *extrazeros*
**where** *extrazeros* = *replicate* ((*length nonzero-H-rcoset-reps*)∗(*length Vfbasis*))
(*0*::′*f*)
**with** *cs* **have** *GRep.induced-vector v* = (*cs*@*extrazeros*) ·¤ ¤ (*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*) ·♯⋆ (*map* $(\psi\ T)$ (*concat indVfbasis*))
**using** $\psi$-*im* $\varphi$-*def indVfbasis*
*VectorSpaceHom.distrib-lincomb*[*OF* $\psi$*D-VectorSpaceHom*]
**by** *auto*
**also have** ... = (*cs*@*extrazeros*) ·♯⋆ (*map T Vfbasis*
@ *concat* (*HRep.negGorbit-list nonzero-H-rcoset-reps T Vfbasis*))
**using** *map-concat*[*of* $\psi$ *T*] $\psi$*D-im*[*OF assms*] *H-rcoset-reps-def*
*FGModuleHomSetD-Im*[*OF assms*] *Vfbasis-V HRep.negGorbit-list-Cons0*
**by** *fastforce*
**also from** *cs*(*1*)
**have** ... = *cs* ·♯⋆ (*map T Vfbasis*) + *extrazeros*
·♯⋆ (*concat* (*HRep.negGorbit-list nonzero-H-rcoset-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 ψ-im φ-def GRep.induced-vector-def ψD-VectorSpaceHom
          VectorSpaceHom.im-zero
          FGModuleHomSetD-FGModuleHom[of T G smult V]
          FGModuleHom.supp suppI-contra
    by   fastforce
 qed
qed

lemma φ-inverse-im : φ ‘ HRepHomSet (⋆) W ⊇ GRepHomSet (⋆) W
 using φψ ψ-im by force

lemma bij-φ : bij-betw φ (HRepHomSet (⋆) W) (GRepHomSet (⋆) W)
 unfolding bij-betw-def
proof

 have ⋀ S T. ⟦ S ∈ HRepHomSet (⋆) W; T ∈ HRepHomSet (⋆) W;
         φ S = φ T ⟧ ⟹ S = T
 proof (rule VectorSpaceHom.same-image-on-spanset-imp-same-hom)
  fix S T
  assume ST: S ∈ HRepHomSet (⋆) W T ∈ HRepHomSet (⋆) W φ S = φ 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-smult.fSpan (concat indVfbasis)
      set (concat indVfbasis) ⊆ indV
    using indVfbasis by auto
```

```
    show ∀f∈set (concat indVfbasis). S f = T f
    proof
      fix f assume f ∈ set (concat indVfbasis)
      from this obtain hfvs where hfvs: hfvs ∈ set indVfbasis f ∈ set hfvs
        using set-concat by fast
      from hfvs(1) obtain h
        where h: h ∈ set H-rcoset-reps
                hfvs = map (Hmult (−h) ∘ induced-vector) Vfbasis
        using indVfbasis-def
              induced-smult.negGorbit-list-def[of H-rcoset-reps induced-vector]
        by    auto
      from hfvs(2) h(2) obtain v
        where v: v ∈ set Vfbasis f = (−h) ∗¤ (induced-vector v)
        by    auto
      from v h(1) ST(1) have S f = (−h) ∗⋆ (φ S v)
        using ST'(1) H-rcoset-reps-H Group.neg-closed[OF HRep.GroupG]
              GRep.induced-vector-indV Vfbasis-V φ-def FGModuleHom.G-map
        by    fastforce
      moreover from v h(1) ST(2) have T f = (−h) ∗⋆ (φ T v)
      using ST'(2) H-rcoset-reps-H Group.neg-closed[OF HRep.GroupG] GRep.induced-vector-indV
              Vfbasis-V φ-def FGModuleHom.G-map
        by    fastforce
      ultimately show S f = T f using ST(3) by simp

    qed
  qed
  thus inj-on φ (HRepHomSet (⋆) W) unfolding inj-on-def by fast

  show φ ‘ HRepHomSet (⋆) W = GRepHomSet (⋆) W
    using φ-im φ-inverse-im by fast

qed

end
```

### 6.8.5 The theorem

Finally we demonstrate that $\varphi$ is an isomorphism of vector spaces between the two hom-sets, leading to Frobenius reciprocity.

```
context FrobeniusReciprocity
begin

lemma VectorSpaceIso-φ :
  VectorSpaceIso Tsmult1 (HRepHomSet (⋆) W) Tsmult2 φ
      (GRepHomSet (⋆) W)
proof (rule VectorSpaceIso.intro, rule VectorSpace.VectorSpaceHomI-fromaxioms)

  from Tsmult1-def show VectorSpace Tsmult1 (HRepHomSet (⋆) W)
    using FHModule-indspace FHModuleW
```

```
      FGModule.VectorSpace-FGModuleHomSet
  by  simp

 from φ-def show supp φ ⊆ HRepHomSet (⋆) W
  using suppD-contra[of φ] by fastforce

 have bij-betw φ (HRepHomSet (⋆) W) (GRepHomSet (⋆) W)
  using bij-φ by fast
 thus VectorSpaceIso-axioms (HRepHomSet (⋆) W) φ (GRepHomSet (⋆) W)
  by unfold-locales

next
 fix S T assume S ∈ HRepHomSet (⋆) W T ∈ HRepHomSet (⋆) W
 thus φ (S + T) = φ S + φ T
  using φ-def Group.add-closed
      FGModule.Group-FGModuleHomSet[OF FHModule-indspace FHModuleW]
  by  auto

next
 fix a T assume T: T ∈ HRepHomSet (⋆) W
 moreover with Tsmult1-def have aT: a ⋆¤ T ∈ HRepHomSet (⋆) W
  using FGModule.VectorSpace-FGModuleHomSet[
        OF FHModule-indspace FHModuleW
       ]
       VectorSpace.smult-closed
  by  simp
 ultimately show φ (a ⋆¤ T) = a ⋆· (φ T)
  using φ-def Tsmult1-def Tsmult2-def by auto

qed

theorem FrobeniusReciprocity :
 VectorSpace.isomorphic Tsmult1 (HRepHomSet smult' W) Tsmult2
     (GRepHomSet smult' W)
 using VectorSpaceIso-φ 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.