example-Z4Z2

By jmaransay

April 17, 2009

Contents

1 Definition of a ring of completion homomorphisms

theory *HomGroupCompletion*
imports
~~/src/HOL/Algebra/Ring

begin

2 Definition of completion functions and some related lemmas

constdefs
completion :: [('*a*, '*c*) *monoid-scheme*, ('*b*, '*d*) *monoid-scheme*, ('*a* => '*b*)] => ('*a* => '*b*)
completion G G' f == (%*x*. *if x* ∈ *carrier G then f x else one G'*)

lemma *completion-in-funcset*: (!!*x*. *x* ∈ *carrier G* ==> *f x* ∈ *carrier G'*) ==> (*completion G G' f*) ∈ *carrier G* −> *carrier G'*
by (*simp add*: *Pi-def completion-def*)

lemma *completion-in-hom*: **includes** *group-hom G G' h* **shows** *completion G G' h* ∈ *hom G G'*
by (*unfold completion-def hom-def Pi-def*, *auto*)

lemma *completion-apply-carrier* [*simp*]: *x* ∈ *carrier G* ==> *completion G G' h x* = *h x*
by (*simp add*: *completion-def*)

lemma *completion-apply-not-carrier* [*simp*]: *x* ∉ *carrier G* ==> *completion G G' h x* = *one G'*
by (*simp add*: *completion-def*)

lemma *completion-ext*: (!!*x*. *x* ∈ *carrier G* ==> *h x* = *g x*) ==> (*completion G G' h*) = (*completion G G' g*)
by (*simp add*: *expand-fun-eq Pi-def completion-def*)

lemma *inj-on-completion-eq*: *inj-on* (*completion G G' h*) (*carrier G*) = *inj-on h* (*carrier G*)
by (*unfold inj-on-def*, *simp*)

constdefs
completion-fun :: [('*a*, '*c*) *monoid-scheme*, ('*b*, '*d*) *monoid-scheme*] => ('*a* => '*b*) *set*
completion-fun G G' == {*f*. *f* = (%*x*. *if x* ∈ *carrier G then f x else one G'*)}

constdefs
completion-fun2 :: [('*a*, '*c*) *monoid-scheme*, ('*b*, '*d*) *monoid-scheme*] => ('*a* => '*b*) *set*
completion-fun2 G G' == {*f*. ∃ *g*. *f* = *completion G G' g*}

lemma *f-in-completion-fun2-f-completion*: *f* ∈ *completion-fun2 G G'* ==> *f* = *completion G G' f*
by (*unfold completion-fun2-def*, *unfold completion-def*, *auto simp add*: *if-def*)

lemma *completion-in-completion-fun*: *completion G G′ h* ∈ *completion-fun G G′*
by (*unfold completion-fun-def completion-def*) (*simp add*: *if-def*)

lemma *completion-in-completion-fun2*: **shows** *completion G G′ h* ∈ *completion-fun2 G G′*
by (*unfold completion-fun2-def*) *auto*

lemma *completion-fun-completion-fun2*: *completion-fun G G′* = *completion-fun2 G G′*
by (*unfold completion-fun-def completion-fun2-def completion-def*) (*auto simp add*: *if-def*)

lemma *completion-id-in-completion-fun*: **shows** *completion G G′ id* ∈ *completion-fun G G′*
by (*unfold completion-fun-def completion-def*, *auto simp add*: *expand-fun-eq*)

lemma *completion-closed2*: **assumes** *h*: *h* ∈ *completion-fun2 G G′* **and** *x*: *x* ∉ *carrier G* **shows** *h x* = *one G′*
using *prems*
by (*unfold completion-fun2-def completion-def*, *auto*)

2.1 Homomorphisms defined as completions

constdefs
hom-completion :: [(′*a*, ′*c*) *monoid-scheme*, (′*b*, ′*d*) *monoid-scheme*] => (′*a* => ′*b*)*set*
hom-completion G G′ == {*h*. *h* ∈ *completion-fun2 G G′* & *h* ∈ *hom G G′*}

lemma *hom-completionI*: **assumes** *h* ∈ *completion-fun2 G G′* **and** *h* ∈ *hom G G′* **shows** *h* ∈ *hom-completion G G′*
by (*unfold hom-completion-def*, *simp add*: *prems*)

lemma *hom-completion-is-hom*: **assumes** *f*: *f* ∈ *hom-completion G G′* **shows** *f* ∈ *hom G G′*
using *f* **by** (*unfold hom-completion-def*, *simp*)

lemma *hom-completion-mult*: **assumes** *h* ∈ *hom-completion G G′* **and** *x* ∈ *carrier G* **and** *y* ∈ *carrier G*
shows *h* (*mult G x y*) = *mult G′* (*h x*) (*h y*)
using *prems* **by** (*simp add*: *hom-completion-is-hom hom-mult*)

lemma *hom-completion-closed*: **assumes** *h*: *h* ∈ *hom-completion G G′* **and** *x*: *x* ∈ *carrier G* **shows** *h x* ∈ *carrier G′*
using *h* **and** *x* **by** (*unfold hom-completion-def hom-def Pi-def*, *simp*)

lemma *hom-completion-one*[*simp*]: **includes** *group G* + *group G′*
assumes *h*: *h* ∈ *hom-completion G G′* **shows** *h* (*one G*) = *one G′*
using *h* **and** *group-hom.hom-one* [*of G G′ h*] **and** *prems*
by (*unfold hom-completion-def group-hom-def group-hom-axioms-def*, *simp*)

lemma *comp-sum*: **includes** *group G*
  assumes *h*: $h \in$ *hom G G* **and** *h'*: $h' \in$ *hom G G* **and** *x*: $x \in$ *carrier G* **and** *y*: $y \in$ *carrier G*
  shows h' (h (*mult G x y*)) = *mult G* (h' (h x)) (h' (h y))
proof −
  have h (*mult G x y*) = *mult G* (h x) (h y)
    by (*rule hom-mult* [*of h G G x y*], *simp-all add*: *h x y*)
  then have h' (h (*mult G x y*)) = h' (*mult G* (h x) (h y)) **by** *simp*
  also from *h h' x y* **have** ... = *mult G* (h' (h x)) (h' (h y))
    by (*intro hom-mult*, *unfold hom-def Pi-def*, *simp-all*)
  finally show *?thesis* **by** *simp*
qed

lemma *comp-is-hom*: **includes** *group G*
  assumes *h*: $h \in$ *hom G G* **and** *h'*: $h' \in$ *hom G G*
  shows $h' \circ h \in$ *hom G G*
  using *h h'* **by** (*unfold hom-def Pi-def*, *simp*)

Usual composition *op* ∘ of completion homomorphisms is closed

lemma *hom-completion-comp*: **includes** *group G*
  assumes $f \in$ *hom-completion G G* **and** $g \in$ *hom-completion G G*
  shows $f \circ g \in$ *hom-completion G G*
proof −
  from *prems* **show** *?thesis*
    apply (*unfold hom-completion-def hom-def Pi-def*, *auto*)
    apply (*unfold completion-fun2-def completion-def*, *simp*)
    apply (*intro exI* [*of - f* ∘ *g*])
    apply (*auto simp add*: *expand-fun-eq*)
    done
qed

2.2 Completion homomorphisms with usual composition form a monoid

The underlying algebraic structures in our development, except otherwise stated, will be commutative groups or differential groups

lemma (**in** *comm-group*) *hom-completion-monoid*:
  shows *monoid* (| *carrier* = *hom-completion G G*, *mult* = *op o*, *one* = (λx. *if* $x \in$ *carrier G then id x else* **1**) |)
  (**is** *monoid ?H-CO*)
proof (*intro monoidI*)
  fix *x y*
  assume *x*: $x \in$ *carrier ?H-CO* **and** *y*: $y \in$ *carrier ?H-CO*
  from *prems* **show** $x \otimes_{?H\text{-}CO} y \in$ *carrier ?H-CO*
    by (*simp*, *intro hom-completion-comp*)(*simp-all add*: *comm-group-def*)
next
  show $\mathbf{1}_{?H\text{-}CO} \in$ *carrier ?H-CO*

by (*unfold hom-completion-def hom-def completion-fun2-def completion-def*)(*auto simp add*: *Pi-def*)
next
fix x y z

show $x \otimes_{?H\text{-}CO} y \otimes_{?H\text{-}CO} z = x \otimes_{?H\text{-}CO} (y \otimes_{?H\text{-}CO} z)$
by (*simp*) (*rule sym*, *rule o-assoc*)
next
fix x
assume x: $x \in$ *carrier ?H-CO*
from *prems* **show** $\mathbf{1}_{?H\text{-}CO} \otimes_{?H\text{-}CO} x = x$
by (*unfold hom-completion-def completion-fun2-def completion-def hom-def Pi-def*, *auto simp add*: *expand-fun-eq*)
next
fix x
assume x: $x \in$ *carrier ?H-CO*
from *prems* **show** $x \otimes_{?H\text{-}CO} \mathbf{1}_{?H\text{-}CO} = x$
by (*unfold hom-completion-def completion-fun2-def*, *auto simp add*: *expand-fun-eq*) (*intro group-hom.hom-one*, *unfold group-hom-def group-hom-axioms-def hom-def Pi-def comm-group-def*, *simp*)
qed

Homomorphisms, without the completion condition, are also a monoid with usual composition and the identity

lemma (**in** *group*) *hom-group-monoid*:
shows *monoid* (| *carrier* = *hom G G*, *mult* = *op o*, *one* = *id* |)
(**is** *monoid ?HOM*)
proof (*intro monoidI*)
fix x y
assume $x \in$ *carrier ?HOM* **and** $y \in$ *carrier ?HOM*
from *prems* **show** $x \otimes_{?HOM} y \in$ *carrier ?HOM* **by** (*simp add*: *Pi-def hom-def*)
next
show $\mathbf{1}_{?HOM} \in$ *carrier ?HOM* **by** (*simp add*: *hom-def Pi-def*)
next
fix x y z
assume $x \in$ *carrier ?HOM* **and** $y \in$ *carrier ?HOM* **and** $z \in$ *carrier ?HOM*
show $x \otimes_{?HOM} y \otimes_{?HOM} z = x \otimes_{?HOM} (y \otimes_{?HOM} z)$ **using** *o-assoc* [*of* x y z] **by** *simp*
next
fix x
assume $x \in$ *carrier ?HOM*
show $\mathbf{1}_{?HOM} \otimes_{?HOM} x = x$ **by** *simp*
next
fix x
assume $x \in$ *carrier ?HOM*
show $x \otimes_{?HOM} \mathbf{1}_{?HOM} = x$ **by** *simp*
qed

2.3 Preliminary facts about addition of homomorphisms

lemma *homI*:
  assumes *closed*: $\bigwedge x.\ x \in carrier\ G \Longrightarrow f\ x \in carrier\ H$
  and *mult*: $\bigwedge x\ y.\ [\![x \in carrier\ G;\ y \in carrier\ G]\!] \Longrightarrow f\ (x \otimes_G y) = f\ x \otimes_H f\ y$
  shows $f \in hom\ G\ H$ **by** (*unfold hom-def*) (*simp add*: *Pi-def closed mult*)

The operation we are going to use as addition for homomorphisms is based on the multiplicative operation of the underlying algebraic structures

The three following lemmas show how we can define the addition of homomorphisms in different ways with satisfactory result

lemma (**in** *comm-group*) *hom-mult-is-hom*: **assumes** F: $f \in hom\ G\ G$ **and** G: $g \in hom\ G\ G$ **shows** $(\lambda x.\ f\ x \otimes g\ x) \in hom\ G\ G$
proof (*rule homI*)
  fix x
  assume X: $x \in carrier\ G$
  from *prems* **show** $f\ x \otimes g\ x \in carrier\ G$ **by** (*intro m-closed*, *simp-all only*: *hom-closed*)
next
  fix $x\ y$
  assume X: $x \in carrier\ G$ **and** Y: $y \in carrier\ G$
  from *prems* **have** $f\ (x \otimes y) \otimes g\ (x \otimes y) = f\ x \otimes f\ y \otimes (g\ x \otimes g\ y)$ **by** (*unfold hom-def*, *simp add*: *m-ac*)
  with *prems* **show** $f\ (x \otimes y) \otimes g\ (x \otimes y) = f\ x \otimes g\ x \otimes (f\ y \otimes g\ y)$ **by** (*simp add*: *m-ac hom-closed*)
qed

lemma (**in** *comm-group*) *hom-mult-is-hom-rest*:
  assumes f: $f \in hom\ G\ G$ **and** g: $g \in hom\ G\ G$
  shows $(\lambda x \in carrier\ G.\ f\ x \otimes g\ x) \in hom\ G\ G$ (**is** $?fg \in$ -)
proof (*rule homI*)
  fix x **assume** $x \in carrier\ G$
  with $f\ g$ **show** $?fg\ x \in carrier\ G$ **by** (*simp add*: *hom-closed*)
next
  fix $x\ y$ **assume** $x \in carrier\ G\ y \in carrier\ G$
  with $f\ g$ **show** $?fg\ (x \otimes y) = ?fg\ x \otimes ?fg\ y$ **by** (*simp add*: *hom-closed hom-mult m-ac*)
qed

lemma (**in** *comm-group*) *hom-mult-is-hom-completion*:
  assumes f: $f \in hom\ G\ G$ **and** g: $g \in hom\ G\ G$
  shows $(\lambda x.\ if\ x \in carrier\ G\ then\ f\ x \otimes g\ x\ else\ \mathbf{1}) \in hom\ G\ G$
  (**is** $?fg \in$ -)
  apply (*rule homI*)
  using $f\ g$ **apply** (*simp add*: *hom-closed*)
  using $f\ g$ **apply** (*simp add*: *hom-mult m-ac hom-closed*)
  done

The inverse for the addition of homomorphisms will be given by the $\lambda x.\ inv$

f x operation

lemma (**in** *comm-group*) *hom-inv-is-hom*: **assumes** *f*: $f \in hom\ G\ G$ **shows** $(\lambda x.\ inv\ f\ x) \in hom\ G\ G$
proof (*unfold hom-def*, *simp*, *intro conjI*)
  from *f* **show** $(\lambda x.\ inv\ f\ x) \in carrier\ G \to carrier\ G$ **by** (*unfold Pi-def hom-def*, *auto*)
next
  show $\forall x{\in}carrier\ G.\ \forall y{\in}carrier\ G.\ inv\ f\ (x \otimes y) = inv\ f\ x \otimes inv\ f\ y$
  proof (*intro ballI*)
    fix *x y* **assume** *x*: $x \in carrier\ G$ **and** *y*: $y \in carrier\ G$
    from *prems* **show** $inv\ f\ (x \otimes y) = inv\ f\ x \otimes inv\ f\ y$
    proof −
      from *prems* **have** $f\ (x \otimes y) = f\ x \otimes f\ y$ **by** (*intro hom-mult*, *simp-all*)
      then have $inv\ f\ (x \otimes y) = inv\ (f\ x \otimes f\ y)$ **by** *simp*
      also from *prems* **have** $\ldots = inv\ (f\ y) \otimes inv\ (f\ x)$
        by (*intro inv-mult-group*) (*simp-all add*: *hom-closed*)
      also from *prems* **have** $\ldots = inv\ (f\ x) \otimes inv\ (f\ y)$
        by (*intro m-comm*) (*simp-all add*: *inv-closed hom-closed*)
      finally show *?thesis* **by** *simp*
    qed
  qed
qed

Lemma $?f \in hom\ G\ G \Longrightarrow (\lambda x.\ inv\ ?f\ x) \in hom\ G\ G$ proves that the multiplicative inverse of the underlying structure preserves the homomorphism definition

locale *group-end* = *group-hom G G h*

Due to the partial definitions of domains, it would not be possible to prove that $h \circ (\lambda x.\ inv\ h\ x) = (\lambda x.\ \mathbf{1})$; the closer fact that can be proven is $h \circ (\lambda x.\ inv\ h\ x) = (\lambda x{\in}carrier\ G.\ \mathbf{1})$;

lemma (**in** *comm-group*) *hom-completion-inv-is-hom-completion*:
  assumes $f \in hom\text{-}completion\ G\ G$
  shows $(\lambda x.\ if\ x \in carrier\ G\ then\ inv\ f\ x\ else\ \mathbf{1}) \in hom\text{-}completion\ G\ G$
proof (*unfold hom-completion-def completion-fun2-def completion-def*, *simp*, *intro conjI*)
  show $\exists g.\ (\lambda g.\ if\ g \in carrier\ G\ then\ inv\ f\ g\ else\ \mathbf{1}) = (\lambda x.\ if\ x \in carrier\ G\ then\ g\ x\ else\ \mathbf{1})$
    by (*rule exI* [*of* - $\lambda x.\ inv\ (f\ x)$], *simp*)
next
  show $(\lambda x.\ if\ x \in carrier\ G\ then\ inv\ f\ x\ else\ \mathbf{1}) \in hom\ G\ G$
  proof (*unfold hom-def*, *simp*, *intro conjI*)
     from *prems* **show** $(\lambda x.\ if\ x \in carrier\ G\ then\ inv\ f\ x\ else\ \mathbf{1}) \in carrier\ G \to carrier\ G$
        by (*unfold Pi-def hom-completion-def completion-fun2-def completion-def hom-def Pi-def*, *auto*)
  next
    show $\forall x{\in}carrier\ G.\ \forall y{\in}carrier\ G.\ inv\ f\ (x \otimes y) = inv\ f\ x \otimes inv\ f\ y$

proof (*intro ballI*)
fix *x y*
assume $x \in$ *carrier G* **and** $y \in$ *carrier G*
show *inv f* $(x \otimes y) =$ *inv f x* $\otimes$ *inv f y*
proof −
from *prems* **have** $f\ (x \otimes y) = f\ x \otimes f\ y$ **by** (*intro hom-mult*, *unfold hom-completion-def*, *simp-all*)
then have *inv f* $(x \otimes y) =$ *inv* $(f\ x \otimes f\ y)$ **by** *simp*
also from *prems* **have** ... = *inv* $(f\ y) \otimes$ *inv* $(f\ x)$
by (*intro inv-mult-group*) (*unfold hom-completion-def hom-def Pi-def*, *simp-all*)
also from *prems* **have** ... = *inv* $(f\ x) \otimes$ *inv* $(f\ y)$
by (*intro m-comm*)(*unfold hom-completion-def hom-def Pi-def*, *simp-all*)
finally show *?thesis* **by** *simp*
qed
qed
qed
qed

lemma (**in** *comm-group*) *hom-completion-mult-inv-is-hom-completion*:
assumes $f \in$ *hom-completion G G*
shows $\exists\, g \in$*hom-completion G G*. (λx. *if* $x \in$ *carrier G then g x* $\otimes$ *f x else* **1**) = (λx. **1**)
proof (*intro bexI* [*of* - (λx. *if* $x \in$ *carrier G then inv* $(f\ x)$ *else* **1**)])
from *prems* **show** (λg. *if* $g \in$ *carrier G then inv f g else* **1**) $\in$ *hom-completion G G*
by (*intro hom-completion-inv-is-hom-completion*)
next
from *prems* **show** (λx. *if* $x \in$ *carrier G then* (*if* $x \in$ *carrier G then inv f x else* **1**) $\otimes$ *f x else* **1**) = (λx. **1**)
by (*unfold hom-completion-def hom-def Pi-def*, *auto simp add*: *expand-fun-eq*)
qed

2.4 Completion homomorphisms are a commutative group with the underlying operation

lemma (**in** *comm-group*) *hom-completion-mult-comm-group*:
shows *comm-group* (|*carrier* = *hom-completion G G*, *mult* = λf. λg. (λx. *if* $x \in$ *carrier G then f x* $\otimes$ *g x else* **1**),
one = (λx. *if* $x \in$ *carrier G then* **1** *else* **1**)|)
(**is** *comm-group ?H-CO*)
proof (*intro comm-groupI*)
fix *x y*
assume $x \in$ *carrier ?H-CO* **and** $y \in$ *carrier ?H-CO*
from *prems* **show** $x \otimes_{?H\text{-}CO} y \in$ *carrier ?H-CO*
by (*unfold hom-completion-def completion-fun2-def completion-def*, *auto simp add*: *hom-mult-is-hom-completion*)
next
show $\mathbf{1}_{?H\text{-}CO} \in$ *carrier ?H-CO*

by (*unfold hom-completion-def completion-fun2-def completion-def hom-def Pi-def expand-fun-eq*, *auto*)
next
fix $x\ y\ z$
assume $x \in$ *carrier ?H-CO* **and** $y \in$ *carrier ?H-CO* **and** $z \in$ *carrier ?H-CO*
from *prems* **show** $x \otimes_{?H\text{-}CO} y \otimes_{?H\text{-}CO} z = x \otimes_{?H\text{-}CO} (y \otimes_{?H\text{-}CO} z)$
by (*unfold hom-completion-def completion-fun2-def completion-def expand-fun-eq*, *auto simp add*: *hom-def Pi-def m-assoc*)
next
fix $x\ y$
assume $x \in$ *carrier ?H-CO* **and** $y \in$ *carrier ?H-CO*
from *prems* **show** $x \otimes_{?H\text{-}CO} y = y \otimes_{?H\text{-}CO} x$
by (*unfold comm-monoid-axioms-def expand-fun-eq hom-completion-def hom-def Pi-def*, *simp add*: *m-comm*)
next
fix x
assume $x \in$ *carrier ?H-CO*
from *prems* **show** $\mathbf{1}_{?H\text{-}CO} \otimes_{?H\text{-}CO} x = x$
by (*unfold hom-completion-def completion-fun2-def completion-def expand-fun-eq group-axioms-def hom-def Pi-def*, *auto*)
next
fix x
assume $x \in$ *carrier ?H-CO*

from *prems* **and** *hom-completion-mult-inv-is-hom-completion* [*of x*] **show** $\exists\, y \in$ *carrier ?H-CO*. $y \otimes_{?H\text{-}CO} x = \mathbf{1}_{\ ?H\text{-}CO}$
by *simp*
qed

lemma (**in** *comm-group*) *hom-completion-mult-comm-group2*:
shows *comm-group* (| *carrier* = *hom-completion G G*, *mult* = $\lambda f.\ \lambda g.\ (\lambda x.$ *if* $x \in$ *carrier G then* $f\ x \otimes g\ x$ *else* $\mathbf{1}$), *one* = $(\lambda x.\ \mathbf{1})$|)
proof −
from *prems* **have** *comm-group* (|*carrier* = *hom-completion G G*, *mult* = $\lambda f\ g\ x$. *if* $x \in$ *carrier G then* $f\ x \otimes g\ x$ *else* $\mathbf{1}$,
one = λx. *if* $x \in$ *carrier G then* $\mathbf{1}$ *else* $\mathbf{1}$|) **by** (*intro hom-completion-mult-comm-group*)
then show *?thesis* **by** *simp*
qed

lemma (**in** *comm-group*) *hom-completion-mult-comm-monoid*:
includes *comm-group G*
shows *comm-monoid* (| *carrier* = *hom-completion G G*, *mult* = $\lambda f.\ \lambda g.\ (\lambda x.$ *if* $x \in$ *carrier G then* $f\ x \otimes g\ x$ *else* $\mathbf{1}$), *one* = $(\lambda x.\ \mathbf{1})$|)
proof −
from *prems* **have** *comm-group* (|*carrier* = *hom-completion G G*, *mult* = $\lambda f\ g\ x$. *if* $x \in$ *carrier G then* $f\ x \otimes g\ x$ *else* $\mathbf{1}$,
one = λx. *if* $x \in$ *carrier G then* $\mathbf{1}$ *else* $\mathbf{1}$|) **by** (*intro hom-completion-mult-comm-group*)

then have *comm-group* (|*carrier* = *hom-completion G G*, *mult* = $\lambda f\ g\ x$. *if* $x \in$ *carrier G then f x* $\otimes$ *g x else* **1**,
one = λx. **1**|) **by** *simp*
then show *?thesis* **by** (*unfold comm-group-def comm-monoid-def*, *simp*)
qed

2.5 Endomorphisms with suitable operations form a ring

The distributive law is proved first

lemma (**in** *comm-group*) *r-mult-dist-add*: **assumes***f* $\in$ *hom-completion G G* **and** *g* $\in$ *hom-completion G G* **and** *h* $\in$ *hom-completion G G*
shows (λx. *if x* $\in$ *carrier G then f x* $\otimes$ *g x else* **1**) *o h* = (λx. *if x* $\in$ *carrier G then* (*f o h*) *x* $\otimes$ (*g o h*) *x else* **1**)
proof (*simp add*: *expand-fun-eq*, *intro allI impI conjI*)
fix *x*
assume *x* $\in$ *carrier G* **and** *h x* $\notin$ *carrier G*
from *prems* **show** **1** = *f* (*h x*) $\otimes$ *g* (*h x*)
by (*unfold hom-completion-def completion-fun2-def completion-def*, *auto*)
next
fix *x*
assume *x* $\notin$ *carrier G* **and** *h x* $\in$ *carrier G*
show *f* (*h x*) $\otimes$ *g* (*h x*) = **1**
proof −
from *prems* **have** *f* (*h x*) = **1** **and** *g* (*h x*) = **1**
apply (*unfold hom-completion-def completion-fun2-def completion-def*, *auto*)
apply (*intro group-hom.hom-one*, *unfold group-hom-def group-hom-axioms-def comm-group-def hom-completion-def hom-def Pi-def*, *auto*)+
done
then show *?thesis* **by** *simp*
qed
qed

lemma (**in** *comm-group*) *l-mult-dist-add*: **assumes***f* $\in$ *hom-completion G G* **and** *g* $\in$ *hom-completion G G* **and** *h* $\in$ *hom-completion G G*
shows *h* $\circ$ (λx. *if x* $\in$ *carrier G then f x* $\otimes$ *g x else* **1**) = (λx. *if x* $\in$ *carrier G then* (*h o f*) *x* $\otimes$ (*h o g*) *x else* **1**)
proof −
from *prems* **show** *?thesis*
apply (*simp add*: *expand-fun-eq*, *intro allI impI conjI*)
apply (*intro hom-mult*, *unfold hom-completion-def hom-def Pi-def*, *auto*)
apply (*intro group-hom.hom-one*)
apply (*unfold comm-group-def group-hom-def group-hom-axioms-def hom-def Pi-def*, *simp*)
done
qed

Endomorphisms with the previous operations form a ring

lemma (**in** *comm-group*) *hom-completion-ring*:

shows *ring* (| *carrier* = *hom-completion G G*, *mult* = *op o*, *one* = (λx. *if* $x \in$ *carrier G then id x else* **1**),
zero = (λx. *if* $x \in$ *carrier G then* **1** *else* **1**), *add* = λf. λg. (λx. *if* $x \in$ *carrier G then f x* $\otimes$ *g x else* **1**)|)
proof (*rule ringI*, *unfold abelian-group-def abelian-monoid-def abelian-group-axioms-def*,

simp-all add: *hom-completion-mult-comm-monoid hom-completion-mult-comm-group*)
show *monoid* (| *carrier* = *hom-completion G G*, *mult* = *op o*, *one* = λx. *if* $x \in$ *carrier G then id x else* **1**, *zero* = λx. **1**,
add = $\lambda f\ g\ x$. *if* $x \in$ *carrier G then f x* $\otimes$ *g x else* **1** |)
proof −
from *comm-group.hom-completion-monoid* **and** *prems*
have *monoid* (|*carrier* = *hom-completion G G*, *mult* = *op* $\circ$, *one* = λx. *if* $x \in$ *carrier G then id x else* **1**|)
by (*unfold comm-group-def*, *auto*)
then show *?thesis* **by** (*unfold monoid-def*, *simp*)
qed
next
show *comm-group* (|*carrier* = *hom-completion G G*, *mult* = $\lambda f\ g\ x$. *if* $x \in$ *carrier G then f x* $\otimes$ *g x else* **1**, *one* = λx. **1**|)
by (*intro hom-completion-mult-comm-group2*)
next
show $\bigwedge x\ y\ z$. $[\![x \in$ *hom-completion G G*; $y \in$ *hom-completion G G*; $z \in$ *hom-completion G G*$]\!]$
$\Longrightarrow$ (λxa. *if xa* $\in$ *carrier G then x xa* $\otimes$ *y xa else* **1**) $\circ$ *z* = (λxa. *if xa* $\in$ *carrier G then* ($x \circ z$) *xa* $\otimes$ ($y \circ z$) *xa else* **1**)
by (*erule r-mult-dist-add*) (*assumption+*)
next
show $\bigwedge x\ y\ z$. $[\![$ $x \in$ *hom-completion G G*; $y \in$ *hom-completion G G*; $z \in$ *hom-completion G G* $]\!]$
$\Longrightarrow$ *z o* (λxa. *if xa* $\in$ *carrier G then x xa* $\otimes$ *y xa else* **1**) = (λxa. *if xa* $\in$ *carrier G then* (*z o x*) *xa* $\otimes$ (*z o y*) *xa else* **1**)
by (*erule l-mult-dist-add*)(*assumption+*)
qed

locale *hom-completion-ring* = *comm-group G* + *ring R* +
assumes *R* = (| *carrier* = *hom-completion G G*, *mult* = *op o*,
one = (λx. *if* $x \in$ *carrier G then id x else* **1**),
zero = (λx. *if* $x \in$ *carrier G then one G else* **1**),
add = λf. λg. (λx. *if* $x \in$ *carrier G then f x* $\otimes$ *g x else* **1**)|)

Some examples where it is shown the usefulness of the previous proofs

lemma (**in** *hom-completion-ring*) *r-dist-minus*:
[|$f \in$ *carrier R*; $g \in$ *carrier R*; $h \in$ *carrier R*|]
==> ($f \ominus_2 g$) $\otimes_2$ h = ($f \otimes_2 h$) $\ominus_2$ ($g \otimes_2 h$) **by** *algebra*

lemma (**in** *hom-completion-ring*) *sublemma*:
[| $f \in$ *carrier R*; $h \in$ *carrier R*; $f \otimes_2 h = h$ |] ==> ($\mathbf{1}_2 \ominus_2 f$) $\otimes_2$ h = $\mathbf{0}_2$ **by** *algebra*

2.6 Definition of differential group

According to Section 2.3 in Aransay's memoir, in the following we will be dealing with ungraded algebraic structures.

The Basic Perturbation Lemma is usually stated in terms of differential structures; these include diferential groups as well as chain complexes.

Moreover, chain complexes can be defined in terms of differential groups (more concretely, as indexed collections of differential groups).

The proof of the Basic Perturbation Lemma does not include any reference to graded structures or proof obligations derived from the degree information.

Thus, we preferred to state and prove the Basic Perturbation Lemma in terms of ungrades structures (differential and abelian groups), for the sake of simplicity, and avoid implementing and dealing with graded structures (chain complexes and graded groups).

record *'a diff-group* = *'a monoid* +
  diff :: *'a* ⇒ *'a* (*differı 81*)

locale *diff-group* = *comm-group D* +
  assumes *diff-hom* : *differ* ∈ *hom-completion D D*
  and *diff-nilpot* : *differ* ∘ *differ* = (λ*x*. **1**)

lemma *diff-groupI*:
  includes *struct D*
  assumes *m-closed*:
  !!*x y*. [| *x* ∈ *carrier D*; *y* ∈ *carrier D* |] ==> *x* ⊗ *y* ∈ *carrier D*
  and *one-closed*: **1** ∈ *carrier D*
  and *m-assoc*:
  !!*x y z*. [| *x* ∈ *carrier D*; *y* ∈ *carrier D*; *z* ∈ *carrier D* |] ==> (*x* ⊗ *y*) ⊗ *z* = *x* ⊗ (*y* ⊗ *z*)
  and *m-comm*:
  !!*x y*. [| *x* ∈ *carrier D*; *y* ∈ *carrier D* |] ==> *x* ⊗ *y* = *y* ⊗ *x*
  and *l-one*: !!*x*. *x* ∈ *carrier D* ==> **1** ⊗ *x* = *x*
  and *l-inv-ex*: !!*x*. *x* ∈ *carrier D* ==> ∃ *y* ∈ *carrier D*. *y* ⊗ *x* = **1**
  and *diff-hom*: *differ* ∈ *hom-completion D D*
  and *diff-nilpot*: !!*x*. (*differ*) ((*differ*) *x*) = **1**
  shows *diff-group D*
  using *prems* **by** (*unfold diff-group-def diff-group-axioms-def comm-group-def group-def group-axioms-def comm-monoid-def*
    comm-monoid-axioms-def Units-def monoid-def, *auto simp add*: *expand-fun-eq*)

2.7 Definition of homomorphisms between differential groups

locale *hom-completion-diff* = *diff-group C* + *diff-group D* + *var f* +
  assumes *f-hom-completion*: *f* ∈ *hom-completion C D*
  and *f-coherent*: *f* ∘ differ_1 = differ_2 ∘ *f*

constdefs (**structure** C **and** D)
hom-diff :: - => - => ('a => 'b) *set*
hom-diff C D == $\{f.\ f \in$ *hom-completion* C D & $(f \circ (differ_C) = (differ_D) \circ f)\}$

lemma *hom-diff-is-hom-completion*: **assumes** h: $h \in$ *hom-diff* C D
shows $h \in$ *hom-completion* C D
using h **by** (*unfold hom-diff-def*, *simp*)

lemma *hom-diff-closed*: **assumes** h: $h \in$ *hom-diff* C D **and** x: $x \in$ *carrier* C **shows** $h\ x \in$ *carrier* D
using h **and** x **by** (*unfold hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)

lemma *hom-diff-mult*: **assumes** h: $h \in$ *hom-diff* C D **and** x: $x \in$ *carrier* C **and** y: $y \in$ *carrier* C **shows** $h\ (x \otimes_C y) = h\ (x) \otimes_D h\ (y)$
using *hom-completion-mult* [*of* h C D x y] **and** h **and** x **and** y **by** (*unfold hom-diff-def*, *simp*)

lemma *hom-diff-coherent*: **assumes** h: $h \in$ *hom-diff* C D **shows** $h \circ differ_C = differ_D \circ h$
using h **by** (*unfold hom-diff-def*, *simp*)

lemma (**in** *diff-group*) *hom-diff-comp-closed*: **assumes** $f \in$ *hom-diff* D D **and** $g \in$ *hom-diff* D D **shows** $g \circ f \in$ *hom-diff* D D
proof −
from *prems* **show** $g \circ f \in$ *hom-diff* D D
proof (*unfold hom-diff-def*, *auto*)
from *prems* **show** $g \circ f \in$ *hom-completion* D D
by (*intro hom-completion-comp*, *unfold comm-group-def group-def group-axioms-def hom-diff-def*, *simp-all*)
from *prems* **show** $g \circ f \circ differ = differ \circ (g \circ f)$
proof −
have $g \circ f \circ differ = g \circ (f \circ differ)$
by (*rule sym*, *rule o-assoc*)
also from *prems* **have** ... $= g \circ (differ \circ f)$
by (*unfold hom-diff-def*, *auto*)
also have ... $= (g \circ differ) \circ f$
by (*rule o-assoc*)
also from *prems* **have** ... $= (differ \circ g) \circ f$
by (*unfold hom-diff-def*, *auto*)
also have ... $= differ \circ (g \circ f)$
by (*rule sym*, *rule o-assoc*)
finally show *?thesis* **by** *simp*
qed
qed
qed

lemma (**in** *diff-group*) *hom-diff-monoid*:

shows *monoid* (|*carrier* = *hom-diff D D*, *mult* = *op o*, *one* = (λx. *if* $x \in$ *carrier D then id x else* **1**)|)
(**is** *monoid ?DIFF*)
proof (*intro monoidI*)
fix *x y*
assume $x \in$ *carrier ?DIFF* $y \in$ *carrier ?DIFF*
then show $x \otimes_{?DIFF} y \in$ *carrier ?DIFF* **by** *simp* (*rule hom-diff-comp-closed*)
next
show $\mathbf{1}_{?DIFF} \in$ *carrier ?DIFF*
proof (*simp*, *unfold hom-diff-def hom-completion-def completion-fun2-def completion-def hom-def Pi-def diff-group-def*, *auto*)
from *prems* **show** (λx. *if* $x \in$ *carrier D then id x else* **1**) $\circ$ *differ* = *differ* $\circ$ (λx. *if* $x \in$ *carrier D then id x else* **1**)
apply (*unfold diff-group-def diff-group-axioms-def hom-completion-def completion-fun2-def completion-def hom-def Pi-def*)
apply (*auto simp add*: *expand-fun-eq*)
apply (*rule sym*, *intro group-hom.hom-one*, *unfold group-hom-def group-hom-axioms-def comm-group-def hom-def Pi-def*, *simp*)
done
qed
next
fix *x y z*
assume $x \in$ *carrier ?DIFF* **and** $y \in$ *carrier ?DIFF* **and** $z \in$ *carrier ?DIFF*
show $x \otimes_{?DIFF} y \otimes_{?DIFF} z = x \otimes_{?DIFF} (y \otimes_{?DIFF} z)$ **by** (*simp add*: *o-assoc*)
next
fix *x*
assume $x \in$ *carrier ?DIFF* **from** *prems* **show** $\mathbf{1}_{?DIFF} \otimes_{?DIFF} x = x$
by (*unfold hom-diff-def hom-completion-def hom-def Pi-def completion-fun2-def completion-def*, *auto simp add*: *expand-fun-eq*)
next
fix *x*
assume $x \in$ *carrier ?DIFF*
from *prems* **show** $x \otimes_{?DIFF} \mathbf{1}_{?DIFF} = x$
by (*unfold hom-diff-def hom-completion-def hom-def Pi-def completion-fun2-def completion-def*,*auto simp add*: *expand-fun-eq*)
(*rule group-hom.hom-one*, *unfold group-hom-def group-hom-axioms-def diff-group-def comm-group-def hom-def Pi-def*, *simp*)
qed

2.8 Completion homomorphisms between differential structures form a commutative group with the underlying operation

lemma (**in** *diff-group*) *hom-diff-mult-closed*: **assumes** $f \in$ *hom-diff D D* **and** $g \in$ *hom-diff D D*
shows (λx. *if* $x \in$ *carrier D then* $f\ x \otimes g\ x$ *else* **1**) $\in$ *hom-diff D D*
proof (*unfold hom-diff-def hom-completion-def completion-fun2-def completion-def*, *simp*, *intro conjI*)
show $\exists$ *ga*. (λx. *if* $x \in$ *carrier D then* $f\ x \otimes g\ x$ *else* **1**) = (λx. *if* $x \in$ *carrier D*

then ga x else **1**)
    by (*rule exI* [*of* - λ*x*. *f x* ⊗ *g x*], *simp*)
next
  from *prems* **show** (λ*x*. *if x* ∈ *carrier D then f x* ⊗ *g x else* **1**) ∈ *hom D D*
     by (*unfold hom-diff-def hom-completion-def hom-def Pi-def*, *auto simp add*: *m-ac*)
next
  show (λ*x*. *if x* ∈ *carrier D then f x* ⊗ *g x else* **1**) ∘ *differ* = *differ* ∘ (λ*x*. *if x* ∈ *carrier D then f x* ⊗ *g x else* **1**)
  proof (*rule ext*)
    fix *x*
    show ((λ*x*. *if x* ∈ *carrier D then f x* ⊗ *g x else* **1**) ∘ *differ*) *x* = (*differ* ∘ (λ*x*. *if x* ∈ *carrier D then f x* ⊗ *g x else* **1**)) *x*
    proof (*cases x* ∈ *carrier D*)
      case *True*
      from *prems* **show** *?thesis*
      by (*unfold diff-group-axioms-def hom-diff-def hom-completion-def completion-fun2-def diff-group-def completion-def hom-def Pi-def*)
      (*auto simp add*: *expand-fun-eq*)
    next
      case *False*
      from *prems* **show** *?thesis*
      by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def completion-fun2-def completion-def hom-def Pi-def*)
      (*auto simp add*: *expand-fun-eq*)
    qed
  qed
qed

lemma (**in** *diff-group*) *hom-diff-inv-def*: **assumes** *f* ∈ *hom-diff D D*
  shows (λ*x*. *if x* ∈ *carrier D then inv f x else* **1**) ∈ *hom-diff D D*
proof (*unfold hom-diff-def*, *auto*)
  from *prems* **show** (λ*x*. *if x* ∈ *carrier D then inv f x else* **1**) ∈ *hom-completion D D*
    by (*intro hom-completion-inv-is-hom-completion*, *unfold hom-diff-def*, *simp*)
next
  from *prems* **show** (λ*x*. *if x* ∈ *carrier D then inv f x else* **1**) ∘ *differ* = *differ* ∘ (λ*x*. *if x* ∈ *carrier D then inv f x else* **1**)
  proof (*simp add*: *expand-fun-eq*, *intro allI impI conjI*)
    fix *x*
    assume *x* ∈ *carrier D*
    show *inv f* ((*differ*) *x*) = (*differ*) (*inv f x*)
    proof −

     have (*inv f* ((*differ*) *x*) = (*differ*) (*inv f x*)) = (*inv f* ((*differ*) *x*) ⊗ *f* ((*differ*) *x*) = (*differ*) (*inv f x*) ⊗ *f* ((*differ*) *x*))
      proof (*rule sym*, *rule r-cancel*)
        from *prems* **show** *f* ((*differ*) *x*) ∈ *carrier D*
       by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def*

hom-def Pi-def, *simp*)
next
from *prems* **show** *inv f* ((*differ*) *x*) ∈ *carrier D*
by (*intro inv-closed*)(*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
next
from *prems* **show** (*differ*) (*inv f x*) ∈ *carrier D*
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
qed
also have *inv f* ((*differ*) *x*) ⊗ *f* ((*differ*) *x*) = (*differ*) (*inv f x*) ⊗ *f* ((*differ*) *x*)
proof −
have *l-h*: *inv f* ((*differ*) *x*) ⊗ *f* ((*differ*) *x*) = **1**
proof (*rule l-inv*)
from *prems* **show** *f* ((*differ*) *x*) ∈ *carrier D*
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
qed
have *r-h*: (*differ*) (*inv f x*) ⊗ *f* ((*differ*) *x*) = **1**
proof −
have (*differ*) (*inv f x*) ⊗ *f* ((*differ*) *x*) = (*differ*) (*inv f x*) ⊗ (*differ*) (*f x*)
proof −
from *prems* **have** *f* ((*differ*) *x*) = (*differ*) (*f x*)
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def hom-def Pi-def*, *simp add*: *expand-fun-eq*)
then show *?thesis* **by** *simp*
qed
also have . . . = (*differ*) (*inv f x* ⊗ *f x*)
proof (*rule sym*, *rule hom-mult*)
from *prems* **show** *differ* ∈ *hom D D*
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def*, *simp*)
next
from *prems* **show** *inv f x* ∈ *carrier D*
by (*intro inv-closed*, *unfold hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
next
from *prems* **show** *f x* ∈ *carrier D*
by (*unfold hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
qed
also have . . . = (*differ*) (**1**)
proof −
from *prems* **have** *inv f x* ⊗ *f x* = **1**
by (*intro l-inv*, *unfold hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
then show *?thesis* **by** *simp*
qed
also from *prems* **have** . . . = **1**

proof (*intro group-hom.hom-one*, *unfold diff-group-def comm-group-def group-hom-def group-hom-axioms-def*,
intro conjI, *simp-all*)
from *prems* **show** *differ* ∈ *hom D D*
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def*, *simp*)
qed
finally show *?thesis* **by** *simp*
qed
from *l-h* **and** *r-h* **show** *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
next

fix *x*
assume *x* ∈ *carrier D* **and** (*differ*) *x* ∉ *carrier D*

from *prems* **show** **1** = (*differ*) (*inv f x*)
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
next
fix *x*
assume *x* ∉ *carrier D*
from *prems* **show** *inv f* ((*differ*) *x*) = (*differ*) **1**
proof −
have *l-h*: *inv f* ((*differ*) *x*) = **1** **thm** *inv-one*
proof −
from *prems* **have** *inv f* ((*differ*) *x*) = *inv f* (**1**)
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def completion-fun2-def completion-def Pi-def*,
auto)
also have ... = *inv* **1**
proof −
from *prems* **have** *f* **1** = **1**
by (*intro group-hom.hom-one*)
(*unfold diff-group-def comm-group-def group-hom-def group-hom-axioms-def hom-diff-def hom-completion-def*, *auto*)
then show *?thesis* **by** *simp*
qed
also have ... = **1**
by (*rule inv-one*)
finally show *?thesis* **by** *simp*
qed
from *prems* **have** *r-h*: (*differ*) **1** = **1**
by (*intro group-hom.hom-one*)
(*unfold diff-group-def comm-group-def diff-group-axioms-def group-hom-def group-hom-axioms-def hom-diff-def hom-completion-def*,
auto)

from *r-h* **and** *l-h* **show** *?thesis* **by** *simp*
qed
next
show **1** = (*differ*) **1**
proof (*rule sym*, *intro group-hom.hom-one*)
from *prems* **show** *group-hom D D* (*differ*)
by (*unfold diff-group-def comm-group-def*
diff-group-axioms-def group-hom-def group-hom-axioms-def hom-diff-def hom-completion-def, *auto*)
qed
qed
qed

lemma (**in** *diff-group*) *hom-diff-inv*: **assumes** $f \in$ *hom-diff D D*
shows $\exists\, g \in$*hom-diff D D*. (λx. *if* $x \in$ *carrier D then g x* $\otimes$ *f x else* **1**) = (λx. **1**)
proof (*rule bexI* [*of* - (λx. *if* $x \in$ *carrier D then inv f x else* **1**)])
from *prems* **show** (λx. *if* $x \in$ *carrier D then* (*if* $x \in$ *carrier D then inv f x else* **1**) $\otimes$ *f x else* **1**) = (λx. **1**)
by (*auto simp add*: *expand-fun-eq*) (*rule l-inv*, *unfold hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
next
show (λx. *if* $x \in$ *carrier D then inv f x else* **1**) $\in$ *hom-diff D D* **by** (*rule hom-diff-inv-def*, *simp add*: *prems*)
qed

2.9 Differential homomorphisms form a commutative group with the underlying operation

lemma (**in** *diff-group*) *hom-diff-mult-comm-group*:
shows *comm-group* (|*carrier* = *hom-diff D D*, *mult* = λf. λg. (λx. *if* $x \in$ *carrier D then f x* $\otimes$ *g x else* **1**),
one = (λx. *if* $x \in$ *carrier D then* **1** *else* **1**)|)
(**is** *comm-group ?DIFF*)
proof (*intro comm-groupI*)
fix *x y*
assume $x \in$ *carrier ?DIFF* **and** $y \in$ *carrier ?DIFF*
then show $x \otimes_{?DIFF} y \in$ *carrier ?DIFF* **by** *simp* (*erule hom-diff-mult-closed*, *assumption*)
next
from *prems* **show** $\mathbf{1}_{?DIFF} \in$ *carrier ?DIFF*
proof (*simp*, *unfold hom-diff-def hom-completion-def hom-def Pi-def completion-fun2-def*, *auto*)
show $\exists\, g$. (λx. **1**) = *completion D D g* **by** (*rule exI* [*of* - λx. **1**], *unfold completion-def*, *simp*)
next
from *prems* **show** (λx. **1**) $\circ$ *differ* = *differ* $\circ$ (λx. **1**)
proof (*auto simp add*: *expand-fun-eq*)
show **1** = (*differ*) **1**
proof (*rule sym*, *intro group-hom.hom-one*)

from *prems* **show** *group-hom D D* (*differ*)
by (*unfold group-hom-def group-hom-axioms-def diff-group-def comm-group-def diff-group-axioms-def hom-diff-def*
hom-completion-def, *simp*)
qed
qed
qed
next
fix *x y z*
assume $x \in$ *carrier ?DIFF* **and** $y \in$ *carrier ?DIFF* **and** $z \in$ *carrier ?DIFF*
from *prems* **show** $x \otimes_{?DIFF} y \otimes_{?DIFF} z = x \otimes_{?DIFF} (y \otimes_{?DIFF} z)$
by (*auto simp add*: *expand-fun-eq*) (*intro m-assoc*, *unfold hom-diff-def hom-completion-def*, *auto simp add*: *hom-closed*)
next
fix *x y*
assume $x \in$ *carrier ?DIFF* **and** $y \in$ *carrier ?DIFF*
from *prems* **show** $x \otimes_{?DIFF} y = y \otimes_{?DIFF} x$
by (*auto simp add*: *expand-fun-eq*) (*intro m-comm*, *unfold hom-diff-def hom-completion-def hom-def Pi-def*, *auto*)
next
fix *x*
assume $x \in$ *carrier ?DIFF*
from *prems* **show** $\mathbf{1}_{?DIFF} \otimes_{?DIFF} x = x$
by (*auto simp add*: *expand-fun-eq*)
(*intro l-one*, *unfold hom-diff-def hom-completion-def completion-fun2-def completion-def hom-def Pi-def*, *auto*)
next
fix *x*
assume $x \in$ *carrier ?DIFF*

from *prems* **and** *hom-diff-inv* [*of x*] **show** $\exists y \in$ *carrier ?DIFF*. $y \otimes_{?DIFF} x = \mathbf{1}_{?DIFF}$ **by** *simp*
qed

The completion homomorphisms between differential groups are a ring with suitable operations

lemma (**in** *diff-group*) *hom-diff-ring*:
shows *ring* (| *carrier* = *hom-diff D D*, *mult* = *op o*, *one* = (λx. *if* $x \in$ *carrier D then id x else* **1**),
zero = (λx. *if* $x \in$ *carrier D then* **1** *else* **1**), *add* = λf. λg. (λx. *if* $x \in$ *carrier D then f x* $\otimes$ *g x else* **1**)|)
(**is** *ring ?DIFF*)
proof (*rule ringI*, *unfold abelian-group-def abelian-group-axioms-def abelian-monoid-def*, *auto*)
show *monoid* (| *carrier* = *hom-diff D D*, *mult* = *op o*, *one* = λx. *if* $x \in$ *carrier D then id x else* **1**, *zero* = λx. **1**,
add = $\lambda f\ g\ x$. *if* $x \in$ *carrier D then f x* $\otimes$ *g x else* **1** |)
proof −
from *diff-group.hom-diff-monoid* **and** *prems* **have** *monoid* (|*carrier* = *hom-diff*

D D, *mult* = *op* ∘,
      one = λ*x*. *if x* ∈ *carrier D then id x else* **1**⦈
      by (*unfold diff-group-def*, *auto*)
    then show *?thesis* **by** (*unfold monoid-def*, *simp*)
  qed
next
  show *comm-monoid* (⦇*carrier* = *hom-diff D D*, *mult* = λ*f g x*. *if x* ∈ *carrier D then f x* ⊗ *g x else* **1**, *one* = λ*x*. **1**⦈)
  proof −
    from *diff-group.hom-diff-mult-comm-group* **and** *prems*
    have *comm-group* (⦇*carrier* = *hom-diff D D*, *mult* = λ*f g x*. *if x* ∈ *carrier D then f x* ⊗ *g x else* **1**,
      one = λ*x*. *if x* ∈ *carrier D then* **1** *else* **1**⦈)
      by (*unfold diff-group-def*, *auto*)
    then show *?thesis* **by** (*unfold comm-group-def comm-monoid-def*, *simp*)
  qed
next
  show *comm-group* (⦇*carrier* = *hom-diff D D*, *mult* = λ*f g x*. *if x* ∈ *carrier D then f x* ⊗ *g x else* **1**, *one* = λ*x*. **1**⦈)
  proof −
    from *diff-group.hom-diff-mult-comm-group* **and** *prems*
    have *comm-group* (⦇*carrier* = *hom-diff D D*, *mult* = λ*f g x*. *if x* ∈ *carrier D then f x* ⊗ *g x else* **1**,
      one = λ*x*. *if x* ∈ *carrier D then* **1** *else* **1**⦈)
      by (*unfold diff-group-def*, *auto*)
    then show *?thesis* **by** *simp*
  qed
next
  show ⋀*x y z*. ⟦*x* ∈ *hom-diff D D*; *y* ∈ *hom-diff D D*; *z* ∈ *hom-diff D D*⟧
    ⟹ (λ*xa*. *if xa* ∈ *carrier D then x xa* ⊗ *y xa else* **1**) ∘ *z* = (λ*xa*. *if xa* ∈ *carrier D then* (*x* ∘ *z*) *xa* ⊗ (*y* ∘ *z*) *xa else* **1**)
  proof −
    fix *x y z*
    assume *x* ∈ *hom-diff D D* **and** *y* ∈ *hom-diff D D* **and** *z* ∈ *hom-diff D D*
    from *prems*
    show (λ*xa*. *if xa* ∈ *carrier D then x xa* ⊗ *y xa else* **1**) ∘ *z* = (λ*xa*. *if xa* ∈ *carrier D then* (*x* ∘ *z*) *xa* ⊗ (*y* ∘ *z*) *xa else* **1**)
      by (*intro r-mult-dist-add*) (*unfold hom-diff-def*, *simp-all*)
  qed
next
  show ⋀*x y z*. ⟦ *x* ∈ *hom-diff D D*; *y* ∈ *hom-diff D D*; *z* ∈ *hom-diff D D*⟧
    ⟹ *z o* (λ*xa*. *if xa* ∈ *carrier D then x xa* ⊗ *y xa else* **1**) = (λ*xa*. *if xa* ∈ *carrier D then* (*z o x*) *xa* ⊗ (*z o y*) *xa else* **1**)
    by (*intro l-mult-dist-add*) (*unfold hom-diff-def*, *simp-all*)
qed (*simp-all add*: *hom-diff-def*)

end

theory *HomGroupsCompletion*
imports
HomGroupCompletion
begin

2.10 Homomorphisms seen as algebraic structures

Homomorphisms with the underlying operation are closed

lemma *hom-mult-completion-is-hom*:
includes *comm-group G + comm-group G′*
shows [|*f* : *hom G G′*; *g* : *hom G G′* |] ==> (%*x*. *if x* ∈ *carrier G then f x* $\otimes_2$ *g x else* $\mathbf{1}_2$) : *hom G G′*
apply (*unfold hom-def*, *simp add*: *prems*, *auto simp add*: *Pi-def m-closed*)
apply (*simp add*: *m-ac hom-completion-closed*)
done

lemma *hom-completion-mult-is-hom-completion*:
includes *comm-group G + comm-group G′*
assumes *f* ∈ *hom-completion G G′* **and** *g* ∈ *hom-completion G G′*
shows (λ*x*. *if x* ∈ *carrier G then f x* $\otimes_{G'}$ *g x else* $\mathbf{1}_{G'}$) ∈ *hom-completion G G′*
proof (*unfold hom-completion-def completion-fun2-def completion-def*, *simp*, *intro conjI*)
show ∃ *ga*. (λ*x*. *if x* ∈ *carrier G then f x* $\otimes_{G'}$ *g x else* $\mathbf{1}_{G'}$) = (λ*x*. *if x* ∈ *carrier G then ga x else* $\mathbf{1}_{G'}$)
by (*rule exI* [*of* - (λ*x*. *f x* $\otimes_{G'}$ *g x*)])(*simp*)
next
from *prems* **show** (λ*x*. *if x* ∈ *carrier G then f x* $\otimes_{G'}$ *g x else* $\mathbf{1}_{G'}$) ∈ *hom G G′*
by (*intro hom-mult-completion-is-hom*, *unfold comm-group-def hom-completion-def*)(*simp-all*)
qed

Proof of the existence of an inverse homomorphism

lemma *hom-completions-mult-inv-is-hom-completion*:
includes *comm-group G + comm-group G′*
assumes *f* ∈ *hom-completion G G′*
shows ∃ *g*∈*hom-completion G G′*. (λ*x*. *if x* ∈ *carrier G then g x* $\otimes_{G'}$ *f x else* $\mathbf{1}_{G'}$) = (λ*x*. $\mathbf{1}_{G'}$)
proof (*intro bexI* [*of* - (λ*x*. *if x* ∈ *carrier G then* $inv_{G'}$ (*f x*) *else* $\mathbf{1}_{G'}$)])
show (λ*g*. *if g* ∈ *carrier G then* $inv_{G'}$ *f g else* $\mathbf{1}_{G'}$) ∈ *hom-completion G G′*
proof (*unfold hom-completion-def completion-fun2-def completion-def*, *simp*, *intro conjI*)
show ∃ *g*. (λ*g*. *if g* ∈ *carrier G then* $inv_{G'}$ *f g else* $\mathbf{1}_{G'}$) = (λ*x*. *if x* ∈ *carrier G then g x else* $\mathbf{1}_{G'}$)
by (*rule exI* [*of* - λ*x*. $inv_{G'}$ (*f x*)], *simp*)
next
show (λ*x*. *if x* ∈ *carrier G then* $inv_{G'}$ *f x else* $\mathbf{1}_{G'}$) ∈ *hom G G′*
proof (*unfold hom-def*, *simp*, *intro conjI*)
from *prems* **show** (λ*x*. *if x* ∈ *carrier G then* $inv_{G'}$ *f x else* $\mathbf{1}_{G'}$) ∈ *carrier G* → *carrier G′*

by (*unfold Pi-def hom-completion-def completion-fun2-def completion-def hom-def Pi-def*, *auto*)
next
show $\forall$ *x*∈*carrier G*. $\forall$ *y*∈*carrier G*. $inv_{G'}$ *f* $(x \otimes y)$ = $inv_{G'}$ *f x* $\otimes_{G'}$ $inv_{G'}$ *f y*
proof (*intro ballI*)
fix *x y*
assume $x \in$ *carrier G* **and** $y \in$ *carrier G*
show $inv_{G'}$ *f* $(x \otimes y)$ = $inv_{G'}$ *f x* $\otimes_{G'}$ $inv_{G'}$ *f y*
proof −
from *prems* **have** *f* $(x \otimes y)$ = *f x* $\otimes_{G'}$ *f y* **by** (*intro hom-mult*, *unfold hom-completion-def*, *simp-all*)
then have $inv_{G'}$ *f* $(x \otimes y)$ = $inv_{G'}$ (*f x* $\otimes_{G'}$ *f y*) **by** *simp*
also from *prems* **have** ... = $inv_{G'}$ (*f y*) $\otimes_{G'}$ $inv_{G'}$ (*f x*)
by (*intro inv-mult-group*) (*unfold hom-completion-def hom-def Pi-def*, *simp-all*)
also from *prems* **have** ... = $inv_{G'}$ (*f x*) $\otimes_{G'}$ $inv_{G'}$ (*f y*)
by (*intro m-comm*)(*unfold hom-completion-def hom-def Pi-def*, *simp-all*)
finally show *?thesis* **by** *simp*
qed
qed
qed
qed
next
from *prems*
show (λx. *if* $x \in$ *carrier G then* (*if* $x \in$ *carrier G then* $inv_{G'}$ *f x else* $\mathbf{1}_{G'}$) $\otimes_{G'}$ *f x else* $\mathbf{1}_{G'}$) = (λx. $\mathbf{1}_{G'}$)
by (*unfold hom-completion-def hom-def Pi-def*, *auto simp add*: *expand-fun-eq*)
qed

2.11 Completion homomorphisms between two algebraic structures form a commutative group

lemma *hom-completion-groups-mult-comm-group*:
includes *comm-group G* + *comm-group G′*
shows *comm-group* (| *carrier* = *hom-completion G G′*, *mult* = λf. λg. (λx. *if x* ∈ *carrier G then f x* $\otimes_2$ *g x else* $\mathbf{1}_2$),
one = (λx. *if* $x \in$ *carrier G then* $\mathbf{1}_2$ *else* $\mathbf{1}_2$)|)
(**is** *comm-group ?H-CO*)
proof (*intro comm-groupI*)
fix *f g*
assume $f \in$ *carrier ?H-CO* **and** $g \in$ *carrier ?H-CO*
from *prems* **show** *f* $\otimes_{?H\text{-}CO}$ *g* ∈ *carrier ?H-CO*
by *simp* (*intro hom-completion-mult-is-hom-completion*, *unfold comm-group-def*, *simp-all*)
next
show $\mathbf{1}_{?H\text{-}CO}$ ∈ *carrier ?H-CO*
proof (*unfold hom-completion-def completion-fun2-def completion-def*, *auto*)
show $\exists$ *g*. (λx. $\mathbf{1}_{G'}$) = (λx. *if* $x \in$ *carrier G then g x else* $\mathbf{1}_{G'}$) **by** (*intro exI*

[*of* - λx. $\mathbf{1}_{G'}$], *simp*)
  next
    show (λx. $\mathbf{1}_{G'}$) $\in$ *hom G G'* **by** (*unfold hom-def Pi-def*, *simp*)
  qed
next
  fix x y z
  assume $x \in$ *carrier ?H-CO* **and** $y \in$ *carrier ?H-CO* **and** $z \in$ *carrier ?H-CO*
  from *prems* **show** $x \otimes_{?H\text{-}CO} y \otimes_{?H\text{-}CO} z = x \otimes_{?H\text{-}CO} (y \otimes_{?H\text{-}CO} z)$
    by (*auto simp add*: *expand-fun-eq hom-completion-def hom-def Pi-def*) (*rule m-assoc*, *simp-all*)
next
  fix x y
  assume $x \in$ *carrier ?H-CO* **and** $y \in$ *carrier ?H-CO*
  from *prems* **show** $x \otimes_{?H\text{-}CO} y = y \otimes_{?H\text{-}CO} x$
    by (*auto simp add*: *expand-fun-eq hom-completion-def hom-def Pi-def*) (*rule m-comm*, *simp-all*)
next
  fix x
  assume $x \in$ *carrier ?H-CO*
  from *prems* **show** $\mathbf{1}_{?H\text{-}CO} \otimes_{?H\text{-}CO} x = x$
    by (*auto simp add*: *expand-fun-eq hom-completion-def completion-fun2-def completion-def hom-def Pi-def*)
next
  fix x
  assume $x \in$ *carrier ?H-CO*
  from *prems* **and** *hom-completions-mult-inv-is-hom-completion* [*of G G' x*] **show** $\exists\, y \in$ *carrier ?H-CO*. $y \otimes_{?H\text{-}CO} x = \mathbf{1}_{?H\text{-}CO}$
    by (*unfold comm-group-def*, *simp*)
qed

2.12 Previous facts about homomorphisms of differential structures

lemma *hom-diff-mult-is-hom-diff*:
  includes *diff-group D* + *diff-group D'*
  assumes $f \in$ *hom-diff D D'* **and** $g \in$ *hom-diff D D'*
  shows (λx. *if* $x \in$ *carrier D then f x* $\otimes_{D'}$ *g x else* $\mathbf{1}_{D'}$) $\in$ *hom-diff D D'*
proof (*unfold hom-diff-def hom-completion-def completion-fun2-def completion-def*, *simp*, *intro conjI*)
  show $\exists$ *ga*. (λx. *if* $x \in$ *carrier D then f x* $\otimes_{D'}$ *g x else* $\mathbf{1}_{D'}$) = (λx. *if* $x \in$ *carrier D then ga x else* $\mathbf{1}_{D'}$)
    by (*rule exI* [*of* - (λx. *f x* $\otimes_{D'}$ *g x*)])(*simp*)
next
  from *prems* **show** (λx. *if* $x \in$ *carrier D then f x* $\otimes_{D'}$ *g x else* $\mathbf{1}_{D'}$) $\in$ *hom D D'*
    by (*unfold hom-diff-def hom-completion-def hom-def Pi-def*, *auto simp add*: *m-ac*)
next
  show (λx. *if* $x \in$ *carrier D then f x* $\otimes_{D'}$ *g x else* $\mathbf{1}_{D'}$) $\circ$ *differ* = *differ*$_{D'}$ $\circ$ (λx. *if* $x \in$ *carrier D then f x* $\otimes_{D'}$ *g x else* $\mathbf{1}_{D'}$)

proof (*rule ext*)
fix *x*
show ((λx. *if* $x \in$ *carrier D then f x* $\otimes_{D'}$ *g x else* $\mathbf{1}_{D'}$) $\circ$ *differ*) *x* = (*differ*$_{D'}$ $\circ$ (λx. *if* $x \in$ *carrier D then f x* $\otimes_{D'}$ *g x else* $\mathbf{1}_{D'}$)) *x*
proof (*cases x* $\in$ *carrier D*)
case *True*
from *prems* **show** *?thesis*
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def*

completion-fun2-def completion-def hom-def Pi-def) (*auto simp add*: *expand-fun-eq*)
next
case *False*
then show *?thesis*
proof (*auto simp add*: *expand-fun-eq*)
show *f* ((*differ*) *x*) $\otimes_{D'}$ *g* ((*differ*) *x*) = (*differ*$_{D'}$) $\mathbf{1}_{D'}$
proof −
from *prems* **have** *l-h-s-1*: *f* ((*differ*) *x*) = $\mathbf{1}_{D'}$
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def completion-fun2-def completion-def*,
auto) (*intro group-hom.hom-one*, *unfold comm-group-def group-hom-def group-hom-axioms-def hom-def Pi-def*, *simp*)
moreover from *prems* **have** *l-h-s-2*: *g* ((*differ*) *x*) = $\mathbf{1}_{D'}$
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def completion-fun2-def completion-def*,
auto) (*intro group-hom.hom-one*, *unfold comm-group-def group-hom-def group-hom-axioms-def hom-def Pi-def*, *simp*)
moreover from *prems* **have** *r-h-s*: (*differ*$_{D'}$) $\mathbf{1}_{D'}$ = $\mathbf{1}_{D'}$
by (*intro group-hom.hom-one*)(*unfold diff-group-def comm-group-def group-hom-def group-hom-axioms-def diff-group-axioms-def*
hom-diff-def hom-completion-def, *simp*)
ultimately show *?thesis* **by** *simp*
qed
next
show $\mathbf{1}_{D'}$ = (*differ*$_{D'}$) $\mathbf{1}_{D'}$
proof (*rule sym*, *intro group-hom.hom-one*)
from *prems* **show** *group-hom D' D'* (*differ*$_{D'}$)
by (*unfold diff-group-def comm-group-def group-hom-def group-hom-axioms-def diff-group-axioms-def hom-diff-def*
hom-completion-def, *simp*)
qed
qed
qed
qed
qed

lemma *hom-diff-mult-inv-is-hom-diff*:
includes *diff-group D* + *diff-group D'*
assumes *f* $\in$ *hom-diff D D'*

shows $\exists\, g{\in}$*hom-diff D D′*. ($\lambda x.$ *if* $x \in$ *carrier D then g x* $\otimes_{D'}$ *f x else* $\mathbf{1}_{D'}$) = ($\lambda x.$ $\mathbf{1}_{D'}$)
proof (*intro bexI* [*of* - ($\lambda x.$ *if* $x \in$ *carrier D then* $inv_{D'}$ (*f x*) *else* $\mathbf{1}_{D'}$)])
show ($\lambda g.$ *if* $g \in$ *carrier D then* $inv_{D'}$ *f g else* $\mathbf{1}_{D'}$) $\in$ *hom-diff D D′*
proof (*unfold hom-diff-def hom-completion-def completion-fun2-def completion-def*, *simp*, *intro conjI*)
show $\exists\, g.$ ($\lambda g.$ *if* $g \in$ *carrier D then* $inv_{D'}$ *f g else* $\mathbf{1}_{D'}$) = ($\lambda x.$ *if* $x \in$ *carrier D then g x else* $\mathbf{1}_{D'}$)
by (*rule exI* [*of* - $\lambda x.$ $inv_{D'}$ (*f x*)], *simp*)
next
show ($\lambda x.$ *if* $x \in$ *carrier D then* $inv_{D'}$ *f x else* $\mathbf{1}_{D'}$) $\in$ *hom D D′*
proof (*unfold hom-def*, *simp*, *intro conjI*)
from *prems* **show** ($\lambda x.$ *if* $x \in$ *carrier D then* $inv_{D'}$ *f x else* $\mathbf{1}_{D'}$) $\in$ *carrier D* $\rightarrow$ *carrier D′*
by (*unfold hom-diff-def hom-completion-def completion-fun2-def completion-def hom-def Pi-def*, *auto*)
next
show $\forall\, x{\in}$*carrier D*. $\forall\, y{\in}$*carrier D*. $inv_{D'}$ *f* ($x \otimes y$) = $inv_{D'}$ *f x* $\otimes_{D'}$ $inv_{D'}$ *f y*
proof (*intro ballI*)
fix *x y*
assume $x \in$ *carrier D* **and** $y \in$ *carrier D*
show $inv_{D'}$ *f* ($x \otimes y$) = $inv_{D'}$ *f x* $\otimes_{D'}$ $inv_{D'}$ *f y*
proof −
from *prems* **have** *f* ($x \otimes y$) = *f x* $\otimes_{D'}$ *f y* **by** (*intro hom-mult*, *unfold hom-diff-def hom-completion-def*, *simp-all*)
then have $inv_{D'}$ *f* ($x \otimes y$) = $inv_{D'}$ (*f x* $\otimes_{D'}$ *f y*) **by** *simp*
also from *prems* **have** ... = $inv_{D'}$ (*f y*) $\otimes_{D'}$ $inv_{D'}$ (*f x*)
by (*intro inv-mult-group*) (*unfold hom-diff-def hom-completion-def hom-def Pi-def*, *simp-all*)
also from *prems* **have** ... = $inv_{D'}$ (*f x*) $\otimes_{D'}$ $inv_{D'}$ (*f y*)
by (*intro m-comm*)(*unfold hom-diff-def hom-completion-def hom-def Pi-def*, *simp-all*)
finally show *?thesis* **by** *simp*
qed
qed
qed
next
show ($\lambda g.$ *if* $g \in$ *carrier D then* $inv_{D'}$ *f g else* $\mathbf{1}_{D'}$) $\circ$ *differ* = $differ_{D'}$ $\circ$ ($\lambda g.$ *if* $g \in$ *carrier D then* $inv_{D'}$ *f g else* $\mathbf{1}_{D'}$)
proof (*simp add*: *expand-fun-eq*, *intro allI impI conjI*)
fix *x*
assume $x \in$ *carrier D*
show $inv_{D'}$ *f* ((*differ*) *x*) = ($differ_{D'}$) ($inv_{D'}$ *f x*)
proof −

have ($inv_{D'}$ *f* ((*differ*) *x*) = ($differ_{D'}$) ($inv_{D'}$ *f x*)) = ($inv_{D'}$ *f* ((*differ*) *x*) $\otimes_{D'}$ *f* ((*differ*) *x*) = ($differ_{D'}$) ($inv_{D'}$ *f x*) $\otimes_{D'}$ *f* ((*differ*) *x*))

proof (*rule sym*, *rule r-cancel*)
from *prems* **show** $f\ ((\mathit{differ})\ x) \in \mathit{carrier}\ D'$
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
next
from *prems* **show** $\mathit{inv}_{D'}\ f\ ((\mathit{differ})\ x) \in \mathit{carrier}\ D'$
by (*intro inv-closed*)(*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
next
from *prems* **show** $(\mathit{differ}_{D'})\ (\mathit{inv}_{D'}\ f\ x) \in \mathit{carrier}\ D'$
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
qed
also have $(\mathit{inv}_{D'}\ f\ ((\mathit{differ})\ x) \otimes_{D'} f\ ((\mathit{differ})\ x) = (\mathit{differ}_{D'})\ (\mathit{inv}_{D'}\ f\ x) \otimes_{D'} f\ ((\mathit{differ})\ x))$
proof −
have *l-h*: $\mathit{inv}_{D'}\ f\ ((\mathit{differ})\ x) \otimes_{D'}\ f\ ((\mathit{differ})\ x) = \mathbf{1}_{D'}$
proof (*rule l-inv*)
from *prems* **show** $f\ ((\mathit{differ})\ x) \in \mathit{carrier}\ D'$
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
qed
moreover have *r-h*: $(\mathit{differ}_{D'})\ (\mathit{inv}_{D'}\ f\ x) \otimes_{D'} f\ ((\mathit{differ})\ x) = \mathbf{1}_{D'}$
proof −
have $(\mathit{differ}_{D'})\ (\mathit{inv}_{D'}\ f\ x) \otimes_{D'} f\ ((\mathit{differ})\ x) = (\mathit{differ}_{D'})\ (\mathit{inv}_{D'}\ f\ x) \otimes_{D'} (\mathit{differ}_{D'})\ (f\ x)$
proof −
from *prems* **have** $f\ ((\mathit{differ})\ x) = (\mathit{differ}_{D'})\ (f\ x)$
by (*unfold diff-group-axioms-def hom-diff-def hom-completion-def hom-def Pi-def*, *simp add*: *expand-fun-eq*)
then show *?thesis* **by** *simp*
qed
also have $\ldots = (\mathit{differ}_{D'})\ (\mathit{inv}_{D'}\ f\ x \otimes_{D'} f\ x)$
proof (*rule sym*, *rule hom-mult*)
from *prems* **show** $\mathit{differ}_{D'} \in \mathit{hom}\ D'\ D'$
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def*, *simp*)
next
from *prems* **show** $\mathit{inv}_{D'}\ f\ x \in \mathit{carrier}\ D'$
by (*intro inv-closed*, *unfold hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
next
from *prems* **show** $f\ x \in \mathit{carrier}\ D'$
by (*unfold hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
qed
also have $\ldots = (\mathit{differ}_{D'})\ (\mathbf{1}_{D'})$
proof −
from *prems* **have** $\mathit{inv}_{D'}\ f\ x \otimes_{D'} f\ x = \mathbf{1}_{D'}$
by (*intro l-inv*, *unfold hom-diff-def hom-completion-def hom-def Pi-def*,

simp)
then show *?thesis* **by** *simp*
qed
also from *prems* **have** $\ldots = \mathbf{1}_{D'}$
proof (*intro group-hom.hom-one*, *unfold diff-group-def comm-group-def group-hom-def group-hom-axioms-def*, *intro conjI*, *simp-all*)
from *prems* **show** $differ_{D'} \in hom\ D'\ D'$
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def*, *simp*)
qed
finally show *?thesis* **by** *simp*
qed
ultimately show *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
next

fix x
assume $x \in carrier\ D$ **and** $(differ)\ x \notin carrier\ D$

from *prems* **show** $\mathbf{1}_{D'} = (differ_{D'})\ (inv_{D'}\ f\ x)$
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def hom-def Pi-def*, *simp*)
next
fix x
assume $x \notin carrier\ D$
from *prems* **show** $inv_{D'}\ f\ ((differ)\ x) = (differ_{D'})\ \mathbf{1}_{D'}$
proof −
have *l-h*: $inv_{D'}\ f\ ((differ)\ x) = \mathbf{1}_{D'}$
proof −
from *prems* **have** $inv_{D'}\ f\ ((differ)\ x) = inv_{D'}\ f\ (\mathbf{1})$
by (*unfold diff-group-def diff-group-axioms-def hom-diff-def hom-completion-def completion-fun2-def completion-def Pi-def*, *auto*)
also have $\ldots = inv_{D'}\ \mathbf{1}_{D'}$
proof −
from *prems* **have** $f\ \mathbf{1} = \mathbf{1}_{D'}$
by (*intro group-hom.hom-one*) (*unfold diff-group-def comm-group-def group-hom-def group-hom-axioms-def hom-diff-def hom-completion-def*, *auto*)
then show *?thesis* **by** *simp*
qed
also have $\ldots = \mathbf{1}_{D'}$
by (*rule inv-one*)
finally show *?thesis* **by** *simp*
qed
moreover from *prems* **have** *r-h*: $(differ_{D'})\ \mathbf{1}_{D'} = \mathbf{1}_{D'}$
by (*intro group-hom.hom-one*)
(*unfold diff-group-def diff-group-axioms-def comm-group-def group-hom-def group-hom-axioms-def hom-diff-def hom-completion-def*, *auto*)

ultimately show *?thesis* **by** *simp*
qed
next
show $\mathbf{1}_{D'} = (\mathit{differ}_{D'})\ \mathbf{1}_{D'}$
proof (*rule sym*, *intro group-hom.hom-one*)
from *prems* **show** *group-hom* D' D' $(\mathit{differ}_{D'})$
by (*unfold diff-group-def comm-group-def diff-group-axioms-def group-hom-def group-hom-axioms-def hom-diff-def hom-completion-def*, *auto*)
qed
qed
qed
next
from *prems* **show** $(\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ D\ \mathit{then}\ (\mathit{if}\ x \in \mathit{carrier}\ D\ \mathit{then}\ \mathit{inv}_{D'}\ f\ x\ \mathit{else}\ \mathbf{1}_{D'}) \otimes_{D'} f\ x\ \mathit{else}\ \mathbf{1}_{D'}) = (\lambda x.\ \mathbf{1}_{D'})$
by (*unfold hom-diff-def hom-completion-def hom-def Pi-def*, *auto simp add*: *expand-fun-eq*)
qed

The set of completion differential homomorphisms between two differential groups are a commutative group

lemma *hom-diff-groups-mult-comm-group*:
includes *diff-group D* + *diff-group* D'
shows *comm-group* (| *carrier* = *hom-diff D* D', *mult* = $\lambda f.\ \lambda g.\ (\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ D\ \mathit{then}\ f\ x \otimes_2 g\ x\ \mathit{else}\ \mathbf{1}_2)$,
one = $(\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ D\ \mathit{then}\ \mathbf{1}_2\ \mathit{else}\ \mathbf{1}_2)$|)
(**is** *comm-group ?H-DI*)
proof (*intro comm-groupI*)
fix *f g*
assume $f \in$ *carrier ?H-DI* **and** $g \in$ *carrier ?H-DI*
from *prems* **and** *hom-diff-mult-is-hom-diff* [*of D* D' *f g*] **show** $f \otimes_{?H\text{-}DI} g \in$ *carrier ?H-DI*
by (*unfold diff-group-def*, *simp*)
next
show $\mathbf{1}_{?H\text{-}DI} \in$ *carrier ?H-DI*
proof (*unfold hom-diff-def hom-completion-def completion-fun2-def completion-def*, *auto*)
show $\exists g.\ (\lambda x.\ \mathbf{1}_{D'}) = (\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ D\ \mathit{then}\ g\ x\ \mathit{else}\ \mathbf{1}_{D'})$
by (*intro exI* [*of* - $\lambda x.\ \mathbf{1}_{D'}$], *simp*)
next
show $(\lambda x.\ \mathbf{1}_{D'}) \in \mathit{hom}\ D\ D'$
by (*unfold hom-def Pi-def*, *simp*)
next
from *prems* **show** $(\lambda x.\ \mathbf{1}_{D'}) \circ \mathit{differ} = \mathit{differ}_{D'} \circ (\lambda x.\ \mathbf{1}_{D'})$
by (*auto simp add*: *expand-fun-eq*) (*rule sym*, *intro group-hom.hom-one*,
unfold diff-group-def comm-group-def group-hom-def group-hom-axioms-def diff-group-axioms-def hom-diff-def hom-completion-def, *simp*)
qed
next
fix *x y z*

assume $x \in$ *carrier ?H-DI* **and** $y \in$ *carrier ?H-DI* **and** $z \in$ *carrier ?H-DI*
from *prems* **show** $x \otimes_{?H\text{-}DI} y \otimes_{?H\text{-}DI} z = x \otimes_{?H\text{-}DI} (y \otimes_{?H\text{-}DI} z)$
by (*auto simp add*: *expand-fun-eq hom-diff-def hom-completion-def hom-def Pi-def*) (*rule m-assoc*, *simp-all*)
next
fix *x y*
assume $x \in$ *carrier ?H-DI* **and** $y \in$ *carrier ?H-DI*
from *prems* **show** $x \otimes_{?H\text{-}DI} y = y \otimes_{?H\text{-}DI} x$
by (*auto simp add*: *expand-fun-eq hom-diff-def hom-completion-def hom-def Pi-def*) (*rule m-comm*, *simp-all*)
next
fix *x*
assume $x \in$ *carrier ?H-DI*
from *prems* **show** $\mathbf{1}_{?H\text{-}DI} \otimes_{?H\text{-}DI} x = x$
by (*auto simp add*: *expand-fun-eq hom-diff-def hom-completion-def completion-fun2-def completion-def hom-def Pi-def*)
next
fix *x*
assume $x \in$ *carrier ?H-DI*

from *prems* **and** *hom-diff-mult-inv-is-hom-diff* [*of D D′ x*] **show** $\exists\, y \in$*carrier ?H-DI*. $y \otimes_{?H\text{-}DI} x = \mathbf{1}_{?H\text{-}DI}$
by (*unfold diff-group-def*, *simp*)
qed

The following result has been already proved in *comm-group* (|*carrier* = *hom-diff D D*, *mult* = $\lambda f\ g\ x.$ *if* $x \in$ *carrier D then* $f\ x \otimes g\ x$ *else* **1**, *one* = $\lambda x.$ *if* $x \in$ *carrier D then* **1** *else* **1**|); now that we have provided a proof of a similar result but for two different differential groups, D and D', it can be trivially proved for the case $D = D'$

lemma (**in** *diff-group*) *hom-diff-group-mult-comm-group-inst*:
shows *comm-group* (| *carrier* = *hom-diff D D*, *mult* = $\lambda f.\ \lambda g.\ (\lambda x.$ *if* $x \in$ *carrier D then* $f\ x \otimes g\ x$ *else* **1**),
one = ($\lambda x.$ *if* $x \in$ *carrier D then* **1** *else* **1**)|)
using *prems hom-diff-groups-mult-comm-group* [*of D D*] **by** *simp*

end

3 Previous definitions and Propositions 2.2.9, 2.2.10 and Lemma 2.2.11 in Aransay's memoir

theory *lemma-2-2-11*
imports
~~/src/HOL/Algebra/Coset
HomGroupsCompletion
begin

Definitions and results leading to prove that the *ker* and *image* sets of a given homomorphism are subgroups and give place to suitable algebraic structures

locale *comm-group-hom* = *group-hom* +
assumes *comm-group-G*: *comm-group G*
and *comm-group-H*: *comm-group H*
and *hom-completion-h*: *h* ∈ *completion-fun2 G H*

lemma *comm-group-hom* [*intro*]: **assumes** *G*: *comm-group G* **and** *H*: *comm-group H* **and** *h*: *h* ∈ *hom-completion G H*
shows *comm-group-hom G H h*
using *G H h*
by (*unfold comm-group-hom-def comm-group-hom-axioms-def group-hom-def group-hom-axioms-def hom-completion-def comm-group-def*, *simp*)

lemma (**in** *comm-group-hom*) *subgroup-kernel*: *subgroup* (*kernel G H h*) *G*
by (*rule subgroup.intro*) (*auto simp add*: *kernel-def*)

lemma (**in** *comm-group-hom*) *kernel-comm-group*: *comm-group* (| *carrier* = (*kernel G H h*), *mult* = *mult G*, *one* = *one G*|)
using *prems*
apply (*intro comm-groupI*)
apply (*unfold comm-group-hom-def comm-group-hom-axioms-def comm-group-def comm-monoid-axioms-def kernel-def*, *auto simp add*: *G.m-assoc*)
apply (*unfold comm-monoid-def comm-monoid-axioms-def*, *simp*)
done

locale *diff-group-hom-diff* = *comm-group-hom D C h* +
assumes *diff-group-axioms-D*: *diff-group-axioms D*
and *diff-group-axioms-C*: *diff-group-axioms C*
and *diff-hom-h*: $h \circ differ_D = differ_C \circ h$

lemma *diff-group-hom-diffI*: **assumes** *d-g-D*: *diff-group D* **and** *d-g-C*: *diff-group C* **and** *h-hom*: *h* ∈ *hom-diff D C*
shows *diff-group-hom-diff D C h*
using *d-g-D* **and** *d-g-C* **and** *h-hom*
by (*unfold diff-group-hom-diff-def diff-group-hom-diff-axioms-def comm-group-hom-def comm-group-hom-axioms-def diff-group-def hom-diff-def hom-completion-def group-hom-def group-hom-axioms-def comm-group-def*, *simp*)

lemma (**in** *diff-group-hom-diff*) *diff-group-D*: **shows** *diff-group D*
using *prems* **by** (*unfold diff-group-def diff-group-hom-diff-def diff-group-hom-diff-axioms-def comm-group-hom-def comm-group-hom-axioms-def group-hom-def group-hom-axioms-def comm-group-def*) (*simp*)

lemma (**in** *diff-group-hom-diff*) *diff-group-C*: **shows** *diff-group C*
using *prems* **by** (*unfold diff-group-def diff-group-hom-diff-def diff-group-hom-diff-axioms-def*

comm-group-hom-def comm-group-hom-axioms-def group-hom-def group-hom-axioms-def comm-group-def) (*simp*)

lemma (**in** *diff-group-hom-diff*) *hom-diff-h*: **shows** $h \in$ *hom-diff D C*
using *prems* **by** (*unfold diff-group-def diff-group-hom-diff-def diff-group-hom-diff-axioms-def comm-group-hom-def comm-group-hom-axioms-def group-hom-def group-hom-axioms-def hom-diff-def hom-completion-def*) *simp*

lemma (**in** *diff-group-hom-diff*) *group-hom-D-D-differ*: **shows** *group-hom D D* ($differ_D$)
using *prems* **by** (*unfold group-hom-def group-hom-axioms-def diff-group-hom-diff-def diff-group-hom-diff-axioms-def comm-group-hom-def diff-group-axioms-def* [*of D*] *hom-completion-def*) *simp*

lemma (**in** *diff-group-hom-diff*) *group-hom-C-C-differ*: **shows** *group-hom C C* ($differ_C$)
using *prems* **by** (*unfold group-hom-def group-hom-axioms-def diff-group-hom-diff-def diff-group-hom-diff-axioms-def comm-group-hom-def diff-group-axioms-def* [*of C*] *hom-completion-def*) *simp*

lemma (**in** *diff-group-hom-diff*) *subgroup-kernel*: *subgroup* (*kernel D C h*) *D*
by (*rule subgroup.intro*) (*auto simp add*: *kernel-def*)

The following lemma corresponds to Proposition 2.2.9 in Aransay's thesis

Due to the use of completion functions for the differential, we need to define the *diff* function, which originally was a completion from D into D, as a completion from the kernel into the original differential group D

lemma (**in** *diff-group-hom-diff*) *kernel-diff-group*:
diff-group (| *carrier* = (*kernel D C h*), *mult* = *mult D*, *one* = *one D*,
diff = *completion* (|*carrier* = (*kernel D C h*), *mult* = *mult D*, *one* = *one D*, *diff* = *diff D* |) *D* (*diff D*)|)
(**is** *diff-group ?KER*)
proof (*intro diff-groupI*, *simp-all*)
fix *x y*
assume *x-in-ker*: $x \in$ *kernel D C h* **and** *y-in-ker*: $y \in$ *kernel D C h*
from *group-hom.subgroup-kernel* [*of D C h*] **and** *subgroup-def* [*of kernel D C h D*] **and** *prems* **show** $x \otimes y \in$ *kernel D C h*
by (*unfold diff-group-hom-diff-def comm-group-hom-def*, *simp*)
next
from *group-hom.subgroup-kernel* [*of D C h*] **and** *subgroup-def* [*of kernel D C h D*] **and** *prems*
show $\mathbf{1} \in$ *kernel D C h* **by** (*unfold diff-group-hom-diff-def comm-group-hom-def*, *simp*)
next
fix *x y z*
assume *x-in-ker*: $x \in$ *kernel D C h* **and** *y-in-ker*: $y \in$ *kernel D C h* **and** *z-in-ker*:

z ∈ *kernel D C h*
from *group-hom.subgroup-kernel* [*of D C h*] **and** *subgroup-def* [*of kernel D C h D*] **and** *prems* **and** *D.m-assoc* [*of x y z*]
show $x \otimes y \otimes z = x \otimes (y \otimes z)$ **by** (*unfold diff-group-hom-diff-def comm-group-hom-def*, *auto*)
next
fix *x y* **assume** *x-in-ker*: *x* ∈ *kernel D C h* **and** *y-in-ker*: *y* ∈ *kernel D C h*
from *group-hom.subgroup-kernel* [*of D C h*] **and** *prems*
and *psubsetD* [*of kernel D C h carrier D x*]
and *psubsetD* [*of kernel D C h carrier D y*]
and *comm-monoid.m-ac* (*2*) [*of D x y*]
show $x \otimes y = y \otimes x$ **unfolding** *diff-group-hom-diff-def* **unfolding** *comm-group-hom-def*

unfolding *subgroup-def*
unfolding *comm-group-hom-axioms-def* **unfolding** *comm-group-def* [*of D*] **by** *auto*
next
fix *x* **assume** *x-in-ker*: *x* ∈ *kernel D C h*
from *group-hom.subgroup-kernel* [*of D C h*] **and** *subgroup-def* [*of kernel D C h D*] **and** *prems* **show** $\mathbf{1} \otimes x = x$
unfolding *diff-group-hom-diff-def comm-group-hom-def* **by** *auto*
next
fix *x* **assume** *x-in-ker*: *x* ∈ *kernel D C h*
from *bexI* [*of* (λ*y*. $y \otimes x = \mathbf{1}$) *inv x kernel D C h*] **show** ∃ *y*∈*kernel D C h*. $y \otimes x = \mathbf{1}$
using *m-inv-def* [*of - x*]
using *prems* (*1*) **using** *group-hom.subgroup-kernel* [*of D C h*] **using** *x-in-ker*
unfolding *diff-group-hom-diff-def* **unfolding** *comm-group-hom-def* **unfolding** *subgroup-def* **by** *auto*
next
from *prems* **and** *diff-group-hom-diff.group-hom-C-C-differ* [*of D C h*] *diff-group-hom-diff.group-hom-D-D-differ* [*of D C h*]
and *group-hom.hom-one* [*of C C* $differ_C$]
show *completion* (|*carrier* = *kernel D C h*, *mult* = *op* ⊗, *one* = **1**, *diff* = *differ*|) *D* (*differ*)
∈ *hom-completion* (|*carrier* = *kernel D C h*, *mult* = *op* ⊗, *one* = **1**,
diff = *completion* (|*carrier* = *kernel D C h*, *mult* = *op* ⊗, *one* = **1**, *diff* = *differ*|) *D* (*differ*)|)
(|*carrier* = *kernel D C h*, *mult* = *op* ⊗, *one* = **1**,
diff = *completion* (|*carrier* = *kernel D C h*, *mult* = *op* ⊗, *one* = **1**, *diff* = *differ*|) *D* (*differ*)|)

unfolding *diff-group-hom-diff-def diff-group-hom-diff-axioms-def comm-group-hom-def comm-group-hom-axioms-def*
group-hom-def group-hom-axioms-def diff-group-axioms-def hom-completion-def hom-def completion-def completion-fun2-def
Pi-def kernel-def **by** (*auto simp add*: *expand-fun-eq*)
next
fix *x*

from *prems* **and** *diff-group.diff-nilpot* [*OF diff-group-hom-diff.diff-group-D* [*of D C h*]] **and** *group-hom.hom-one* [*of D D differ*$_D$]
and *diff-group-hom-diff.group-hom-D-D-differ* [*of D C h*]
show *completion* (|*carrier* = *kernel D C h*, *mult* = *op* ⊗, *one* = **1**, *diff* = *differ*|) *D* (*differ*)
(*completion* (|*carrier* = *kernel D C h*, *mult* = *op* ⊗, *one* = **1**, *diff* = *differ*|) *D* (*differ*) *x*) = **1**
unfolding *completion-def diff-group-hom-diff-def comm-group-hom-def* **by** (*auto simp add*: *expand-fun-eq*)
qed

The following lemma corresponds to Proposition 2.2.10 in Aransay's thesis; here it is proved for a generic homomorphism *h*

lemma (**in** *diff-group-hom-diff*) *image-diff-group*:
diff-group (| *carrier* = *image h* (*carrier D*), *mult* = *mult C*, *one* = *one C*,
diff = *completion* (| *carrier* = *image h* (*carrier D*), *mult* = *mult C*, *one* = *one C*, *diff* = *diff C*|) *C* (*diff C*)|)
(**is** *diff-group* (| *carrier* = *?img-set*, *mult* = *mult C*, *one* = *one C*, *diff* = *?compl* |) **is** *diff-group ?IMG*)
proof (*intro diff-groupI*, *auto*)
fix *x y*
assume *x*: *x* ∈ *carrier D* **and** *y*: *y* ∈ *carrier D*
from *hom-mult* [*OF x y*] **and** *D.m-closed* [*OF x y*] **show** *h x* ⊗$_C$ *h y* ∈ *?img-set* **unfolding** *image-def* **by** *force*
next
from *D.one-closed* **and** *group-hom.hom-one* [*of D C h*] **show** **1**$_C$ ∈ *?img-set* **unfolding** *image-def* **by** *force*
next
fix *x y z*
assume *x*: *x* ∈ *carrier D* **and** *y*: *y* ∈ *carrier D* **and** *z*: *z* ∈ *carrier D*
from *m-assoc* **and** *hom-closed* **and** *x y z* **show** *h x* ⊗$_C$ *h y* ⊗$_C$ *h z* = *h x* ⊗$_C$ (*h y* ⊗$_C$ *h z*) **by** *simp*
next
fix *x y*
assume *x*: *x* ∈ *carrier D* **and** *y*: *y* ∈ *carrier D*
from *prems* **and** *hom-closed* [*OF x*] **and** *hom-closed* [*OF y*] **and** *x y* **and** *comm-monoid.m-comm* [*of C h x h y*]
and *diff-group-hom-diff.diff-group-C* [*of D C h*]
show *h x* ⊗$_C$ *h y* = *h y* ⊗$_C$ *h x* **unfolding** *diff-group-def comm-group-def* [*of C*] **by** *auto*
next
fix *x*
assume *x*: *x* ∈ *carrier D*

from *Units-def* [*of D*] **and** *D.Units-eq* **and** *x* **obtain** *y* **where** *y*: *y* ∈ *carrier D* **and** *y-x*: *y* ⊗$_D$ *x* = **1**$_D$ **by** *fast*
from *group-hom.hom-one* [*of D C h*] **and** *hom-mult* [*OF y x*] **and** *y-x* **and** *y* **show** ∃ *y*∈*carrier D*. *h y* ⊗$_C$ *h x* = **1**$_C$ **by** *auto*
next

```
  show ?compl∈ hom-completion ?IMG ?IMG
  proof (intro hom-completionI homI, auto)
    show ?compl ∈ completion-fun2 ?IMG ?IMG unfolding completion-fun2-def
completion-def by (auto simp add: expand-fun-eq)
  next
    fix x
    assume x: x ∈ carrier D
    from prems and diff-group-hom-diff.diff-group-D [of D C h]
      and hom-completion-closed [OF diff-group.diff-hom [of D]  x]
       and imageI [of (differ_D) x carrier D h] and diff-group-hom-diff.diff-hom-h
[of D C h]
   show (differ_C) (h x) ∈ ?img-set unfolding image-def by (simp add: expand-fun-eq)
  next
    fix x y
    assume x: x ∈ carrier D and y: y ∈ carrier D
    from hom-mult [OF x y] and D.m-closed [OF x y]
    have ?compl (h x ⊗_C h y) = (differ_C) (h (x ⊗ y)) by (unfold completion-def
image-def, auto)
    also from diff-group-hom-diff.diff-group-C [of D C h] and hom-mult [OF x y]
       and hom-completion-mult [OF diff-group.diff-hom [of C] hom-closed [OF x]
hom-closed [OF y]] and prems
    have ... = (differ_C) (h x) ⊗_C (differ_C) (h y) by simp
     finally show ?compl (h x ⊗_C h y) = (differ_C) (h x) ⊗_C (differ_C) (h y) by
simp
  qed
next
  from diff-group-hom-diff.diff-group-C [of D C h] and diff-group.diff-nilpot [of
C]
   and diff-group.diff-hom [of C] and image-def and diff-group-hom-diff.group-hom-C-C-differ
[of D C h]
   and group-hom.hom-one [of C C differ_C] and prems show ⋀x. ?compl (?compl
x) = 1_C
     unfolding hom-completion-def completion-fun2-def completion-def by (auto
simp add: expand-fun-eq)
qed
```

Before proving Lemma 2.2.11, we first must introduce the definition of *reduction*

```
locale reduction = diff-group D + diff-group C + var f + var g + var h +
  assumes f-hom-diff: f ∈ hom-diff D C
  and g-hom-diff: g ∈ hom-diff C D
  and h-hom-compl: h ∈ hom-completion D D
  and fg: f ∘ g = (λx. if x ∈ carrier C then id x else 1_C)
   and gf-dh-hd: (λx. if x ∈ carrier D then (g ∘ f) x ⊗ (if x ∈ carrier D then
((differ) ∘ h) x ⊗ (h ∘ (differ)) x else 1_D) else 1_D) =
  (λx. if x ∈ carrier D then id x else 1_D)
  and fh: f ∘ h = (λx. if x ∈ carrier D then 1_C else 1_C)
  and hg: h ∘ g = (λx. if x ∈ carrier C then 1_D else 1_D)
  and hh: h ∘ h = (λx. if x ∈ carrier D then 1_D else 1_D)
```

Due to the nature of the formula (λx. *if* $x \in$ *carrier D then* $(g \circ f)$ $x \otimes$ (*if* $x \in$ *carrier D then* (*differ* $\circ$ *h*) $x \otimes$ (*h* $\circ$ *differ*) *x else* **1**) *else* **1**) = (λx. *if* $x \in$ *carrier D then id x else* **1**), we associate first the addition of $d \circ h$ and $h \circ d$, and then $g \circ f$

lemma *reductionI*:
  includes *struct D* + *struct C*
  assumes *src-diff-group*: *diff-group D*
  and *trg-diff-group*: *diff-group C*
  assumes $f \in$ *hom-diff D C*
  and $g \in$ *hom-diff C D*
  and $h \in$ *hom-completion D D*
  and $f \circ g$ = (λx. *if* $x \in$ *carrier C then id x else* $\mathbf{1}_C$)
  and (λx. *if* $x \in$ *carrier D then* $(g \circ f)$ $x \otimes$ (*if* $x \in$ *carrier D then* ((*differ*) $\circ$ *h*) $x \otimes$ (*h* $\circ$ (*differ*)) *x else* $\mathbf{1}_D$) *else* $\mathbf{1}_D$) =
  (λx. *if* $x \in$ *carrier D then id x else* $\mathbf{1}_D$)
  and $f \circ h$ = (λx. *if* $x \in$ *carrier D then* $\mathbf{1}_C$ *else* $\mathbf{1}_C$)
  and $h \circ g$ = (λx. *if* $x \in$ *carrier C then* $\mathbf{1}_D$ *else* $\mathbf{1}_D$)
  and $h \circ h$ = (λx. *if* $x \in$ *carrier D then* $\mathbf{1}_D$ *else* $\mathbf{1}_D$)
  shows *reduction D C f g h*
  using *prems* **unfolding** *reduction-def reduction-axioms-def diff-group-def* **by** *simp*

lemma (**in** *reduction*) *C-diff-group*: **shows** *diff-group C* **using** *prems* **unfolding** *reduction-def* **by** *simp*

lemma (**in** *reduction*) *D-diff-group*: **shows** *diff-group D* **using** *prems* **unfolding** *reduction-def* **by** *simp*

lemma (**in** *reduction*) *D-C-f-diff-group-hom-diff*: **shows** *diff-group-hom-diff D C f* **using** *prems* **and** *diff-group-hom-diffI* [*of D C f*]
  unfolding *reduction-def reduction-axioms-def* **by** *auto*

lemma (**in** *reduction*) *D-C-f-group-hom*: **shows** *group-hom D C f* **using** *D-C-f-diff-group-hom-diff*
  unfolding *diff-group-hom-diff-def comm-group-hom-def* **by** *simp*

lemma (**in** *reduction*) *C-D-g-diff-group-hom-diff*: **shows** *diff-group-hom-diff C D g* **using** *prems* **and** *diff-group-hom-diffI* [*of C D g*]
  unfolding *reduction-def reduction-axioms-def* **by** *auto*

lemma (**in** *reduction*) *C-D-g-group-hom*: **shows** *group-hom C D g* **using** *C-D-g-diff-group-hom-diff*
  unfolding *diff-group-hom-diff-def comm-group-hom-def* **by** *simp*

3.1 Definition of isomorphic differential groups

Lemma 2.2.11, which corresponds to the first lemma in the BPL proof, has been already proved in our first approach.

It requires introducing first the notion of isomorphic differential groups; the definition is based on the one of isomorphic monoids presented in Group.thy for homomorphisms, by extending it to be coherent with the differentials.

constdefs
iso-diff :: ('*a*, '*c*) *diff-group-scheme* => ('*b*, '*d*) *diff-group-scheme* => ('*a* => '*b*) *set* (**infixr** $\cong_{diff}$ *60*)
$D \cong_{diff} C$ == {*h*. *h* ∈ *hom-diff D C* & *bij-betw h* (*carrier D*) (*carrier C*)}

lemma *iso-diffI*: **assumes** *closed*: ⋀*x*. *x* ∈ *carrier D* ⟹ *h x* ∈ *carrier C*
and *mult*: ⋀*x y*. ⟦*x* ∈ *carrier D*; *y* ∈ *carrier D*⟧ ⟹ *h* (*x* $\otimes_D$ *y*) = *h x* $\otimes_C$ *h y*
and *complect*: ∃ *g*. *h* = (λ*x*. *if x* ∈ *carrier D then g x else* $\mathbf{1}_C$)
and *coherent*: ⋀*x*. *h* ((*differ*$_D$) *x*) = (*differ*$_C$) (*h x*)
and *inj-on*: ⋀*x y*. ⟦ *x* ∈ *carrier D*; *y* ∈ *carrier D*; *h* (*x*) = *h* (*y*) ⟧ ⟹ *x*=*y*
and *image*: ⋀*y*. *y*∈*carrier C* ⟹ ∃ *x* ∈ *carrier D*. *y* = *h* (*x*)
shows *h* ∈ $D \cong_{diff} C$
using *prems*
unfolding *iso-diff-def* **unfolding** *hom-diff-def* **apply** (*simp add*: *expand-fun-eq*)
unfolding *hom-completion-def* **apply** *simp*
unfolding *completion-fun2-def completion-def* **apply** (*simp add*: *expand-fun-eq*)
unfolding *hom-def* **unfolding** *Pi-def* **apply** *simp*
unfolding *bij-betw-def inj-on-def* **apply** *simp*
unfolding *image-def* **by** *auto*

definition
iso-inv-diff :: ('*a*, '*c*) *diff-group-scheme* => ('*b*, '*d*) *diff-group-scheme* => (('*a* => '*b*) × ('*b* => '*a*)) *set* (**infixr** $\cong_{invdiff}$ *60*)
where $D \cong_{invdiff} C$ == {(*f*, *g*). *f* ∈ ($D \cong_{diff} C$) & *g* ∈ ($C \cong_{diff} D$) & (*f* ∘ *g* = *completion C C id*) & (*g* ∘ *f* = *completion D D id*)}

lemma *iso-inv-diffI*: **assumes** *f*: *f* ∈ ($D \cong_{diff} C$) **and** *g*: *g* ∈ ($C \cong_{diff} D$) **and** *fg-id*: (*f* ∘ *g* = *completion C C id*)
and *gf-id*: (*g* ∘ *f* = *completion D D id*) **shows** (*f*, *g*) ∈ ($D \cong_{invdiff} C$)
using *f g fg-id gf-id* **unfolding** *iso-inv-diff-def* **by** *simp*

lemma *iso-inv-diff-iso-diff*: **assumes** *f-f'*: (*f*, *f'*) ∈ ($D \cong_{invdiff} C$) **shows** *f* ∈ ($D \cong_{diff} C$)
using *f-f'* **unfolding** *iso-inv-diff-def* **by** *simp*

lemma *iso-inv-diff-iso-diff2*: **assumes** *f-f'*: (*f*, *f'*) ∈ ($D \cong_{invdiff} C$) **shows** *f'* ∈ ($C \cong_{diff} D$)
using *f-f'* **unfolding** *iso-inv-diff-def* **by** *simp*

lemma *iso-inv-diff-id*: **assumes** *f-f'*: (*f*, *f'*) ∈ ($D \cong_{invdiff} C$) **shows** *f'* ∘ *f* = *completion D D id*
using *f-f'* **unfolding** *iso-inv-diff-def* **by** *simp*

lemma *iso-inv-diff-id2*: **assumes** *f-f'*: $(f, f') \in (D \cong_{invdiff} C)$ **shows** $f \circ f' =$ *completion C C id*
using *f-f'* **unfolding** *iso-inv-diff-def* **by** *simp*

lemma *iso-inv-diff-rev*: **assumes** *f-f'*: $(f, f') \in (D \cong_{invdiff} C)$ **shows** $(f', f) \in (C \cong_{invdiff} D)$
using *f-f'* **unfolding** *iso-inv-diff-def* **by** *simp*

lemma *iso-diff-hom-diff*: **assumes** *h*: $h \in D \cong_{diff} C$ **shows** $h \in$ *hom-diff D C*
using *h* **unfolding** *iso-diff-def* **by** *simp*

3.2 Previous facts for Lemma 2.2.11

lemma (**in** *reduction*) *g-f-hom-diff*: **shows** $g \circ f \in$ *hom-diff D D*
proof (*unfold hom-diff-def hom-completion-def*, *auto*)
from *f-hom-diff* **and** *g-hom-diff* **have** *f*: $f \in$ *completion-fun2 D C* **and** *g*: $g \in$ *completion-fun2 C D*
by (*unfold hom-diff-def hom-completion-def*, *simp-all*)
show $g \circ f \in$ *completion-fun2 D D*
proof (*unfold completion-fun2-def*, *simp*, *intro exI* [*of - g* $\circ$ *f*], *unfold completion-def*, *auto simp add*: *expand-fun-eq*)
fix *x* **assume** *x*: $x \notin$ *carrier D*
from *C-diff-group D-diff-group g-hom-diff* **and** *completion-closed2* [*OF f x*] **and** *group-hom.hom-one* [*of C D g*]
show $g\ (f\,x) = \mathbf{1}$ **unfolding** *diff-group-def comm-group-def group-def group-hom-def group-hom-axioms-def hom-diff-def hom-completion-def*
by *simp*
qed
next
show $g \circ f \in$ *hom D D*
proof (*intro homI*)
fix *x*
assume *x*: $x \in$ *carrier D*
from *hom-diff-closed* [*OF f-hom-diff x*] **and** *hom-diff-closed* [*OF g-hom-diff*, *of f x*] **show** $(g \circ f)\ x \in$ *carrier D* **by** *simp*
next
fix *x y*
assume *x*: $x \in$ *carrier D* **and** *y*: $y \in$ *carrier D*
from *f-hom-diff g-hom-diff* **and** *x y* **and** *hom-completion-mult* [*of f D C x y*] *hom-completion-mult* [*of g C D f x f y*]
and *hom-diff-closed* [*of f D C x*] *hom-diff-closed* [*of f D C y*]
show $(g \circ f)\ (x \otimes y) = (g \circ f)\ x \otimes (g \circ f)\ y$ **unfolding** *hom-diff-def* **by** *simp*
qed
next
from *hom-diff-coherent* [*OF f-hom-diff*] **and** *hom-diff-coherent* [*OF g-hom-diff*] **and** *o-assoc* [*of g f differ*]
and *o-assoc* [*of g* $differ_C$ *f*] **and** *o-assoc* [*of differ g f*] **show** $g \circ f \circ$ *differ* $=$ *differ* $\circ\ (g \circ f)$ **by** *simp*

qed

lemma (**in** *reduction*) *D-D-g-f-diff-group-hom-diff*: **shows** *diff-group-hom-diff D D* ($g \circ f$) **using** *g-f-hom-diff* **and** *D-diff-group*
and *diff-group-hom-diffI* [*of D D* ($g \circ f$)] **by** *simp-all*

lemma (**in** *reduction*) *D-D-g-f-group-hom*: **shows** *group-hom D D* ($g \circ f$) **using** *D-D-g-f-diff-group-hom-diff*
unfolding *diff-group-hom-diff-def comm-group-hom-def* **by** *simp*

The following lemma proves that, in a general reduction, *f*, *g*, *h*, the set image of $g \circ f$ with the operations inherited from D is a differential group.

lemma (**in** *reduction*) *image-g-f-diff-group*: **shows** *diff-group* (|*carrier* = *image* ($g \circ f$) (*carrier D*), *mult* = *mult D*, *one* = *one D*,
diff = *completion* (|*carrier* = *image* ($g \circ f$) (*carrier D*), *mult* = *mult D*, *one* = *one D*, *diff* = *diff D* |) *D* (*diff D*) |)
using *diff-group-hom-diff.image-diff-group* [*of D D g* $\circ$ *f*] **and** *diff-group-hom-diffI* [*OF D-diff-group D-diff-group g-f-hom-diff*] **by** *simp*

3.3 Lemma 2.2.11

The following lemmas correpond to Lemma 2.2.11 in Aransay's thesis

In the version in the thesis, two differential groups are defined to be isomorphic whenever there exists two homomorphisms f and g such that their composition is the identity in both directions

The Isabelle definition is slightly different, and it requires proving that there exists *one homomorphism*, which is, additionally, injective and surjective

This is the reason why the lemma is proved in Isabelle in four different lemmas; the first two, prove that the isomorphism exists, and then we prove that they are mutually inverse

We first introduce a locale which only contains some abbreviations, the main reason is to shorten proofs and statements

We will avoid the use of record update operations

locale *lemma-2-2-11* = *reduction D C f g h*

FIXME:Probably the following *definitions* would be more suitably stored as *abbreviations* or *notations*

context *lemma-2-2-11*
begin

definition *im-gf* **where** *im-gf* == *image* ($g \circ f$) (*carrier D*)

definition *diff-group-im-gf* **where** *diff-group-im-gf* == (|*carrier* = *image* ($g \circ f$) (*carrier D*), *mult* = *mult D*, *one* = *one D*, *diff* =*completion* (|*carrier* = *image* (*g* ∘ *f*) (*carrier D*), *mult* = *mult D*, *one* = *one D*, *diff* = *diff D*|) *D* (*diff D*)|)

definition *diff-im-gf* **where** *diff-im-gf* == *completion diff-group-im-gf D* (*diff D*)

end

lemma (**in** *lemma-2-2-11*) *lemma-2-2-11-first-part*: **shows** $g \in (C \cong_{diff}$ *diff-group-im-gf*)
proof (*intro iso-diffI*)
 fix *x*
 assume *x*: *x* ∈ *carrier C*
 show *g x* ∈ *carrier diff-group-im-gf*
 proof (*unfold diff-group-im-gf-def image-def*, *simp*, *intro bexI* [*of - g x*])
  from *fg* **and** *x* **show** *g x* = *g* (*f* (*g x*)) **by** (*auto simp add*: *expand-fun-eq*)
 next
  from *hom-diff-closed* [*OF g-hom-diff x*]**show** *g x* ∈ *carrier D* **by** *simp*
 qed
next
 fix *x y*
 assume *x*: *x* ∈ *carrier C* **and** *y*: *y* ∈ *carrier C*
 from *hom-diff-mult* [*OF g-hom-diff x y*] **and** *diff-group-im-gf-def* **show** $g\ (x \otimes_C y) = g\ x \otimes_{diff\text{-}group\text{-}im\text{-}gf} g\ y$ **by** *simp*
next
 from *g-hom-diff* **and** *f-in-completion-fun2-f-completion* [*of g C D*] **and** *completion-def* [*of C D g*] **and** *diff-group-im-gf-def*
 show ∃ *ga*. *g* = (λ*x*. *if x* ∈ *carrier C then ga x else* $\mathbf{1}_{diff\text{-}group\text{-}im\text{-}gf}$)
  unfolding *hom-diff-def hom-completion-def completion-fun2-def expand-fun-eq* **by** (*intro exI* [*of - g*]) *auto*
next
 fix *x*
 show $g\ ((differ_C)\ x) = (differ_{diff\text{-}group\text{-}im\text{-}gf})\ (g\ x)$
 proof (*cases x* ∈ *carrier C*)
  case *True* **then have** *g x* ∈ (*g* ∘ *f*) ' *carrier D*
  proof (*unfold image-def*, *simp*, *intro bexI* [*of - g x*])
   from *fg* **and** *True* **show** *g x* = *g* (*f* (*g x*)) **by** (*simp add*: *expand-fun-eq*)
  next
   from *hom-diff-closed* [*OF g-hom-diff True*] **show** *g x* ∈ *carrier D* **by** *simp*
  qed
  then have $(differ_{diff\text{-}group\text{-}im\text{-}gf})\ (g\ x) = (differ)\ (g\ x)$ **unfolding** *completion-def diff-group-im-gf-def im-gf-def* **by** *auto*
   with *g-hom-diff* **and** *hom-diff-coherent* [*of g C D*] **show** $g\ ((differ_C)\ x) = (differ_{diff\text{-}group\text{-}im\text{-}gf})\ (g\ x)$
   unfolding *diff-group-im-gf-def im-gf-def* **by** (*simp add*: *expand-fun-eq*)
 next
  case *False*
  from *C.diff-hom* **and** *hom-completion-def* [*of C C*] **and** *completion-closed2* [*OF - False*, *of* $differ_C$ *C*] **have** $(differ_C)\ x = \mathbf{1}_C$ **by** *simp*
  with *group-hom.hom-one* [*of C D g*] **and** *C-D-g-group-hom*
  have $g\ ((differ_C)\ x) = \mathbf{1}_D$ **by** *simp*

moreover
from *D-C-f-group-hom C-D-g-group-hom D.diff-hom* **and** *group-hom.hom-one* [*of D C f*] *group-hom.hom-one* [*of C D g*]
group-hom.hom-one [*of D D differ*] **and** *D.one-closed* **and** *False* **and** *diff-group-im-gf-def* **and** *C-diff-group D-diff-group g-hom-diff f-hom-diff*
have *comp-one*: $(differ_{diff\text{-}group\text{-}im\text{-}gf})\ (g\ x) = \mathbf{1}_D$
unfolding *group-hom-def group-hom-axioms-def hom-diff-def hom-completion-def completion-fun2-def completion-def*
diff-group-def comm-group-def **by** *auto*
ultimately show $g\ ((differ_C)\ x) = (differ_{diff\text{-}group\text{-}im\text{-}gf})\ (g\ x)$ **by** *simp*
qed
next
fix *x y*
assume *x*: $x \in$ *carrier C* **and** *y*: $y \in$ *carrier C* **and** *g-eq*: $g\ x = g\ y$
from *g-eq* **have** *fg-eq*: $f\ (g\ x) = f\ (g\ y)$ **by** *simp*
with *fg* **and** *x* **and** *y* **show** $x = y$ **by** (*simp add*: *expand-fun-eq*)
next
fix *y*
assume *y*: $y \in$ *carrier diff-group-im-gf* **then have** $y \in (g \circ f)$‘ (*carrier D*) **by** (*unfold diff-group-im-gf-def*, *simp*)
then obtain *x* **where** $g\ (f\ x) = y$ **and** *x*: $x \in$ *carrier D* **by** *auto*
with *bexI* [*of* $\lambda x.\ (y = g\ x)$ *f x carrier C*] **and** *hom-diff-closed* [*OF f-hom-diff x*] **show** $\exists x \in$*carrier C*. $y = g\ x$ **by** *simp*
qed

The inverse of g is the restriction of f to the image set of $g \circ f$

lemma (**in** *lemma-2-2-11*) *lemma-2-2-11-second-part*: **shows** *completion diff-group-im-gf C f* $\in$ (*diff-group-im-gf* $\cong_{diff}$ *C*)
(**is** *?compl-f* $\in$ (*?IM* $\cong_{diff}$ *C*))
proof (*intro iso-diffI*)
fix *x*
assume *x*: $x \in$ *carrier ?IM* **then have** *x-im*: $x \in (g \circ f)$‘ (*carrier D*) **by** (*unfold diff-group-im-gf-def*, *simp*)
then obtain *y* **where** *gf-y*: $g\ (f\ y) = x$ **and** *y*: $y \in$ *carrier D* **by** *auto*
from *x-im* **have** *?compl-f x* = *f x* **by** (*unfold completion-def diff-group-im-gf-def*, *simp*)
with *gf-y* **and** *hom-diff-closed* [*OF f-hom-diff y*] *hom-diff-closed* [*OF g-hom-diff*, *of f y*] *hom-diff-closed* [*OF f-hom-diff*, *of g* (*f y*)]
show *?compl-f x* $\in$ *carrier C* **by** *simp*
next
fix *x y*
assume *x*: $x \in$ *carrier ?IM* **and** *y*: $y \in$ *carrier ?IM* **then obtain** x' y' **where** *gf-x'*: $g\ (f\ x') = x$ **and** *gf-y'*: $g\ (f\ y') = y$
and *x'*: $x' \in$ *carrier D* **and** *y'*: $y' \in$ *carrier D* **unfolding** *diff-group-im-gf-def* **by** *auto*
with *sym* [*OF hom-diff-mult* [*OF g-hom-diff*, *of f x' f y'*]] **and** *hom-diff-closed* [*OF f-hom-diff x'*] *hom-diff-closed* [*OF f-hom-diff y'*]
and *x y* **and** *sym* [*OF hom-diff-mult* [*OF f-hom-diff x' y'*]]
have *?compl-f* $(x \otimes_{?IM} y) = f\ (x \otimes_D y)$ **unfolding** *diff-group-im-gf-def completion-def*

by *auto*
also from *gf-x′ gf-y′* **have** . . . = f (g (f x') $\otimes_D$ g (f y')) **by** *simp*
also from *hom-diff-closed* [*OF f-hom-diff x′*] *hom-diff-closed* [*OF f-hom-diff y′*] *hom-diff-closed* [*OF g-hom-diff* , *of f x′*]
hom-diff-closed [*OF g-hom-diff* , *of f y′*] **and** *hom-diff-mult* [*OF f-hom-diff*] **have** . . . = f (g (f x')) $\otimes_C$ f (g (f y')) **by** *simp*
also from *gf-x′ gf-y′* **and** x' y' **have** . . . = *?compl-f* x $\otimes_C$ *?compl-f* y **unfolding** *completion-def diff-group-im-gf-def* **by** *auto*
finally show *?compl-f* (x $\otimes_{?IM}$ y) = *?compl-f* x $\otimes_C$ *?compl-f* y **by** *simp*
next
show $\exists$ g. *?compl-f* = (λx. *if* x $\in$ *carrier ?IM then* g x *else* $\mathbf{1}_C$) **unfolding** *completion-def* **by** (*rule exI* [*of - f*]) *simp*
next
fix x
show *?compl-f* ((*differ* $_{?IM}$) x) = (*differ* $_C$) (*?compl-f* x)
proof (*cases* x $\in$ *carrier ?IM*)
case *True* **then obtain** y **where** y: y $\in$ *carrier D* **and** *gf-y*: x = g (f y) **unfolding** *diff-group-im-gf-def* **by** *auto*
then have *?compl-f* ((*differ* $_{?IM}$) x) = *?compl-f* ((*differ*) (g (f y))) **unfolding** *completion-def diff-group-im-gf-def* **by** *auto*
also have . . . = f ((*differ*) (g (f y)))
proof −
from *hom-diff-coherent* [*OF g-hom-diff*] **and** *hom-diff-coherent* [*OF f-hom-diff*] **have** ((*differ*) (g (f y))) = (g (f ((*differ*) y)))
by (*simp add*: *expand-fun-eq*)
with *hom-completion-closed* [*OF D.diff-hom y*] **show** *?thesis* **unfolding** *image-def diff-group-im-gf-def completion-def* **by** *auto*
qed
also from *hom-diff-coherent* [*OF f-hom-diff*] **have** . . . = (*differ* $_C$) (f (g (f y))) **by** (*simp add*: *expand-fun-eq*)
also from y **and** *gf-y* **have** . . . = (*differ* $_C$) (*?compl-f* x) **unfolding** *completion-def image-def diff-group-im-gf-def* **by** *auto*
finally show *?thesis* **by** *simp*
next
case *False* **then have** *diff-one*: ((*differ* $_{?IM}$) x) = $\mathbf{1}$ **unfolding** *diff-group-im-gf-def completion-def* **by** *auto*
from *D-C-f-group-hom* **and** *C-D-g-group-hom* **and** *group-hom.hom-one* [*of D C f*] **and** *group-hom.hom-one* [*of C D g*] **and** *bexI* [*of -* $\mathbf{1}$]
and *diff-group-im-gf-def* **and** *im-gf-def* **have** *one-image*: $\mathbf{1}$ $\in$ (g $\circ$ f)‘ *carrier D* **unfolding** *image-def* **by** *simp*
with *diff-one* **have** *?compl-f* ((*differ* $_{?IM}$) x) = f $\mathbf{1}$ **unfolding** *completion-def diff-group-im-gf-def* **by** *simp*
also from *D-C-f-group-hom* **and** *group-hom.hom-one* [*of D C f*] **have** . . . = $\mathbf{1}_C$ **by** *simp*
finally have *l-h-s*: *?compl-f* ((*differ* $_{?IM}$) x) = $\mathbf{1}_C$ **by** *simp*
from *False* **have** *diff-one*: (*differ* $_C$) (*?compl-f* x) = (*differ* $_C$) $\mathbf{1}_C$ **unfolding** *completion-def* **by** *simp*
also from *C.diff-hom* **and** *group-hom.hom-one* [*of C C* (*differ* $_C$)] **and** *C-diff-group* **have** . . . = $\mathbf{1}_C$

unfolding *group-hom-def group-hom-axioms-def diff-group-def comm-group-def hom-completion-def* **by** *simp*
with *diff-one* **have** *r-h-s*: $(differ_C)$ $(?compl\text{-}f\ x) = \mathbf{1}_C$ **by** *simp*
from *l-h-s* **and** *r-h-s* **show** *?thesis* **by** *simp*
qed
next
fix *x y*
assume *x*: $x \in carrier\ ?IM$ **and** *y*: $y \in carrier\ ?IM$ **then obtain** x' y' **where** *gf-x'*: $g\ (f\ x') = x$ **and** *gf-y'*: $g\ (f\ y') = y$
and x': $x' \in carrier\ D$ **and** y': $y' \in carrier\ D$ **unfolding** *diff-group-im-gf-def* **by** *auto*
assume *eq*: $?compl\text{-}f\ x = ?compl\text{-}f\ y$ **with** *x y* **have** $f\ x = f\ y$ **unfolding** *completion-def* **by** *simp*
with *gf-x' gf-y'* **have** $f\ (g\ (f\ x')) = f\ (g\ (f\ y'))$ **by** *simp*
then have $g\ (f\ (g\ (f\ x'))) = g\ (f\ (g\ (f\ y')))$ **by** *simp*
with *fg* **and** *hom-diff-closed* [*OF f-hom-diff x'*] *hom-diff-closed* [*OF f-hom-diff y'*] **and** *gf-x' gf-y'* **show** $x = y$
by (*auto simp add*: *expand-fun-eq*)
next
fix *y*
assume *y*: $y \in carrier\ C$
from *fg* **have** *fg-idemp*: $(f \circ g) \circ (f \circ g) = (f \circ g)$ **by** (*simp add*: *expand-fun-eq*)
with *bexI* [*of* $\lambda x.\ (y = f\ (g\ (f\ x)))\ g\ y$ *carrier D*] **and** *hom-diff-closed* [*OF g-hom-diff y*] **and** *fg* **and** *y*
show $\exists x \in carrier\ ?IM.\ y = ?compl\text{-}f\ x$ **unfolding** *completion-def diff-group-im-gf-def image-def* **by** (*auto simp add*: *expand-fun-eq*)
qed

We now prove that g and the restriction of f are inverse of each other.

lemma (**in** *lemma-2-2-11*) *lemma-2-2-11-third-part*: **shows** *completion diff-group-im-gf C f* $\circ\ g = (\lambda x.\ if\ x \in carrier\ C\ then\ id\ x\ else\ \mathbf{1}_C)$
(**is** $?compl\text{-}f \circ g = ?id\text{-}C$)
proof (*unfold expand-fun-eq*, *auto*)
fix *x*
assume *x*: $x \in carrier\ C$ **with** *diff-group-im-gf-def* **and** *image-def* [*of* $(g \circ f)$ *carrier D*]
and *bexI* [*of* $\lambda y.\ (g\ x = (g \circ f)\ y)\ g\ x$ *carrier D*] **and** *fg* **and** *hom-diff-closed* [*OF g-hom-diff x*]
show $?compl\text{-}f\ (g\ x) = x$ **unfolding** *completion-def* **by** (*simp add*: *expand-fun-eq*)
next
fix *x*
assume *x*: $x \notin carrier\ C$
from *diff-group-im-gf-def* **and** *g-hom-diff* **and** *completion-closed2* [*OF - x*, *of g D*] **and** *D-C-f-group-hom* **and** *group-hom.hom-one* [*of D C f*]
show $?compl\text{-}f\ (g\ x) = \mathbf{1}_C$ **unfolding** *hom-diff-def hom-completion-def completion-def* **by** *simp*
qed

lemma (**in** *lemma-2-2-11*) *lemma-2-2-11-fourth-part*:

shows $g \circ$ *completion diff-group-im-gf C f* $= (\lambda x.$ *if* $x \in$ *carrier diff-group-im-gf then id x else* $\mathbf{1}_{\textit{diff-group-im-gf}})$
(**is** $g \circ$ *?compl-f* = *?id-IM*)
proof (*unfold diff-group-im-gf-def completion-def expand-fun-eq*, *auto*)
fix *x*
assume *x*: $x \in$ *carrier D*
from *fg* **and** *hom-diff-closed* [*OF f-hom-diff x*] **show** $g\ (f\ (g\ (f\ x))) = g\ (f\ x)$
by (*simp add*: *expand-fun-eq*)
next
from *C-D-g-group-hom* **and** *group-hom.hom-one* [*of C D g*] **show** $g\ \mathbf{1}_C = \mathbf{1}_D$
by *simp*
qed

The following is just the recollection of the four parts in which we have divided the proof of Lemma 2.2.11

The following statement should be compared to Lemma 2.2.11 in Aransay memoir

lemma (**in** *lemma-2-2-11*) *lemma-2-2-11*: **shows** (*g*, *completion diff-group-im-gf C f*) $\in$ ($C \cong_{invdiff}$ *diff-group-im-gf*)
using *lemma-2-2-11-first-part* **and** *lemma-2-2-11-second-part* **and** *lemma-2-2-11-third-part* **and** *lemma-2-2-11-fourth-part*
unfolding *iso-inv-diff-def completion-def* **by** *simp*

end

4 Propositions 2.2.12, 2.2.13 and Lemma 2.2.14 in Aransay's memoir

theory *lemma-2-2-14*
imports
lemma-2-2-11
begin

4.1 Previous definitions for Lemma 2.2.14

In the following we introduce some locale specifications and definitions that will ease our proofs

For instance, we introduce the locale *ring-endomorphisms* which will allow us to apply equational reasoning with endomorphisms

In the *ring-endomorphisms* specification we introduce as an assumption the fact *ring-R*, stating that completion endomorphisms are a ring; we have proved this fact in the library *HomGroupCompletion.thy* and here it should be introduced by means of an *interpretation*, but some technical limitations in the interpretation mechanism led us to introduce this fact as an assumption

```
locale ring-endomorphisms = diff-group D + ring R +
  assumes ring-R: R = (| carrier = hom-completion D D, mult = op o,
  one = (λx. if x ∈ carrier D then id x else 1),
  zero = (λx. if x ∈ carrier D then 1 else 1),
  add = λf. λg. (λx. if x ∈ carrier D then f x ⊗ g x else 1)|)

locale lemma-2-2-14 = ring-endomorphisms D R + var h +
  assumes h-hom: h ∈ hom-completion D D
  and h-nil: h ⊗_R h = 0_R
  and hdh-h: h ⊗_R differ ⊗_R h = h

context lemma-2-2-14
begin

definition p where p == (differ) ⊗_R h ⊕_R h ⊗_R (differ)

definition ker-p where ker-p == kernel D D p

definition diff-group-ker-p where diff-group-ker-p == (| carrier = kernel D D p,
mult = mult D, one = one D,
  diff = completion (|carrier = kernel D D p, mult = mult D, one = one D, diff
= diff D|) D (diff D)|)
definition inc-ker-p where inc-ker-p == (λx. if x ∈ (kernel D D p) then x else
1_D)

end

lemma (in ring-endomorphisms) D-diff-group: shows diff-group D using prems
unfolding ring-endomorphisms-def by simp

lemma (in ring-endomorphisms) diff-in-R [simp]: shows differ ∈ carrier R using
D.diff-hom and ring-R by simp

lemma (in lemma-2-2-14) h-in-R [simp]: shows h ∈ carrier R using h-hom and
ring-R by simp

lemma (in lemma-2-2-14) p-in-R [simp]: shows p ∈ carrier R using p-def by
simp

lemma (in ring-endomorphisms) diff-nilpot[simp]: shows differ ⊗_R differ = 0_R
using ring-R and D.diff-nilpot by simp
```

4.2 Proposition 2.2.12

The following two lemmas correspond to Proposition 2.2.12 in Aransay's memoir

```
lemma (in lemma-2-2-14) p-in-hom-diff: shows p ∈ hom-diff D D
proof (unfold hom-diff-def, simp, intro conjI)
  from ring-R and p-in-R show p ∈ hom-completion D D by simp
```

next
show $p \circ \mathit{differ} = \mathit{differ} \circ p$
proof −

from *ring-R* **have** $p \circ \mathit{differ} = p \otimes_R \mathit{differ}$ **by** *simp*
also from *p-def* **have** $\ldots = ((\mathit{differ}) \otimes_R h \oplus_R h \otimes_R (\mathit{differ})) \otimes_R \mathit{differ}$ **by** *simp*
also from *h-in-R* **and** *diff-in-R* **have** $\ldots = ((\mathit{differ}) \otimes_R h) \otimes_R (\mathit{differ}) \oplus_R (h \otimes_R (\mathit{differ})) \otimes_R \mathit{differ}$ **by** *algebra*
also from *h-in-R* **and** *diff-in-R* **have** $\ldots = (\mathit{differ}) \otimes_R (h \otimes_R (\mathit{differ})) \oplus_R (h \otimes_R (\mathit{differ})) \otimes_R \mathit{differ}$ **by** *algebra*
also from *h-in-R* **and** *diff-in-R* **have** $\ldots = (\mathit{differ}) \otimes_R (h \otimes_R (\mathit{differ})) \oplus_R h \otimes_R (\mathit{differ} \otimes_R \mathit{differ})$ **by** *algebra*
also from *h-in-R* **and** *diff-in-R* **and** *diff-nilpot* **have** $\ldots = (\mathit{differ}) \otimes_R (h \otimes_R (\mathit{differ})) \oplus_R \mathbf{0}_R$ **by** *algebra*
also have $\ldots = (\mathit{differ}) \otimes_R (h \otimes_R (\mathit{differ})) \oplus_R (\mathit{differ} \otimes_R \mathit{differ} \otimes_R h)$ **by** *simp*
also from *h-in-R* **and** *diff-in-R* **have** $\ldots = (\mathit{differ}) \otimes_R (h \otimes_R (\mathit{differ})) \oplus_R \mathit{differ} \otimes_R (\mathit{differ} \otimes_R h)$ **by** *algebra*
also from *h-in-R* **and** *diff-in-R* **have** $\ldots = \mathit{differ} \otimes_R (h \otimes_R \mathit{differ} \oplus_R \mathit{differ} \otimes_R h)$ **by** *algebra*
also from *h-in-R* **and** *diff-in-R* **have** $\ldots = \mathit{differ} \otimes_R (\mathit{differ} \otimes_R h \oplus_R h \otimes_R \mathit{differ})$ **by** *algebra*
also from *p-def* **and** *ring-R* **have** $\ldots = \mathit{differ} \circ p$ **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed

lemma (**in** *lemma-2-2-14*) *ker-p-diff-group*: *diff-group diff-group-ker-p*
using *diff-group-hom-diffI* [*OF D-diff-group D-diff-group p-in-hom-diff*] **and** *diff-group-ker-p-def*
using *diff-group-hom-diff.kernel-diff-group* **by** *simp*

4.3 Proposition 2.2.13

The following lemma corresponds to Proposition 2.2.13 in Aransay's Ph.D.

lemma (**in** *ring-endomorphisms*) *image-subset*: **assumes** *p-in-R*: $p \in \mathit{carrier}\ R$ **and** *p-idemp*: $p \otimes_R p = p$
shows $\mathit{image}\ (\mathbf{1}_R \ominus_R p)\ (\mathit{carrier}\ D) \subseteq \mathit{kernel}\ D\ D\ p$
proof (*unfold image-def kernel-def*, *auto*)
fix x
assume *x-in-D*: $x \in \mathit{carrier}\ D$
from *minus-closed* [*OF one-closed p-in-R*] **and** *ring-R* **have** *one-minus-p*: $(\mathbf{1}_R \ominus_R p) \in \mathit{hom\text{-}completion}\ D\ D$ **by** *simp*
from *hom-completion-closed* [*OF one-minus-p x-in-D*] **show** $(\mathbf{1}_R \ominus_R p)\ x \in \mathit{carrier}\ D$ **by** *simp*
next
fix x
assume $x \in \mathit{carrier}\ D$

show $p\ ((\mathbf{1}_R \ominus_R p)\ x) = \mathbf{1}$
proof −

from *p-in-R* **have** $p \otimes_R (\mathbf{1}_R \ominus_R p) = p \otimes_R \mathbf{1}_R \ominus_R\ p \otimes_R p$ **by** *algebra*
also from *p-in-R* **and** *p-idemp* **have** $\ldots = p \ominus_R\ p$ **by** *simp*
also from *p-in-R* **have** $\ldots = \mathbf{0}_R$ **by** *algebra*
finally have $p \otimes_R (\mathbf{1}_R \ominus_R p) = \mathbf{0}_R$ **by** *simp*
then have $(p \otimes_R (\mathbf{1}_R \ominus_R p))(x) = \mathbf{0}_R\ (x)$ **by** (*simp only*: *expand-fun-eq*)
with *ring-R* **show** *?thesis* **by** *simp*
qed
qed

lemma (**in** *group-hom*) *ker-m-closed*: **assumes** *x-in-ker*: $x \in$ *kernel G H h* **and** *y-in-ker*: $y \in$ *kernel G H h*
shows $x \otimes y \in$ *kernel G H h*
using *x-in-ker* **and** *y-in-ker* **unfolding** *kernel-def* **by** *auto*

4.4 Lemma 2.2.14

The following lemma, proved in a generic ring, will help us to prove that $p = d \otimes_R h \oplus_R h \otimes_R d$ is a projector

lemma (**in** *ring*) *idemp-prod*: **assumes** *a*: $a \in$ *carrier R* **and** *b*: $b \in$ *carrier R* **and** *a-idemp*: $a \otimes a = a$ **and** *b-idemp*: $b \otimes b = b$
and *a-b*: $a \otimes b = \mathbf{0}$ **and** *b-a*: $b \otimes a = \mathbf{0}$ **shows** $(a \oplus b) \otimes (a \oplus b) = (a \oplus b)$
using *a b a-idemp b-idemp a-b b-a* **by** *algebra*

The following lemma corresponds to the first part of Lemma 2.2.14 as stated in Aransay's memoir

lemma (**in** *lemma-2-2-14*) *p-projector*: **shows** $p \otimes_R p = p$
proof −
from *p-def* **have** $p \otimes_R p = (\mathit{differ} \otimes_R h \oplus_R h \otimes_R \mathit{differ}) \otimes_R (\mathit{differ} \otimes_R h \oplus_R h \otimes_R \mathit{differ})$
(**is** $p \otimes_R p = (?d\text{-}h \oplus_R ?h\text{-}d) \otimes_R (?d\text{-}h \oplus_R ?h\text{-}d)$) **by** *simp*
also have $\ldots = (?d\text{-}h \oplus_R ?h\text{-}d)$
proof (*intro ring.idemp-prod*)
from *ring-endomorphisms-def* [*of D R*] **and** *prems* **show** *ring R* **by** (*unfold lemma-2-2-14-def*, *simp*)
show $?d\text{-}h \in$ *carrier R* **by** *simp*
show $?h\text{-}d \in$ *carrier R* **by** *simp*
from *R.m-assoc* [*of differ h differ* $\otimes_R$ *h*] **and** *R.m-assoc* [*of h differ h*] **and** *diff-in-R h-in-R hdh-h*
show $?d\text{-}h \otimes_R ?d\text{-}h = ?d\text{-}h$ **by** *simp*
from *R.m-assoc* [*of h* $\otimes_R$ *differ h differ*] **and** *diff-in-R* **and** *h-in-R* **and** *hdh-h* **show** $?h\text{-}d \otimes_R ?h\text{-}d = ?h\text{-}d$ **by** *simp*
from *h-nil* **and** *R.m-assoc* [*of differ h h* $\otimes_R$ *differ*] **and** *sym* [*OF R.m-assoc* [*of h h differ*]] **show** $?d\text{-}h \otimes_R ?h\text{-}d = \mathbf{0}_R$ **by** *simp*
from *h-nil* **and** *R.m-assoc* [*of h differ differ* $\otimes_R$ *h*] **and** *sym* [*OF R.m-assoc* [*of differ differ h*]] **show** $?h\text{-}d \otimes_R ?d\text{-}h = \mathbf{0}_R$ **by** *simp*

qed
also from *p-def* **have** . . . = *p* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma (**in** *abelian-group*) *minus-equality*:
[| $x \in$ *carrier G*; $y \in$ *carrier G*; $y \oplus x = \mathbf{0}$ |] ==> $\ominus\, x = y$
using *group.inv-equality* [*OF a-group*, *of y x*] **unfolding** *a-inv-def* **by** *simp*

lemma (**in** *abelian-monoid*) *minus-unique*:
[| $x \in$ *carrier G*; $y \in$ *carrier G*; $y' \in$ *carrier G*; $y \oplus x = \mathbf{0}$; $x \oplus y' = \mathbf{0}$ |] ==> $y = y'$
using *monoid.inv-unique* [*OF a-monoid*, *of y x*] **by** *simp*

When proving that R is a ring, you have to give an element such that it satisfies the condition of the additive inverse; nevertheless, when you really want to know the explicit expression of this inverse, there is no direct way to recover it. This makes a difference with the rest of constants and operations in a ring, such as the addition, the product, or the units.

This is the reason why we had to introduce the following lemma, giving us the expression of the additive inverse of any element a in R

lemma (**in** *ring-endomorphisms*) *minus-interpret*: **assumes** *a*: $a \in$ *carrier R*
shows $(\ominus_R\, a) = (\lambda x.$ *if* $x \in$ *carrier D then* inv_D $(a\ x)$ *else* $\mathbf{1}_D)$
proof (*rule abelian-group.minus-equality*)
from *prems* **show** *abelian-group R* **by** *intro-locales*
next
show $a \in$ *carrier R* **by** *assumption*
next
from *ring-R* **and** *a* **and** *comm-group.hom-completion-inv-is-hom-completion* [*of D a*] **and** *D-diff-group*
show *inv-in-R*: (λx. *if* $x \in$ *carrier D then inv a x else* **1**) $\in$ *carrier R* **unfolding** *diff-group-def comm-group-def* **by** *simp*
next
from *ring-R* **and** *a* **and** *hom-completion-closed* [*of a D D*] **show** (λx. *if* $x \in$ *carrier D then inv a x else* **1**) $\oplus_R\ a = \mathbf{0}_R$
by (*auto simp add*: *expand-fun-eq*)
qed

The following proof is a nice example of how we can take advantage of reasoning with endomorphisms as elements of a ring, making use of the automatic tactics for this structure (*by algebra*, . . .)

lemma (**in** *lemma-2-2-14*) *one-minus-p-hom-diff*: **shows** $\mathbf{1}_R \ominus_R p \in$ *hom-diff D D*
proof (*unfold hom-diff-def*, *simp*, *intro conjI*)
from *R.minus-closed* [*OF R.one-closed p-in-R*] **and** *ring-R* **show** $\mathbf{1}_R \ominus_R p \in$ *hom-completion D D* **by** *simp*

next
  show $\mathbf{1}_R \ominus_R p \circ$ *differ* $=$ *differ* $\circ\ \mathbf{1}_R \ominus_R p$
  proof −
    from *ring-R* **have** $\mathbf{1}_R \ominus_R p \circ$ *differ* $= (\mathbf{1}_R \ominus_R p) \otimes_R$ *differ* **by** *simp*
    also from *diff-in-R* **and** *p-in-R* **have** ... = (*differ*) $\ominus_R$ ($p \otimes_R$ *differ*) **by** *algebra*
    also from *ring-R* **and** *hom-diff-coherent* [*OF p-in-hom-diff*] **have** ... = *differ* $\ominus_R$ (*differ* $\otimes_R p$) **by** *simp*
    also from *sym* [*OF R.r-one* [*of differ*]] **have** ... = (*differ* $\otimes_R \mathbf{1}_R) \ominus_R$ (*differ* $\otimes_R p$) **by** *simp*
    also from *diff-in-R* **and** *p-in-R* **have** ... = *differ* $\otimes_R (\mathbf{1}_R \ominus_R p)$ **by** *algebra*
    also from *ring-R* **have** ... = *differ* $\circ$ $(\mathbf{1}_R \ominus_R p)$ **by** *simp*
    finally show *?thesis* **by** *simp*
  qed
qed

The following lemma allows us to change the codomain of a homomorphism, whenever its image set is a subset of the new codomain

lemma (**in** *diff-group-hom-diff*) *h-image-hom-diff*: **assumes** *image-subset*: *image h* (*carrier D*) $\subseteq C'$
  shows $h \in$ *hom-diff D* (| *carrier* = C', *mult* = *mult C*, *one* = *one C*,
  diff = *completion* (|*carrier* = C', *mult* = *mult C*, *one* = *one C*, *diff* = *diff C*|) *C* (*diff C*)|)
proof −
  from *diff-group-hom-diff.hom-diff-h* [*of D C h*] **and** *prems* **have** *h-hom-diff*: $h \in$ *hom-diff D C* **by** *simp*
  with *image-subset* **and** *group-hom.hom-one* [*of C C* $\textit{differ}_C$] **and** *diff-group-hom-diff.group-hom-C-C-differ* [*of D C h*] **and** *prems*
  show *?thesis* **unfolding** *hom-diff-def hom-completion-def hom-def completion-fun2-def completion-def image-def Pi-def*
    unfolding *expand-fun-eq* **by** *auto+*
qed

We denote as inclusion, *inc*, a homomorphism from a subgroup into a group such that it maps every element to the same element

lemma *inc-ker-hom-diff*: **includes** *diff-group D*
  assumes *h-hom-diff*: $h \in$ *hom-diff D D*
  shows (λx. *if* $x \in$ *kernel D D h then x else* $\mathbf{1}_D$) $\in$
  hom-diff (|*carrier* = *kernel D D h*, *mult* = *mult D*, *one* = *one D*,
  diff = *completion* (|*carrier* = *kernel D D h*, *mult* = *mult D*, *one* = *one D*, *diff* = *diff D*|) *D* (*diff D*)|) *D*
  (**is** *?inc-KER* $\in$ *hom-diff ?KER* -)
proof (*unfold hom-diff-def hom-completion-def*, *auto*)
  show *?inc-KER* $\in$ *completion-fun2 ?KER D* **unfolding** *completion-fun2-def* **by** (*auto simp add*: *expand-fun-eq*)
next
  show *?inc-KER* $\in$ *hom ?KER D*
  proof (*rule homI*, *auto*)
    fix *x*

assume *x* ∈ *kernel D D h* **then show** *x* ∈ *carrier D* **unfolding** *kernel-def* **by** *simp*
next
fix *x y*
assume *x-ker*: *x* ∈ *kernel D D h* **and** *y-ker*: *y* ∈ *kernel D D h* **and** *x-times-y*: $x \otimes y \notin$ *kernel D D h*
from *group-hom.ker-m-closed* [*of D D h x y*] **and** *prems* **show** $\mathbf{1} = x \otimes y$
unfolding *group-hom-def group-hom-axioms-def hom-diff-def hom-completion-def diff-group-def comm-group-def* **by** *simp*
qed
next
show *?inc-KER* ∘ *completion* (|*carrier* = *kernel D D h*, *mult* = *mult D*, *one* = *one D*, *diff* = *diff D*|) *D* (*differ*) = *differ* ∘ *?inc-KER*
(**is** *?inc-KER* ∘ *?compl-diff* = *differ* ∘ *?inc-KER*)
proof (*rule ext*)
fix *x*
show (*?inc-KER* ∘ *?compl-diff*) *x* = (*differ* ∘ *?inc-KER*) *x*
proof (*cases x* ∈ *kernel D D h*)
case *True*
from *True* **and** *diff-group.diff-hom* [*of D*] **and** *hom-completion-closed* [*of differ D D x*] **and**
hom-diff-def [*of D D*] **and** *h-hom-diff* **and** *expand-fun-eq* [*of h* ∘ *differ differ* ∘ *h*] **and** *group-hom.hom-one* [*of D D differ*]
and *prems*
show (*?inc-KER* ∘ *?compl-diff*) *x* = (*differ* ∘ *?inc-KER*) *x*
unfolding *diff-group-def comm-group-def group-hom-def*
group-hom-axioms-def kernel-def subset-eq hom-diff-def hom-completion-def
by *simp*
next
case *False*
from *False* **and** *diff-group.diff-hom* [*of D*] **and** *group-hom.hom-one* [*of D D differ*] **and** *prems*
show (*?inc-KER* ∘ *?compl-diff*) *x* = (*differ* ∘ *?inc-KER*) *x*
unfolding *diff-group-def comm-group-def group-hom-def group-hom-axioms-def hom-completion-def* **by** *auto*
qed
qed
qed

The following lemma corresponds to the second part of Lemma 2.2.14 in Aransay's memoir; we prove that a given triple of homomorphisms is a reduction

lemma (**in** *lemma-2-2-14*) *lemma-2-2-14*: **shows** *reduction D diff-group-ker-p* ($\mathbf{1}_R \ominus_R p$) *inc-ker-p h*
(**is** *reduction D ?KER* ($\mathbf{1}_R \ominus_R p$) *?inc-KER h*)
proof (*intro reductionI*)
from *D-diff-group* **show** *diff-group D* **by** *simp*
next
from *D-diff-group diff-group-hom-diff.kernel-diff-group* [*of D D p*] **and** *diff-group-ker-p-def*

and *diff-group-hom-diffI* [*OF - - p-in-hom-diff*]
show *diff-group ?KER* **by** *simp*
next
from *diff-group-hom-diff.h-image-hom-diff* [*of D D* $\mathbf{1}_R \ominus_R$ *p kernel D D p*] **and** *diff-group-hom-diffI* [*OF - - one-minus-p-hom-diff*]
and *image-subset* [*OF p-in-R p-projector*] **and** *D-diff-group* **and** *diff-group-ker-p-def*
show $\mathbf{1}_R \ominus_R p \in$ *hom-diff D ?KER* **by** *simp*
next
from *D-diff-group* **and** *inc-ker-hom-diff* [*OF - p-in-hom-diff*] **and** *diff-group-ker-p-def* **and** *inc-ker-p-def*
show *?inc-KER* $\in$ *hom-diff ?KER D* **by** *simp*
next
from *h-hom* **show** $h \in$ *hom-completion D D* **by** *simp*
next

show $\mathbf{1}_R \ominus_R p \circ$ *?inc-KER* = ($\lambda x.$ *if* $x \in$ *carrier ?KER then id x else* $\mathbf{1}_{?KER}$)
(**is** $\mathbf{1}_R \ominus_R p \circ$ *?inc-KER* = *?id-KER*)
proof −

from *p-in-R* **have** $(\mathbf{1}_R \ominus_R p) = (\mathbf{1}_R \oplus_R (\ominus_R p))$ **by** *algebra*
also from *ring-R* **and** *minus-interpret* [*OF p-in-R*]
have ... = ($\lambda x.$ *if* $x \in$ *carrier D then* ($\lambda x.$ *if* $x \in$ *carrier D then id x else* $\mathbf{1}_D$) *x*
$\otimes$ ($\lambda x.$ *if* $x \in$ *carrier D then inv p x else* $\mathbf{1}_D$) *x else* $\mathbf{1}_D$) **by** *simp*
finally have *one-minus-p*: $(\mathbf{1}_R \ominus_R p)$ = ($\lambda x.$ *if* $x \in$ *carrier D then* ($\lambda x.$ *if* $x \in$ *carrier D then id x else* $\mathbf{1}_D$) *x*
$\otimes$ ($\lambda x.$ *if* $x \in$ *carrier D then inv p x else* $\mathbf{1}_D$) *x else* $\mathbf{1}_D$) **by** *simp*
then show $\mathbf{1}_R \ominus_R p \circ$ *?inc-KER* = *?id-KER*
proof −
from *group-hom.hom-one* [*of D D p*] **and** *p-in-hom-diff* **and** *D-diff-group* **have** $p\ \mathbf{1} = \mathbf{1}$
unfolding *group-hom-def group-hom-axioms-def hom-diff-def diff-group-def comm-group-def hom-completion-def* **by** *simp*
then have *inv-p-one*: *inv p* $\mathbf{1} = \mathbf{1}$ **by** *simp*
with *one-minus-p* **and** *diff-group-ker-p-def* **and** *inc-ker-p-def* **show** *?thesis* **by** (*auto simp add*: *kernel-def expand-fun-eq*)
qed
qed
next

show ($\lambda x.$ *if* $x \in$ *carrier D then*
(*?inc-KER* $\circ\ \mathbf{1}_R \ominus_R p$) $x \otimes$ (*if* $x \in$ *carrier D then* (*differ* $\circ$ *h*) $x \otimes$ (*h* $\circ$ *differ*) *x else* $\mathbf{1}$) *else* $\mathbf{1}$) =
($\lambda x.$ *if* $x \in$ *carrier D then id x else* $\mathbf{1}$)

proof −
from *ring-R*
have ($\lambda x.$ *if* $x \in$ *carrier D then* (*?inc-KER* $\circ\ \mathbf{1}_R \ominus_R p$) $x \otimes$ (*if* $x \in$ *carrier D*

then (*differ* ∘ *h*) *x* ⊗ (*h* ∘ *differ*) *x* *else* **1**) *else* **1**)
= (*?inc-KER* $\otimes_R$ ($\mathbf{1}_R \ominus_R p$) $\oplus_R$ (*differ* $\otimes_R$ *h* $\oplus_R$ *h* $\otimes_R$ *differ*)) **by** (*simp add*: *expand-fun-eq*)
also have ... = ($\mathbf{1}_R \ominus_R p$) $\oplus_R$ (*differ* $\otimes_R$ *h* $\oplus_R$ *h* $\otimes_R$ *differ*)
proof −
from *ring-R* **have** *?inc-KER* $\otimes_R$ ($\mathbf{1}_R \ominus_R p$) = *?inc-KER* ∘ $\mathbf{1}_R \ominus_R p$ **by** *simp*
also have *?inc-KER* ∘ $\mathbf{1}_R \ominus_R p$ = $\mathbf{1}_R \ominus_R p$
proof (*rule ext*)
fix *x*
show (*?inc-KER* ∘ $\mathbf{1}_R \ominus_R p$) *x* = ($\mathbf{1}_R \ominus_R p$) *x*
proof (*cases x* ∈ *carrier D*)
case *True* **with** *image-subset* [*OF p-in-R p-projector*] **have** ($\mathbf{1}_R \ominus_R p$) *x* ∈ *kernel D D p* **by** (*auto simp add*: *imageI*)
with *inc-ker-p-def* **show** *?thesis* **by** *simp*
next
case *False* **from** *inc-ker-p-def* **and** *minus-closed* [*OF one-closed p-in-R*] **and** *ring-R*
and *completion-closed2* [*OF - False*, *of* ($\mathbf{1}_R \ominus_R p$) *D*] **show** *?thesis* **unfolding** *hom-completion-def* **by** *simp*
qed
qed
finally have *?inc-KER* $\otimes_R$ ($\mathbf{1}_R \ominus_R p$) = $\mathbf{1}_R \ominus_R p$ **by** *simp*
then show (*?inc-KER* $\otimes_R$ ($\mathbf{1}_R \ominus_R p$) $\oplus_R$ (*differ* $\otimes_R$ *h* $\oplus_R$ *h* $\otimes_R$ *differ*)) = ($\mathbf{1}_R \ominus_R p$) $\oplus_R$ (*differ* $\otimes_R$ *h* $\oplus_R$ *h* $\otimes_R$ *differ*) **by** *simp*
qed
also from *p-def* **have** ... = ($\mathbf{1}_R \ominus_R p$) $\oplus_R$ *p* **by** *simp*
also from *p-in-R* **have** ... = $\mathbf{1}_R$ **by** *algebra*
also from *ring-R* **have** ... = (λ*x*. *if x* ∈ *carrier D then id x else* **1**) **by** *simp*
finally show *?thesis* **by** *simp*
qed
next

from *prems* **show** $\mathbf{1}_R \ominus_R p$ ∘ *h* = (λ*x*. *if x* ∈ *carrier D then* $\mathbf{1}_{?KER}$ *else* $\mathbf{1}_{?KER}$)
(**is** $\mathbf{1}_R \ominus_R p$ ∘ *h* = *?zero-R*)
proof −

from *ring-R* **have** $\mathbf{1}_R \ominus_R p$ ∘ *h* = ($\mathbf{1}_R \ominus_R p$) $\otimes_R$ *h* **by** *simp*
also from *h-in-R* **and** *p-in-R* **have** ... = *h* $\ominus_R$ (*p* $\otimes_R$ *h*) **by** *algebra*
also from *p-def* **have** ... = *h* $\ominus_R$ ((*differ* $\otimes_R$ *h* $\oplus_R$ *h* $\otimes_R$ *differ*) $\otimes_R$ *h*) **by** *simp*
also from *diff-in-R* **and** *h-in-R* **and** *hdh-h* **and** *h-nil* **have** ... = *h* $\ominus_R$ *h* **by** *algebra*
also from *h-in-R* **have** ... = $\mathbf{0}_R$ **by** *algebra*
also from *ring-R* **and** *diff-group-ker-p-def* **have** ... = *?zero-R* **by** *simp*
finally show $\mathbf{1}_R \ominus_R p$ ∘ *h* = *?zero-R* **by** *simp*
qed
next
show *h* ∘ *?inc-KER* = (λ*x*. *if x* ∈ *carrier diff-group-ker-p then* **1** *else* **1**)

(**is** *h* ∘ *?inc-KER* = *?zero-KER*)

proof (*unfold inc-ker-p-def diff-group-ker-p-def*, *auto simp add*: *expand-fun-eq*)

fix *x*
assume *x-in-ker*: $x \in$ *kernel D D p*
show $h\ x = \mathbf{1}$
proof −
from *hdh-h* **and** *ring-R* **and** *expand-fun-eq* [*of h* ∘ *differ* ∘ *h h*] **have** *h x* = *h* ((*differ*) (*h x*)) **by** *auto*
also from *sym* [*OF D.r-one* [*of h* ((*differ*) (*h x*))]] **and** *hom-completion-closed* [*OF h-hom*, *of x*]
and *x-in-ker* **and** *hom-completion-closed* [*of differ D D h x*] **and** *D.diff-hom*

and *hom-completion-closed* [*OF h-hom*, *of* (*differ*) (*h x*)] **have** ... = *h* ((*differ*) (*h x*)) $\otimes_D$ **1** **by** (*unfold kernel-def*, *simp*)
also from *h-nil* **and** *ring-R* **have** ... = *h* ((*differ*) (*h x*)) $\otimes_D$ *h* (*h* ((*differ*) *x*)) **by** (*simp add*: *expand-fun-eq*)
also from *sym* [*OF hom-completion-mult* [*OF h-hom*, *of* (*differ*) (*h x*) *h* ((*differ*) *x*)]] **and** *D.diff-hom*
and *hom-completion-closed* [*OF h-hom*, *of* (*differ*) *x*] *hom-completion-closed* [*OF D.diff-hom*, *of x*]
and *hom-completion-closed* [*OF D.diff-hom*, *of h x*] *hom-completion-closed* [*OF h-hom*, *of x*] **and** *x-in-ker*
have ... = *h* ((*differ*) (*h x*) $\otimes_D$ *h* ((*differ*) *x*)) **unfolding** *kernel-def* **by** *simp*
also from *p-def* **and** *ring-R* **and** *x-in-ker* **have** ... = *h* (*p* (*x*)) **unfolding** *expand-fun-eq kernel-def* **by** *simp*
also from *x-in-ker* **have** ... = *h* $\mathbf{1}_D$ **unfolding** *kernel-def* **by** *simp*
also from *hom-completion-one* [*OF - - h-hom*] **and** *D-diff-group* **have** ... = $\mathbf{1}_D$ **unfolding** *diff-group-def comm-group-def group-def* **by** *simp*
finally show $h\ x = \mathbf{1}$ **by** *simp*
qed
next

fix *x* **assume** $x \notin$ *kernel D D p*
from *hom-completion-one* [*OF - - h-hom*] **and** *D-diff-group* **show** *h* $\mathbf{1}_D = \mathbf{1}_D$ **unfolding** *diff-group-def comm-group-def group-def* **by** *simp*
qed
next

from *h-nil* **and** *ring-R* **show** *h* ∘ *h* = (λ*x*. *if x* ∈ *carrier D then* **1** *else* **1**) **by** *simp*
qed

end

5 Infinite Sets and Related Concepts

theory *Infinite-Set*

imports *ATP-Linkup*
begin

5.1 Infinite Sets

Some elementary facts about infinite sets, mostly by Stefan Merz. Beware! Because "infinite" merely abbreviates a negation, these lemmas may not work well with *blast*.

```
abbreviation
  infinite :: 'a set ⇒ bool where
  infinite S == ¬ finite S
```

Infinite sets are non-empty, and if we remove some elements from an infinite set, the result is still infinite.

```
lemma infinite-imp-nonempty: infinite S ==> S ≠ {}
  by auto

lemma infinite-remove:
  infinite S ⟹ infinite (S − {a})
  by simp

lemma Diff-infinite-finite:
  assumes T: finite T and S: infinite S
  shows infinite (S − T)
  using T
proof induct
  from S
  show infinite (S − {}) by auto
next
  fix T x
  assume ih: infinite (S − T)
  have S − (insert x T) = (S − T) − {x}
    by (rule Diff-insert)
  with ih
  show infinite (S − (insert x T))
    by (simp add: infinite-remove)
qed

lemma Un-infinite: infinite S ⟹ infinite (S ∪ T)
  by simp

lemma infinite-super:
  assumes T: S ⊆ T and S: infinite S
  shows infinite T
proof
  assume finite T
  with T have finite S by (simp add: finite-subset)
  with S show False by simp
```

```
qed
```

As a concrete example, we prove that the set of natural numbers is infinite.

```
lemma finite-nat-bounded:
  assumes S: finite (S::nat set)
  shows ∃k. S ⊆ {..<k} (is ∃k. ?bounded S k)
using S
proof induct
  have ?bounded {} 0 by simp
  then show ∃k. ?bounded {} k ..
next
  fix S x
  assume ∃k. ?bounded S k
  then obtain k where k: ?bounded S k ..
  show ∃k. ?bounded (insert x S) k
  proof (cases x < k)
    case True
    with k show ?thesis by auto
  next
    case False
    with k have ?bounded S (Suc x) by auto
    then show ?thesis by auto
  qed
qed

lemma finite-nat-iff-bounded:
  finite (S::nat set) = (∃k. S ⊆ {..<k}) (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs by (rule finite-nat-bounded)
next
  assume ?rhs
  then obtain k where S ⊆ {..<k} ..
  then show finite S
    by (rule finite-subset) simp
qed

lemma finite-nat-iff-bounded-le:
  finite (S::nat set) = (∃k. S ⊆ {..k}) (is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain k where S ⊆ {..<k}
    by (blast dest: finite-nat-bounded)
  then have S ⊆ {..k} by auto
  then show ?rhs ..
next
  assume ?rhs
  then obtain k where S ⊆ {..k} ..
  then show finite S
```

```
    by (rule finite-subset) simp
qed

lemma infinite-nat-iff-unbounded:
  infinite (S::nat set) = (∀ m. ∃ n. m<n ∧ n∈S)
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  show ?rhs
  proof (rule ccontr)
    assume ¬ ?rhs
    then obtain m where m: ∀ n. m<n ⟶ n∉S by blast
    then have S ⊆ {..m}
      by (auto simp add: sym [OF linorder-not-less])
    with ‹?lhs› show False
      by (simp add: finite-nat-iff-bounded-le)
  qed
next
  assume ?rhs
  show ?lhs
  proof
    assume finite S
    then obtain m where S ⊆ {..m}
      by (auto simp add: finite-nat-iff-bounded-le)
    then have ∀ n. m<n ⟶ n∉S by auto
    with ‹?rhs› show False by blast
  qed
qed

lemma infinite-nat-iff-unbounded-le:
  infinite (S::nat set) = (∀ m. ∃ n. m≤n ∧ n∈S)
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  show ?rhs
  proof
    fix m
    from ‹?lhs› obtain n where m<n ∧ n∈S
      by (auto simp add: infinite-nat-iff-unbounded)
    then have m≤n ∧ n∈S by simp
    then show ∃ n. m ≤ n ∧ n ∈ S ..
  qed
next
  assume ?rhs
  show ?lhs
  proof (auto simp add: infinite-nat-iff-unbounded)
    fix m
    from ‹?rhs› obtain n where Suc m ≤ n ∧ n∈S
      by blast
```

```
    then have m<n ∧ n∈S by simp
    then show ∃ n. m < n ∧ n ∈ S ..
  qed
qed
```

For a set of natural numbers to be infinite, it is enough to know that for any number larger than some k, there is some larger number that is an element of the set.

```
lemma unbounded-k-infinite:
  assumes k: ∀ m. k<m ⟶ (∃ n. m<n ∧ n∈S)
  shows infinite (S::nat set)
proof −
  {
    fix m have ∃ n. m<n ∧ n∈S
    proof (cases k<m)
      case True
      with k show ?thesis by blast
    next
      case False
      from k obtain n where Suc k < n ∧ n∈S by auto
      with False have m<n ∧ n∈S by auto
      then show ?thesis ..
    qed
  }
  then show ?thesis
    by (auto simp add: infinite-nat-iff-unbounded)
qed

lemma nat-infinite [simp]: infinite (UNIV :: nat set)
  by (auto simp add: infinite-nat-iff-unbounded)

lemma nat-not-finite [elim]: finite (UNIV::nat set) ⟹ R
  by simp
```

Every infinite set contains a countable subset. More precisely we show that a set S is infinite if and only if there exists an injective function from the naturals into S.

```
lemma range-inj-infinite:
  inj (f::nat ⇒ 'a) ⟹ infinite (range f)
proof
  assume inj f
    and  finite (range f)
  then have finite (UNIV::nat set)
    by (auto intro: finite-imageD simp del: nat-infinite)
  then show False by simp
qed

lemma int-infinite [simp]:
  shows infinite (UNIV::int set)
```

proof –
  from *inj-int* **have** *infinite* (*range int*) **by** (*rule range-inj-infinite*)
  moreover
  have *range int* ⊆ (*UNIV*::*int set*) **by** *simp*
  ultimately show *infinite* (*UNIV*::*int set*) **by** (*simp add*: *infinite-super*)
qed

The "only if" direction is harder because it requires the construction of a sequence of pairwise different elements of an infinite set S. The idea is to construct a sequence of non-empty and infinite subsets of S obtained by successively removing elements of S.

lemma *linorder-injI*:
  assumes *hyp*: !!*x y*. *x* < (*y*::'*a*::*linorder*) ==> *f x* ≠ *f y*
  shows *inj f*
proof (*rule inj-onI*)
  fix *x y*
  assume *f-eq*: *f x* = *f y*
  show *x* = *y*
  proof (*rule linorder-cases*)
    assume *x* < *y*
    with *hyp* **have** *f x* ≠ *f y* **by** *blast*
    with *f-eq* **show** *?thesis* **by** *simp*
  next
    assume *x* = *y*
    then show *?thesis* **.**
  next
    assume *y* < *x*
    with *hyp* **have** *f y* ≠ *f x* **by** *blast*
    with *f-eq* **show** *?thesis* **by** *simp*
  qed
qed

lemma *infinite-countable-subset*:
  assumes *inf*: *infinite* (*S*::'*a set*)
  shows ∃*f*. *inj* (*f*::*nat* ⇒ '*a*) ∧ *range f* ⊆ *S*
proof –
  def *Sseq* ≡ *nat-rec S* (λ*n T*. *T* – {*SOME e*. *e* ∈ *T*})
  def *pick* ≡ λ*n*. (*SOME e*. *e* ∈ *Sseq n*)
  have *Sseq-inf*: ⋀*n*. *infinite* (*Sseq n*)
  proof –
    fix *n*
    show *infinite* (*Sseq n*)
    proof (*induct n*)
      from *inf* **show** *infinite* (*Sseq 0*)
        by (*simp add*: *Sseq-def*)
    next
      fix *n*
      assume *infinite* (*Sseq n*) **then show** *infinite* (*Sseq* (*Suc n*))
        by (*simp add*: *Sseq-def infinite-remove*)

```
  qed
qed
have Sseq-S: ⋀n. Sseq n ⊆ S
proof −
  fix n
  show Sseq n ⊆ S
    by (induct n) (auto simp add: Sseq-def)
qed
have Sseq-pick: ⋀n. pick n ∈ Sseq n
proof −
  fix n
  show pick n ∈ Sseq n
  proof (unfold pick-def, rule someI-ex)
    from Sseq-inf have infinite (Sseq n) .
    then have Sseq n ≠ {} by auto
    then show ∃x. x ∈ Sseq n by auto
  qed
qed
with Sseq-S have rng: range pick ⊆ S
  by auto
have pick-Sseq-gt: ⋀n m. pick n ∉ Sseq (n + Suc m)
proof −
  fix n m
  show pick n ∉ Sseq (n + Suc m)
    by (induct m) (auto simp add: Sseq-def pick-def)
qed
have pick-pick: ⋀n m. pick n ≠ pick (n + Suc m)
proof −
  fix n m
  from Sseq-pick have pick (n + Suc m) ∈ Sseq (n + Suc m) .
  moreover from pick-Sseq-gt
  have pick n ∉ Sseq (n + Suc m) .
  ultimately show pick n ≠ pick (n + Suc m)
    by auto
qed
have inj: inj pick
proof (rule linorder-injI)
  fix i j :: nat
  assume i < j
  show pick i ≠ pick j
  proof
    assume eq: pick i = pick j
    from ‹i < j› obtain k where j = i + Suc k
      by (auto simp add: less-iff-Suc-add)
    with pick-pick have pick i ≠ pick j by simp
    with eq show False by simp
  qed
qed
from rng inj show ?thesis by auto
```

qed

lemma *infinite-iff-countable-subset*:
  infinite $S = (\exists f.\ inj\ (f{::}nat \Rightarrow 'a) \wedge range\ f \subseteq S)$
  by (*auto simp add*: *infinite-countable-subset range-inj-infinite infinite-super*)

For any function with infinite domain and finite range there is some element that is the image of infinitely many domain elements. In particular, any infinite sequence of elements from a finite set contains some element that occurs infinitely often.

lemma *inf-img-fin-dom*:
  assumes *img*: *finite* (*f*‘*A*) **and** *dom*: *infinite A*
  shows $\exists\, y \in f`A.$ *infinite* $(f\ -`\ \{y\})$
proof (*rule ccontr*)
  assume ¬ *?thesis*
  with *img* **have** *finite* (*UN y*:*f*‘*A*. $f\ -`\ \{y\}$) **by** (*blast intro*: *finite-UN-I*)
  moreover have $A \subseteq$ (*UN y*:*f*‘*A*. $f\ -`\ \{y\}$) **by** *auto*
  moreover note *dom*
  ultimately show *False* **by** (*simp add*: *infinite-super*)
qed

lemma *inf-img-fin-domE*:
  assumes *finite* (*f*‘*A*) **and** *infinite A*
  obtains *y* **where** $y \in f`A$ **and** *infinite* $(f\ -`\ \{y\})$
  using *assms* **by** (*blast dest*: *inf-img-fin-dom*)

5.2 Infinitely Many and Almost All

We often need to reason about the existence of infinitely many (resp., all but finitely many) objects satisfying some predicate, so we introduce corresponding binders and their proof rules.

definition
  Inf-many :: $('a \Rightarrow bool) \Rightarrow bool$ (**binder** *INFM 10*) **where**
  Inf-many P = *infinite* $\{x.\ P\ x\}$

definition
  Alm-all :: $('a \Rightarrow bool) \Rightarrow bool$ (**binder** *MOST 10*) **where**
  Alm-all P = (¬ (*INFM x*. ¬ *P x*))

notation (*xsymbols*)
  Inf-many (**binder** $\exists_\infty$ *10*) **and**
  Alm-all (**binder** $\forall_\infty$ *10*)

notation (*HTML* **output**)
  Inf-many (**binder** $\exists_\infty$ *10*) **and**
  Alm-all (**binder** $\forall_\infty$ *10*)

lemma *INF-EX*:

$(\exists_\infty x.\ P\ x) \Longrightarrow (\exists x.\ P\ x)$
unfolding *Inf-many-def*
proof (*rule ccontr*)
assume *inf*: *infinite* $\{x.\ P\ x\}$
assume $\neg$ *?thesis* **then have** $\{x.\ P\ x\} = \{\}$ **by** *simp*
then have *finite* $\{x.\ P\ x\}$ **by** *simp*
with *inf* **show** *False* **by** *simp*
qed

lemma *MOST-iff-finiteNeg*: $(\forall_\infty x.\ P\ x) = \mathit{finite}\ \{x.\ \neg\ P\ x\}$
by (*simp add*: *Alm-all-def Inf-many-def*)

lemma *ALL-MOST*: $\forall x.\ P\ x \Longrightarrow \forall_\infty x.\ P\ x$
by (*simp add*: *MOST-iff-finiteNeg*)

lemma *INF-mono*:
assumes *inf*: $\exists_\infty x.\ P\ x$ **and** *q*: $\bigwedge x.\ P\ x \Longrightarrow Q\ x$
shows $\exists_\infty x.\ Q\ x$
proof $-$
from *inf* **have** *infinite* $\{x.\ P\ x\}$ **unfolding** *Inf-many-def* **.**
moreover from *q* **have** $\{x.\ P\ x\} \subseteq \{x.\ Q\ x\}$ **by** *auto*
ultimately show *?thesis*
by (*simp add*: *Inf-many-def infinite-super*)
qed

lemma *MOST-mono*: $\forall_\infty x.\ P\ x \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow Q\ x) \Longrightarrow \forall_\infty x.\ Q\ x$
unfolding *Alm-all-def* **by** (*blast intro*: *INF-mono*)

lemma *INF-nat*: $(\exists_\infty n.\ P\ (n{::}nat)) = (\forall m.\ \exists n.\ m<n \wedge P\ n)$
by (*simp add*: *Inf-many-def infinite-nat-iff-unbounded*)

lemma *INF-nat-le*: $(\exists_\infty n.\ P\ (n{::}nat)) = (\forall m.\ \exists n.\ m\leq n \wedge P\ n)$
by (*simp add*: *Inf-many-def infinite-nat-iff-unbounded-le*)

lemma *MOST-nat*: $(\forall_\infty n.\ P\ (n{::}nat)) = (\exists m.\ \forall n.\ m<n \longrightarrow P\ n)$
by (*simp add*: *Alm-all-def INF-nat*)

lemma *MOST-nat-le*: $(\forall_\infty n.\ P\ (n{::}nat)) = (\exists m.\ \forall n.\ m\leq n \longrightarrow P\ n)$
by (*simp add*: *Alm-all-def INF-nat-le*)

5.3 Enumeration of an Infinite Set

The set's element type must be wellordered (e.g. the natural numbers).

consts
enumerate :: *'a::wellorder set => (nat => 'a::wellorder)*
primrec
enumerate-0: *enumerate S 0* $=$ (*LEAST n.* $n \in S$)
enumerate-Suc: *enumerate S* (*Suc n*) $=$ *enumerate* ($S - \{$*LEAST n.* $n \in S\}$) *n*

lemma *enumerate-Suc'*:
  enumerate S (Suc n) = enumerate (S − {enumerate S 0}) n
 by *simp*

lemma *enumerate-in-set*: *infinite S ⟹ enumerate S n : S*
 apply (*induct n arbitrary*: *S*)
  apply (*fastsimp intro*: *LeastI dest*!: *infinite-imp-nonempty*)
 apply (*fastsimp iff*: *finite-Diff-singleton*)
 done

declare *enumerate-0* [*simp del*] *enumerate-Suc* [*simp del*]

lemma *enumerate-step*: *infinite S ⟹ enumerate S n < enumerate S (Suc n)*
 apply (*induct n arbitrary*: *S*)
  apply (*rule order-le-neq-trans*)
   apply (*simp add*: *enumerate-0 Least-le enumerate-in-set*)
  apply (*simp only*: *enumerate-Suc'*)
  apply (*subgoal-tac enumerate (S − {enumerate S 0}) 0 : S − {enumerate S 0}*)
   apply (*blast intro*: *sym*)
  apply (*simp add*: *enumerate-in-set del*: *Diff-iff*)
 apply (*simp add*: *enumerate-Suc'*)
 done

lemma *enumerate-mono*: *m<n ⟹ infinite S ⟹ enumerate S m < enumerate S n*
 apply (*erule less-Suc-induct*)
 apply (*auto intro*: *enumerate-step*)
 done

5.4 Miscellaneous

A few trivial lemmas about sets that contain at most one element. These simplify the reasoning about deterministic automata.

definition
 atmost-one :: *'a set ⇒ bool* **where**
 atmost-one S = (∀ x y. x∈S ∧ y∈S ⟶ x=y)

lemma *atmost-one-empty*: *S = {} ⟹ atmost-one S*
 by (*simp add*: *atmost-one-def*)

lemma *atmost-one-singleton*: *S = {x} ⟹ atmost-one S*
 by (*simp add*: *atmost-one-def*)

lemma *atmost-one-unique* [*elim*]: *atmost-one S ⟹ x ∈ S ⟹ y ∈ S ⟹ y = x*
 by (*simp add*: *atmost-one-def*)

end

6 Definition of local nilpotency and Lemmas 2.2.1 to 2.2.6 in Aransay's memoir

theory *analytic-part-local*
  imports
  lemma-2-2-14
  ~~/src/HOL/Library/Infinite-Set
  begin

6.1 Definition of local nilpotent element and the bound function

locale *local-nilpotent-term = ring-endomorphisms D R + var a + var bound-funct* +
  constrains *bound-funct* :: *'a => nat*
  assumes *a-in-R*: $a \in carrier\ R$
  and *a-local-nilpot*: $\forall x \in carrier\ D.\ (a\ (\hat{\ })_R\ (bound\text{-}funct\ x))\ x = \mathbf{1}_D$
  and *bound-is-least*: $bound\text{-}funct\ x = (LEAST\ n.\ (a\ (\hat{\ })_R\ (n::nat))\ x = \mathbf{1}_D)$

The following lemma maybe could be included in the *Group.thy* file; there is already a lemma called $\mathbf{1}\ (\hat{\ })\ ?n = \mathbf{1}$, about $\mathbf{1}$, but nothing about $x\ (\hat{\ })\ (1::'c)$

lemma (**in** *monoid*) *nat-pow-1*: **assumes** *x*: $x \in carrier\ G$ **shows** $x\ (\hat{\ })_G\ (1::nat) = x$
  using *nat-pow-Suc* [*of x 0*] **and** *nat-pow-0* [*of x*] **and** *l-one* [*OF x*] **by** *simp*

If the element a is nilpotent, with $(a\ (\hat{\ })_R\ bound\ x)\ x = \mathbf{1}$, and *bound-funct* $x \leq m$, then $(a\ (\hat{\ })_R\ m)\ x = \mathbf{1}$

lemma (**in** *local-nilpotent-term*) *a-n-zero-a-m-zero*: **assumes** *bound-le-m*: $bound\text{-}funct\ x \leq m$
  shows $(a(\hat{\ })_R(m))\ x = \mathbf{1}_D$
proof (*cases* $x \in carrier\ D$)
  case *True*
  then have *x-in-D*: $x \in carrier\ D$ **by** *simp*
  show $(a\ (\hat{\ })_R\ m)\ x = \mathbf{1}$
    using *a-local-nilpot* **and** *bound-le-m* **proof** (*induct m*)
    case *0* **assume** *bound-le-zero*: $bound\text{-}funct\ x \leq 0$ **and** *alpha-n*: $\forall x \in carrier\ D.\ (a\ (\hat{\ })_R\ bound\text{-}funct\ x)\ x = \mathbf{1}$ **with** *x-in-D*
    show $(a\ (\hat{\ })_R\ (0::nat))\ x = \mathbf{1}_D$ **by** *auto*
  next
    case (*Suc m*)
    assume *hypo*: $[\![\forall x \in carrier\ D.\ (a\ (\hat{\ })_R\ bound\text{-}funct\ x)\ x = \mathbf{1};\ bound\text{-}funct\ x \leq m]\!] \Longrightarrow (a\ (\hat{\ })_R\ m)\ x = \mathbf{1}$
        and *a-bound-funct*: $\forall x \in carrier\ D.\ (a\ (\hat{\ })_R\ bound\text{-}funct\ x)\ x = \mathbf{1}$ **and** *bound-funct-le-Suc-m*: $bound\text{-}funct\ x \leq Suc\ m$
    then show $(a\ (\hat{\ })_R\ Suc\ m)\ x = \mathbf{1}$
    proof (*cases* $bound\text{-}funct\ x = Suc\ m$)
      case *True* **with** *a-bound-funct* **and** *x-in-D* **show** *?thesis* **by** *auto*

next
case *False* **with** *bound-funct-le-Suc-m* **have** *bound-funct* $x \leq m$ **by** *arith*
with *hypo* **and** *a-bound-funct* **have** *a-m*: $(a\ (\hat{\ })_R\ m)\ x = \mathbf{1}$ **by** *simp*
have $(a\ (\hat{\ })_R\ Suc\ m)\ x\ = (a \otimes_R (a\ (\hat{\ })_R\ m))\ x$
proof $-$
have $Suc\ m = (1::nat) + m$ **by** *arith*
with *sym* [*OF nat-pow-mult* [*OF a-in-R*, *of 1 m*]] **and** *nat-pow-1* [*OF a-in-R*]
show *?thesis* **by** (*simp only*: *expand-fun-eq*)
qed
also from *ring-R* **and** *x-in-D* **and** *a-m* **have** $\ldots = a\ \mathbf{1}$ **by** *simp*
also from *ring-R* **and** *hom-completion-one* [*of D D a*] **and** *D-diff-group* **and** *a-in-R* **have** $\ldots = \mathbf{1}$
unfolding *diff-group-def comm-group-def group-def* **by** *simp*
finally show $(a\ (\hat{\ })_R\ Suc\ m)\ x = \mathbf{1}$ **by** *simp*
qed
qed
next
case *False*
with *nat-pow-closed* [*OF a-in-R*, *of m*] **and** *ring-R* **and** *completion-closed2* [*of* $a\ (\hat{\ })_R\ m$ *D D x*]
show $(a\ (\hat{\ })_R\ m)\ x = \mathbf{1}$ **unfolding** *hom-completion-def completion-fun2-def* **by** *simp*
qed

The following definition is the power series of the local nilpotent endomorphism a in an element of its domain x; the power series is defined as the finite product in the differential group D of the powers $\lambda i.\ (a\ (\hat{\ })_R\ i)\ x$, up to *bound-funct x*

A different solution would be to consider the finite sum in the ring of endomorphisms R of terms *op* $(\hat{\ })_R\ a$ and then apply it to each element of the domain x

The first solution seems to me more coherent with the notion of "local nilpotency" we are dealing with, but both are identical

6.2 Definition of power series and some lemmas

context *local-nilpotent-term*
begin
definition *power-series* x == *finprod* D $(\lambda i::nat.\ (a(\hat{\ })_R\ i)\ x)$ $\{..bound\text{-}funct\ x\}$
end

Some results about the power series

lemma (**in** *local-nilpotent-term*) *power-pi*: $(op\ (\hat{\ })_R\ a) \in \{..(k::nat)\} \rightarrow carrier\ R$
using *nat-pow-closed* [*OF a-in-R*] **and** *Pi-def* [*of* $\{..k\}$ $(\lambda i::nat.\ carrier\ R)$] **by** *simp*

lemma (**in** *local-nilpotent-term*) *power-pi-D*: (λi::*nat*. ($a(\hat{})_R\ i$) x) $\in$ {..(k::*nat*)} $\rightarrow$ *carrier D*
proof (*unfold Pi-def*, *auto*, *cases* $x \in$ *carrier D*)
  fix i
  assume $i \leq k$
  from *nat-pow-closed* [*OF a-in-R*, *of i*] **have** *a-i-in-R*: ($a\ (\hat{})_R\ i$) $\in$ *carrier R* **by** *simp*
  case *True* **with** *a-i-in-R* **and** *ring-R* **and** *hom-completion-closed* [*of* ($a(\hat{})_R\ i$) *D D x*] **show** ($a\ (\hat{})_R\ i$) $x \in$ *carrier D* **by** *simp*
next
  fix i
  assume $i \leq k$
  from *nat-pow-closed* [*OF a-in-R*, *of i*] **have** *a-i-in-R*: ($a\ (\hat{})_R\ i$) $\in$ *carrier R* **by** *simp*
  case *False* **with** *a-i-in-R completion-closed2* [*of* ($a\ (\hat{})_R\ i$) *D D x*] **and** *ring-R* **show** ($a\ (\hat{})_R\ i$) $x \in$ *carrier D*
    unfolding *hom-completion-def completion-fun2-def* **by** *simp*
qed

As we already stated, $\lambda x.\ \bigotimes i \in \{..j\}.\ (a\ (\hat{})_R\ i)\ x$ is equal to *finsum R* (*op* $(\hat{})_R\ a$) {..j})

lemma (**in** *local-nilpotent-term*) *finprod-eq-finsum-bound-funct*:
  shows *finprod D* (λi::*nat*. ($a(\hat{})_R\ i$) x) {..*bound-funct x*} = ((*finsum R* (λi::*nat*. ($a(\hat{})_R\ i$)) {..*bound-funct x*}) x)
proof (*induct bound-funct x*)
  case *0*
  from *nat-pow-0* [*of a*] **and** *finsum-0* [*of op* $(\hat{})_R\ a$] **and** *power-pi* [*of 0*::*nat*] **and** *finprod-0* [*of* (λi::*nat*. ($a(\hat{})_R\ i$) x)]
    and *power-pi-D* [*of x 0*::*nat*]
  show ($\bigotimes i$::*nat*$\in${..*0*::*nat*}. ($a\ (\hat{})_R\ i$) x) = *finsum R* (*op* $(\hat{})_R\ a$) {..*0*::*nat*} x **by** *simp*
next
  case (*Suc n*)
  assume *hypo*: ($\bigotimes i \in$ {..n}. ($a\ (\hat{})_R\ i$) x) = *finsum R* (*op* $(\hat{})_R\ a$) {..n} x
  show ($\bigotimes i \in$ {..*Suc n*}. ($a\ (\hat{})_R\ i$) x) = *finsum R* (*op* $(\hat{})_R\ a$) {..*Suc n*} x
  proof (*cases* $x \in$ *carrier D*)
    case *True*
    from *finsum-Suc* [*OF power-pi*, *of n*] **have** *finsum R* (*op* $(\hat{})_R\ a$) {..*Suc n*} = $a\ (\hat{})_R$ *Suc n* $\oplus_R$ *finsum R* (*op* $(\hat{})_R\ a$) {..n} **by** *simp*
    with *ring-R* **and** *True* **have** *finsum R* (*op* $(\hat{})_R\ a$) {..*Suc n*} x = ($a\ (\hat{})_R$ *Suc n*) $x \otimes_D$ (*finsum R* (*op* $(\hat{})_R\ a$) {..n}) x **by** *simp*
    moreover
    from *True* **and** *finprod-Suc* [*OF power-pi-D*, *of x n*] **have** ($\bigotimes i \in$ {..*Suc n*}. ($a\ (\hat{})_R\ i$) x) = ($a\ (\hat{})_R$ *Suc n*) $x \otimes$ ($\bigotimes i \in$ {..n}. ($a\ (\hat{})_R\ i$) x)
      by *simp*
    with *hypo* **have** ($\bigotimes i \in$ {..*Suc n*}. ($a\ (\hat{})_R\ i$) x) = ($a\ (\hat{})_R$ *Suc n*) $x \otimes$ (*finsum R* (*op* $(\hat{})_R\ a$) {..n}) x **by** *simp*
    ultimately show *?thesis* **by** *simp*
  next

case *False*
from *finsum-closed* [*OF* - *power-pi*] **and** *finite-nat-iff-bounded* [*of* {..*Suc n*}]
have *finsum R* (*op* (^)$_R$ *a*) {..*Suc n*} ∈ *carrier R* **by** *simp*
with *ring-R* **and** *completion-closed2* [*of finsum R* (*op* (^)$_R$ *a*) {..*Suc n*} *D D*]
and *False*
have *finsum-one*: *finsum R* (*op* (^)$_R$ *a*) {..*Suc n*} *x* = **1** **unfolding** *hom-completion-def completion-fun2-def* **by** *simp*
moreover
have ∀ *i*∈{..*Suc n*}. (*a* (^)$_R$ *i*) *x* = **1**
proof
fix *i* **assume** *i-in-nat*: *i* ∈ {..*Suc n*}
from *nat-pow-closed* [*OF a-in-R*, *of i*] **and** *ring-R* **and** *completion-closed2* [*of* - *D D*] **and** *False* **show** (*a* (^)$_R$ *i*) *x* = **1**
unfolding *hom-completion-def completion-fun2-def* **by** *simp*
qed
with *finprod-cong* [*of* {..*Suc n*} {..*Suc n*} (λ*i*. (*a* (^)$_R$ *i*) *x*) λ*x*. **1**] **and** *power-pi-D* [*of x Suc n*]
have (⨂ *i*∈{..*Suc n*}. (*a* (^)$_R$ *i*) *x*) = (⨂ *i*∈{..*Suc n*}. **1**) **by** *simp*
with *finprod-one* [*of* {..*Suc n*}] **and** *finite-nat-iff-bounded* [*of* {..*Suc n*}] **have** *finprod-one*: (⨂ *i*∈{..*Suc n*}. (*a* (^)$_R$ *i*) *x*) = **1**
by *simp*
ultimately show *?thesis* **by** *simp*
qed
qed

lemma (**in** *local-nilpotent-term*) *power-series-closed*: **shows** (⨂ *i*∈{..*m*::*nat*}. (*a* (^)$_R$ *i*) *x*) ∈ *carrier D*
proof (*rule finprod-closed*)
from *finite-nat-iff-bounded* **show** *finite* {..*m*} **by** *auto*
from *power-pi-D* [*of x m*] **show** (λ*i*::*nat*. (*a* (^)$_R$ *i*) *x*) ∈ {..*m*} → *carrier D* **by** *simp*
qed

The following result is equal to the previous one but for the case of definition of the power series

lemma (**in** *local-nilpotent-term*) *power-series-closed2*: (⨂ *i*∈{..*bound-funct x*}. (*a* (^)$_R$ *i*) *x*) ∈ *carrier D*
proof (*rule finprod-closed*)
from *finite-nat-iff-bounded* **show** *finite* {..*bound-funct x*} **by** *auto*
from *power-pi-D* [*of x bound-funct x*] **and** *Pi-def* [*of* {..*bound-funct x*} (λ*i*::*nat*. *carrier R*)]
show (λ*i*::*nat*. (*a* (^)$_R$ *i*) *x*) ∈ {..*bound-funct x*} → *carrier D* **by** *simp*
qed

lemma (**in** *local-nilpotent-term*) *power-series-extended*: **assumes** *bf-le-m*: *bound-funct x* ≤ *m*
shows *power-series x* = *finprod D* (λ*i*::*nat*. (*a* (^)$_R$ *i*) *x*) {..*m*}
using *bf-le-m* **proof** (*induct m*)
case *0*

from *this* **and** *power-series-def* [*of x*] **show** *power-series x* = ($\bigotimes$ *i*∈{..(*0*::*nat*)}. (*a* (ˆ)$_R$ *i*) *x*) **by** *simp*
next
case (*Suc m*)
assume *hypo*: *bound-funct x* ≤ *m* ⟹ *power-series x* = ($\bigotimes$ *i*∈{..*m*}. (*a* (ˆ)$_R$ *i*) *x*) **and** *bf-le-Suc-m*: *bound-funct x* ≤ *Suc m*
show *power-series x* = ($\bigotimes$ *i*∈{..*Suc m*}. (*a* (ˆ)$_R$ *i*) *x*)
proof (*cases bound-funct x* = *Suc m*)
case *True* **thus** *?thesis* **unfolding** *power-series-def* **by** *simp*
next
case *False* **with** *bf-le-Suc-m* **and** *hypo* **have** *hypo-m*: *power-series x* = ($\bigotimes$ *i*∈{..*m*}. (*a* (ˆ)$_R$ *i*) *x*) **by** *arith*
from *a-n-zero-a-m-zero* [*OF bf-le-Suc-m*] **have** *a-Suc-m-one*: (*a* (ˆ)$_R$ *Suc m*) *x* = **1**.
from *power-pi-D* [*of x Suc m*] **and** *finprod-Suc* [*of* (λ*i*::*nat*. (*a* (ˆ)$_R$ *i*) *x*) *m*]
have ($\bigotimes$ *i*∈{..*Suc m*}. (*a* (ˆ)$_R$ *i*) *x*) = (*a* (ˆ)$_R$ *Suc m*) *x* ⊗ ($\bigotimes$ *i*∈{..*m*}. (*a* (ˆ)$_R$ *i*) *x*) **by** *simp*
also from *a-Suc-m-one* **have** ... = **1** ⊗ ($\bigotimes$ *i*∈{..*m*}. (*a* (ˆ)$_R$ *i*) *x*) **by** *simp*
also from *hypo-m* **and** *D.l-one* [*OF power-series-closed* [*of x m*]] **have** ... = *power-series x* **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed

The power series is itself an endomorphism of the differential group

lemma (**in** *local-nilpotent-term*) *power-series-in-R*: **shows** *power-series* ∈ *carrier R*
proof −
have *power-series* ∈ *hom-completion D D*
proof (*unfold hom-completion-def hom-def Pi-def*, *auto*)
fix *x*
assume *x-in-D*: *x* ∈ *carrier D*
from *power-series-closed* [*of x bound-funct x*] **and** *power-series-def* [*of x*] **show** *power-series x* ∈ *carrier D* **by** *simp*
next
fix *x y*
assume *x-in-D*: *x* ∈ *carrier D* **and** *y-in-D*: *y* ∈ *carrier D*
show *power-series* (*x* ⊗ *y*) = *power-series x* ⊗ *power-series y*
proof −
let *?max* = *max* (*bound-funct* (*x* ⊗ *y*)) (*max* (*bound-funct x*) (*bound-funct y*))
from *power-series-extended* [*of x* ⊗ *y ?max*] **have** *p-s-ex-xy*: *power-series* (*x* ⊗ *y*) = ($\bigotimes$ *i*∈{..*?max*}. (*a* (ˆ)$_R$ *i*) (*x* ⊗ *y*)) **by** *arith*
from *power-series-extended* [*of x ?max*] **have** *p-s-ex-x*: *power-series x* = ($\bigotimes$ *i*∈{..*?max*}. (*a* (ˆ)$_R$ *i*) *x*) **by** *arith*
from *power-series-extended* [*of y ?max*] **have** *p-s-ex-y*: *power-series y* = ($\bigotimes$ *i*∈{..*?max*}. (*a* (ˆ)$_R$ *i*) *y*) **by** *arith*
from *p-s-ex-x* **and** *p-s-ex-y*
have *power-series x* ⊗ *power-series y* = ($\bigotimes$ *i*∈{..*?max*}. (*a* (ˆ)$_R$ *i*) *x*) ⊗

($\bigotimes$ $i \in \{..?max\}$. (a (^)$_R$ i) y) **by** *simp*
also from *sym* [*OF finprod-mult* [*of* (λi::*nat*. (a (^)$_R$ i) x) *?max* (λi::*nat*. (a (^)$_R$ i) y)]]
and *power-pi-D* [*of x ?max*] *power-pi-D* [*of y ?max*] **and** *finite-nat-iff-bounded* [*of* {..*?max*}]
have ... = ($\bigotimes$ $i \in \{..?max\}$. (a (^)$_R$ i) $x \otimes$ (a (^)$_R$ i) y) **by** *simp*
also from *nat-pow-closed* [*OF a-in-R*] **and** *hom-completion-mult* [*OF - x-in-D y-in-D*, *of - D*] **and** *ring-R*
have ... = ($\bigotimes$ $i \in \{..?max\}$. (a (^)$_R$ i) ($x \otimes y$)) **by** *simp*
also from *sym* [*OF p-s-ex-xy*] **have** ... = *power-series* ($x \otimes y$) **by** *simp*
finally show *?thesis* **by** *simp*
qed
next
show *power-series* $\in$ *completion-fun2 D D*
proof (*unfold completion-fun2-def completion-def expand-fun-eq*, *simp*, *intro exI* [*of - power-series*], *auto*)
fix *x*
assume $x \notin$ *carrier D*
with *nat-pow-closed* [*OF a-in-R*] **and** *completion-closed2* [*of - D D x*] **and** *ring-R* **have** (λi::*nat*. (a (^)$_R$ i) x) = (λi::*nat*. **1**)
by (*unfold hom-completion-def expand-fun-eq*, *auto*)
with *finprod-one* [*of* {..*bound-funct x*}] **and** *power-series-def* [*of x*]
show *power-series x* = **1** **by** *auto*
qed
qed
with *ring-R* **show** *power-series* $\in$ *carrier R* **by** *simp*
qed

6.3 Some basic operations over finite series

Right distributivity of the product

lemma (**in** *ring*) *finsum-dist-r*: **assumes** *a-in-R*: $a \in$ *carrier R* **and** *b-in-R*: $b \in$ *carrier R*
shows $b \otimes$ *finsum R* (*op* (^) a) {..(*m*::*nat*)} = ($\bigoplus$ $i \in$ {..(*m*::*nat*)}. $b \otimes a$ (^) i)
proof (*induct m*)
case *0*
from *finsum-0* [*of* λi::*nat*. a(^)i] **and** *finsum-0* [*of* λi::*nat*. $b \otimes a$(^)i] **and** *b-in-R*
show $b \otimes$ *finsum R* (*op* (^) a) {..(*0*::*nat*)} = ($\bigoplus$ $i \in$ {..(*0*::*nat*)}. $b \otimes a$ (^) i) **by** *simp*
next
case (*Suc m*)
assume *hypo*: $b \otimes$ *finsum R* (*op* (^) a) {..*m*} = ($\bigoplus$ $i \in$ {..*m*}. $b \otimes a$ (^) i)
show $b \otimes$ *finsum R* (*op* (^) a) {..*Suc m*} = ($\bigoplus$ $i \in$ {..*Suc m*}. $b \otimes a$ (^) i)
proof −
from *finsum-Suc* [*of* (*op* (^) a) *m*] **and** *Pi-def* [*of* {..*Suc m*} λi::*nat*. *carrier R*] *nat-pow-closed* [*OF a-in-R*]
have $b \otimes$ ($\bigoplus$ $i \in$ {..*Suc m*}. a (^) i) = $b \otimes$ (a(^)(*Suc m*) $\oplus$ ($\bigoplus$ $i \in$ {..*m*}. a (^) i)) **by** *simp*

also from *r-distr* [*OF nat-pow-closed* [*OF a-in-R*, *of Suc m*] - *b-in-R*, *of finsum R* (*op* (^) *a*) {..*m*}]
b-in-R nat-pow-closed [*OF a-in-R*, *of Suc m*] *finsum-closed* [*of* {..*m*} *op* (^) *a*]
and *Pi-def* [*of* {..*m*} λ*i*::*nat*. *carrier R*] *nat-pow-closed* [*OF a-in-R*] **and** *finite-nat-iff-bounded* [*of* {..*m*}]
have ... = $b \otimes a(\hat{})(Suc\ m) \oplus b \otimes (\bigoplus i \in \{..m\}.\ a\ (\hat{})\ i)$ **by** *simp*
also from *hypo* **have** ... = $b \otimes a(\hat{})(Suc\ m) \oplus (\bigoplus i \in \{..m\}.\ b \otimes a\ (\hat{})\ i)$ **by** *simp*
also from *finsum-Suc* [*of* (λ*i*::*nat*. $b \otimes a$ (^) *i*), *of m*] **and** *Pi-def* [*of* {..*Suc m*} λ*i*::*nat*. *carrier R*]
nat-pow-closed [*OF a-in-R*] **and** *b-in-R*
have ... = $(\bigoplus i \in \{..Suc\ m\}.\ b \otimes a\ (\hat{})\ i)$ **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed

lemma (**in** *local-nilpotent-term*) *b-power-pi-D*: **assumes** *b-in-R*: $b \in carrier\ R$
shows $(\lambda i.\ b\ ((a\ (\hat{})_R\ i)\ x)) \in \{..(k::nat)\} \rightarrow carrier\ D$
using *power-pi-D* [*of x k*] **and** *ring-R* **and** *b-in-R* **unfolding** *hom-completion-def completion-fun2-def completion-def hom-def Pi-def* **by** *auto*

lemma (**in** *local-nilpotent-term*) *nat-pow-closed-D*: **shows** $(a\ (\hat{})_R\ (m::nat))\ x \in carrier\ D$
using *ring-R* **and** *nat-pow-closed* [*OF a-in-R*, *of m*] **unfolding** *hom-completion-def completion-fun2-def completion-def hom-def Pi-def* **by** *auto*

Left distributivity of the product of a finite sum

lemma (**in** *local-nilpotent-term*) *power-series-dist-l*: **assumes** *b-in-R*: $b \in carrier\ R$
shows $b\ (\bigotimes i \in \{..(m::nat)\}.\ (a\ (\hat{})_R\ i)\ x) = (\bigotimes i \in \{..(m::nat)\}.\ (b\ ((a\ (\hat{})_R\ i)\ x)))$
proof (*induct m*)
case *0*
from *finprod-0* [*of* (λ*i*. $(a\ (\hat{})_R\ i)\ x$)] **and** *power-pi-D* [*of x 0*::*nat*] **and** *finprod-0* [*of* (λ*i*. $b\ ((a\ (\hat{})_R\ i)\ x)$)]
and *b-power-pi-D* [*OF b-in-R*, *of x 0*::*nat*] **show** $b\ (\bigotimes i \in \{..0::nat\}.\ (a\ (\hat{})_R\ i)\ x) = (\bigotimes i \in \{..0::nat\}.\ b\ ((a\ (\hat{})_R\ i)\ x))$ **by** *simp*
next
case (*Suc m*)
assume *hypo*: $b\ (\bigotimes i \in \{..m\}.\ (a\ (\hat{})_R\ i)\ x) = (\bigotimes i \in \{..m\}.\ b\ ((a\ (\hat{})_R\ i)\ x))$
show $b\ (\bigotimes i \in \{..Suc\ m\}.\ (a\ (\hat{})_R\ i)\ x) = (\bigotimes i \in \{..Suc\ m\}.\ b\ ((a\ (\hat{})_R\ i)\ x))$
proof −
from *finprod-Suc* [*OF power-pi-D* [*of x*], *of m*] **have** $b\ (\bigotimes i \in \{..Suc\ m\}.\ (a\ (\hat{})_R\ i)\ x) = b\ ((a\ (\hat{})_R\ Suc\ m)\ x \otimes (\bigotimes i \in \{..m\}.\ (a\ (\hat{})_R\ i)\ x))$
by *simp*
also from *b-in-R* **and** *ring-R* **and** *hom-completion-mult* [*of b D D* $(a\ (\hat{})_R\ Suc\ m)\ x$ $(\bigotimes i \in \{..m\}.\ (a\ (\hat{})_R\ i)\ x)$]
and *finprod-closed* [*of* {..*m*} (λ*i*. $(a\ (\hat{})_R\ i)\ x$)] *power-pi-D* [*of x m*] *nat-pow-closed-D*

[*of Suc m x*]
  have … = *b* ((*a* (ˆ)$_R$ *Suc m*) *x*) ⊗ *b* ((⨂ *i*∈{..*m*}. (*a* (ˆ)$_R$ *i*) *x*)) **by** *simp*
  also from *hypo* **have** … = *b* ((*a* (ˆ)$_R$ *Suc m*) *x*) ⊗ (⨂ *i*∈{..*m*}. *b* ((*a* (ˆ)$_R$ *i*) *x*)) **by** *simp*
  also from *sym* [*OF finprod-Suc* [*OF b-power-pi-D* [*OF b-in-R*, *of x*], *of m*]] **have** … = (⨂ *i*∈{..*Suc m*}. *b* ((*a* (ˆ)$_R$ *i*) *x*)) **by** *simp*
  finally show *?thesis* **by** *simp*
 qed
qed

lemma (**in** *local-nilpotent-term*) *power-pi-b-D*: **assumes** *b-in-R*: *b* ∈ *carrier R* **shows** (λ*i*. (*a* (ˆ)$_R$ *i*) (*b x*)) ∈ {..(*k*::*nat*)} → *carrier D*
 using *power-pi-D* [*of b x k*] **and** *ring-R* **and** *b-in-R* **by** *simp*

lemma (**in** *local-nilpotent-term*) *power-series-dist-r*: **assumes** *b-in-R*: *b* ∈ *carrier R*
 shows (λ*x*. (⨂ *i*∈{..*m*}. (*a* (ˆ)$_R$ *i*) *x*)) (*b x*) = (⨂ *i*∈{..(*m*::*nat*)}. ((*a* (ˆ)$_R$ *i*) (*b x*))) **by** *simp*

The following lemma showed to be useful in some situations

lemma (**in** *comm-monoid*) *finprod-singleton* [*simp*]:
 f ∈ {*i*::*nat*} −> *carrier G* ==> *finprod G f* {*i*} = *f i* **by** (*simp add*: *Pi-def*)

Finite series can be decomposed in the product of its first element and the remaining part

lemma (**in** *local-nilpotent-term*) *power-series-first-element*:
 shows *finprod D* (λ*i*::*nat*. (*a* (ˆ)$_R$ *i*) *x*) {..(*i*::*nat*)} = (*a*(ˆ)$_R$ (*0*::*nat*)) *x* ⊗ *finprod D* (λ*i*::*nat*. (*a* (ˆ)$_R$ *i*) *x*) {*1*..(*i*::*nat*)}
proof (*induct i*)
 case *0*
  from *ring-R* **have** *one-x*: $\mathbf{1}_R$ *x* ∈ *carrier D* **unfolding** *hom-completion-def completion-fun2-def completion-def* **by** *auto*
  from *finprod-0* [*of* (λ*i*. (*a* (ˆ)$_R$ *i*) *x*)] **and** *power-pi-D* [*of x 0*] **and** *D.r-one* [*OF one-x*]
 show (⨂ *i*∈{..(*0*::*nat*)}. (*a* (ˆ)$_R$ *i*) *x*) = (*a* (ˆ)$_R$ (*0*::*nat*)) *x* ⊗ (⨂ *i*∈{(*1*::*nat*)..*0*}. (*a* (ˆ)$_R$ *i*) *x*) **by** *simp*
next
 case (*Suc i*)
 assume *hypo*: (⨂ *i*∈{..*i*}. (*a* (ˆ)$_R$ *i*) *x*) = (*a* (ˆ)$_R$ (*0*::*nat*)) *x* ⊗ (⨂ *i*∈{*1*..*i*}. (*a* (ˆ)$_R$ *i*) *x*)
 show (⨂ *i*∈{..*Suc i*}. (*a* (ˆ)$_R$ *i*) *x*) = (*a* (ˆ)$_R$ (*0*::*nat*)) *x* ⊗ (⨂ *i*∈{*1*..*Suc i*}. (*a* (ˆ)$_R$ *i*) *x*)
 proof −
  from *finprod-Suc* [*of* (λ*i*::*nat*. (*a* (ˆ)$_R$ *i*) *x*) *i*] **and** *power-pi-D* [*of x Suc i*]
  have (⨂ *i*∈{..*Suc i*}. (*a* (ˆ)$_R$ *i*) *x*) = (*a* (ˆ)$_R$ *Suc i*) *x* ⊗ (⨂ *i*∈{..*i*}. (*a* (ˆ)$_R$ *i*) *x*) **by** *simp*
   also from *hypo* **have** … = (*a* (ˆ)$_R$ *Suc i*) *x* ⊗ ((*a* (ˆ)$_R$ (*0*::*nat*)) *x* ⊗ (⨂ *i*∈{*1*..*i*}. (*a* (ˆ)$_R$ *i*) *x*)) **by** *simp*

also have ... = $((a\ (\hat{\ })_R\ \mathit{Suc}\ i)\ x \otimes (a\ (\hat{\ })_R\ (0::nat))\ x) \otimes (\bigotimes i{\in}\{1..i\}.\ (a\ (\hat{\ })_R\ i)\ x)$
proof (*rule sym* [*OF D.m-assoc* [*of* $(a\ (\hat{\ })_R\ \mathit{Suc}\ i)\ x\ (a\ (\hat{\ })_R\ (0::nat))\ x\ (\bigotimes i{\in}\{1..i\}.\ (a\ (\hat{\ })_R\ i)\ x)$]]])
from *nat-pow-closed-D* [*of Suc i x*] **show** $(a\ (\hat{\ })_R\ \mathit{Suc}\ i)\ x \in \mathit{carrier}\ D$ **by** *simp*
from *nat-pow-closed-D* [*of 0::nat x*] **show** $(a\ (\hat{\ })_R\ (0::nat))\ x \in \mathit{carrier}\ D$ **by** *simp*
from *finprod-closed* [*of* $\{1..i\}\ (\lambda i::nat.\ (a(\hat{\ })_R i)\ x)$] **and** *finite-nat-iff-bounded* [*of* $\{1..i\}$]
and *nat-pow-closed-D* [*of - x*] **and** *Pi-def* [*of* $\{1..i\}\ \lambda i::nat.\ \mathit{carrier}\ D$]
show $(\bigotimes i{\in}\{1..i\}.\ (a\ (\hat{\ })_R\ i)\ x) \in \mathit{carrier}\ D$ **by** *simp*
qed
also from *m-comm* [*OF nat-pow-closed-D* [*of Suc i x*] *nat-pow-closed-D* [*of 0::nat x*]]
have ... = $((a\ (\hat{\ })_R\ (0::nat))\ x \otimes (a\ (\hat{\ })_R\ \mathit{Suc}\ i)\ x) \otimes (\bigotimes i{\in}\{1..i\}.\ (a\ (\hat{\ })_R\ i)\ x)$ **by** *simp*
also have ... = $(a\ (\hat{\ })_R\ (0::nat))\ x \otimes ((a\ (\hat{\ })_R\ \mathit{Suc}\ i)\ x \otimes (\bigotimes i{\in}\{1..i\}.\ (a\ (\hat{\ })_R\ i)\ x))$
proof (*rule D.m-assoc* [*of* $(a\ (\hat{\ })_R\ (0::nat))\ x\ (a\ (\hat{\ })_R\ (\mathit{Suc}\ i))\ x\ (\bigotimes i{\in}\{1..i\}.\ (a\ (\hat{\ })_R\ i)\ x)$]])
from *nat-pow-closed-D* [*of Suc i x*] **show** $(a\ (\hat{\ })_R\ \mathit{Suc}\ i)\ x \in \mathit{carrier}\ D$ **by** *simp*
from *nat-pow-closed-D* [*of 0::nat x*] **show** $(a\ (\hat{\ })_R\ (0::nat))\ x \in \mathit{carrier}\ D$ **by** *simp*
from *finprod-closed* [*of* $\{1..i\}\ (\lambda i::nat.\ (a(\hat{\ })_R i)\ x)$] **and** *finite-nat-iff-bounded* [*of* $\{1..i\}$]
and *nat-pow-closed-D* [*of - x*] **and** *Pi-def* [*of* $\{1..i\}\ \lambda i::nat.\ \mathit{carrier}\ D$]
show $(\bigotimes i{\in}\{1..i\}.\ (a\ (\hat{\ })_R\ i)\ x) \in \mathit{carrier}\ D$ **by** *simp*
qed
also from *nat-pow-closed-D* [*of Suc i x*]
have ... = $(a\ (\hat{\ })_R\ (0::nat))\ x \otimes ((\bigotimes i{\in}\{\mathit{Suc}\ i\}.\ (a\ (\hat{\ })_R\ i)\ x) \otimes (\bigotimes i{\in}\{1..i\}.\ (a\ (\hat{\ })_R\ i)\ x))$ **by** *simp*
also from *sym* [*OF finprod-Un-disjoint* [*of* $\{\mathit{Suc}\ i\}\ \{1..i\}\ (\lambda i::nat.\ (a\ (\hat{\ })_R\ i)\ x)$]]
and *finite-nat-iff-bounded* [*of* $\{1..i\}$] *finite-nat-iff-bounded* [*of* $\{\mathit{Suc}\ i\}$] **and** *Pi-def* [*of* $\{(1::nat)..i\}\ (\lambda i::nat.\ \mathit{carrier}\ D)$]
Pi-def [*of* $\{\mathit{Suc}\ i\}\ (\lambda i::nat.\ \mathit{carrier}\ D)$] **and** *nat-pow-closed-D* [*of Suc i x*] *nat-pow-closed-D* [*of - x*]
have $(a\ (\hat{\ })_R\ (0::nat))\ x \otimes ((\bigotimes i{\in}\{\mathit{Suc}\ i\}.\ (a\ (\hat{\ })_R\ i)\ x) \otimes (\bigotimes i{\in}\{1..i\}.\ (a\ (\hat{\ })_R\ i)\ x))$
$= (a\ (\hat{\ })_R\ (0::nat))\ x \otimes (\bigotimes i{\in}\{1..i\} \cup \{\mathit{Suc}\ i\}.\ (a\ (\hat{\ })_R\ i)\ x)$ **by** *simp*
also have ... = $(a\ (\hat{\ })_R\ (0::nat))\ x \otimes (\bigotimes i{\in}\{1..\mathit{Suc}\ i\}.\ (a\ (\hat{\ })_R\ i)\ x)$
proof –
have $\{1..i\} \cup \{\mathit{Suc}\ i\} = \{1..\mathit{Suc}\ i\}$ **by** *auto*
then show *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed

qed

Finite series which start in index one can be seen as the product of the generic term and the finite series in index zero

lemma (**in** *local-nilpotent-term*) *power-series-factor*: **shows** ($\bigotimes j \in \{(1::nat)..Suc\ i\}$. $(a\ (\hat{\ })_R\ j)\ x) = a\ (\bigotimes j \in \{..i\}$. $(a\ (\hat{\ })_R\ j)\ x)$
proof (*induct i*)
  case *0*
  have $a\ x = a\ (\mathbf{1}_R\ x)$
  proof (*cases* $x \in$ *carrier D*)
    case *True* **with** *ring-R* **show** $a\ x = a\ (\mathbf{1}_R\ x)$ **by** *simp*
  next
    case *False* **with** *completion-closed2* [*of a D D x*] *hom-completion-one* [*of D D a*] **and** *D-diff-group* **and** *a-in-R* **and** *ring-R*
      show $a\ x = a\ (\mathbf{1}_R\ x)$ **unfolding** *diff-group-def comm-group-def group-def hom-completion-def completion-fun2-def* **by** *simp*
  qed
  with *finprod-0* [*of* $(\lambda j::nat.\ (a\ (\hat{\ })_R\ j)\ x)$] *finprod-insert* [*of* {} *Suc 0* $(\lambda j::nat.\ (a\ (\hat{\ })_R\ j)\ x)$]
      and *nat-pow-closed-D* [*of Suc 0 x*] **and** *nat-pow-closed-D* [*of 0 x*] *a-in-R finprod-singleton* [*of* $(\lambda j::nat.\ (a\ (\hat{\ })_R\ j)\ x)$ *Suc 0*]
  show $(\bigotimes j \in \{1..Suc\ 0\}.\ (a\ (\hat{\ })_R\ j)\ x) = a\ (\bigotimes j \in \{..0::nat\}.\ (a\ (\hat{\ })_R\ j)\ x)$ **by** *simp*
next
  case (*Suc i*)
  assume *hypo*: $(\bigotimes j \in \{1..Suc\ i\}.\ (a\ (\hat{\ })_R\ j)\ x) = a\ (\bigotimes j \in \{..i\}.\ (a\ (\hat{\ })_R\ j)\ x)$
  show $(\bigotimes j \in \{1..Suc\ (Suc\ i)\}.\ (a\ (\hat{\ })_R\ j)\ x) = a\ (\bigotimes j \in \{..Suc\ i\}.\ (a\ (\hat{\ })_R\ j)\ x)$
  proof −
    have $(\bigotimes j \in \{1..Suc\ (Suc\ i)\}.\ (a\ (\hat{\ })_R\ j)\ x) = (\bigotimes j \in \{1..Suc\ i\} \cup \{Suc\ (Suc\ i)\}.\ (a\ (\hat{\ })_R\ j)\ x)$
    proof −
      have $\{1..Suc\ i\} \cup \{Suc\ (Suc\ i)\} = \{1..Suc\ (Suc\ i)\}$ **by** *auto*
      thus *?thesis* **by** *simp*
    qed
    also from *finprod-Un-disjoint* [*of* $\{1..Suc\ i\}$ $\{Suc\ (Suc\ i)\}$ $(\lambda j::nat.\ (a\ (\hat{\ })_R\ j)\ x)$]
      and *finite-nat-iff-bounded* [*of* $\{1..Suc\ i\}$] *finite-nat-iff-bounded* [*of* $\{Suc\ (Suc\ i)\}$] **and**
      Pi-def [*of* $\{(1::nat)..Suc\ i\}$ $(\lambda j::nat.\ carrier\ D)$]
      Pi-def [*of* $\{Suc\ (Suc\ i)\}$ $(\lambda j::nat.\ carrier\ D)$] **and** *nat-pow-closed-D* [*of Suc (Suc i) x*] *nat-pow-closed-D* [*of - x*]
    have $\ldots = (\bigotimes j \in \{1..Suc\ i\}.\ (a\ (\hat{\ })_R\ j)\ x) \otimes (\bigotimes j \in \{Suc\ (Suc\ i)\}.\ (a\ (\hat{\ })_R\ j)\ x)$ **by** *simp*
    also from *hypo* **and** *finprod-singleton* [*of* $(\lambda j::nat.\ (a\ (\hat{\ })_R\ j)\ x)$ *Suc (Suc i)*] **and** *nat-pow-closed-D* [*of Suc (Suc i) x*]
    have $\ldots = a\ (\bigotimes j \in \{..i\}.\ (a\ (\hat{\ })_R\ j)\ x) \otimes ((a\ (\hat{\ })_R\ Suc\ (Suc\ i))\ x)$ **by** *simp*
    also from *ring-R* **have** $\ldots = a\ (\bigotimes j \in \{..i\}.\ (a\ (\hat{\ })_R\ j)\ x) \otimes a\ ((a\ (\hat{\ })_R\ Suc\ i)\ x)$
    proof −

have *1* + *Suc i* = *Suc* (*Suc i*) **by** *arith*
with *sym* [*OF nat-pow-mult* [*OF a-in-R*, *of 1 Suc i*]] **and** *nat-pow-1* [*OF a-in-R*]
have *a* (^)$_R$ *Suc* (*Suc i*) = *a* ⊗$_R$ (*a* (^)$_R$ *Suc i*) **by** *simp*
with *ring-R* **have** ((*a* (^)$_R$ *Suc* (*Suc i*)) *x*) = *a* ((*a* (^)$_R$ *Suc i*) *x*) **by** *simp*
then show *?thesis* **by** *simp*
qed
also have ... = *a* ((⨂*j*∈{..*i*}. (*a* (^)$_R$ *j*) *x*) ⊗ (*a* (^)$_R$ *Suc i*) *x*)
proof (*intro sym* [*OF hom-completion-mult* [*of a D D* ⨂*j*∈{..*i*}. (*a* (^)$_R$ *j*) *x* (*a* (^)$_R$ *Suc i*) *x*]])
from *ring-R* **and** *a-in-R* **show** *a* ∈ *hom-completion D D* **by** *simp*
from *finprod-closed* [*of* {..*i*} (λ*j*::*nat*. (*a* (^)$_R$ *j*) *x*)] **and** *power-pi-D* [*of x i*]
show (⨂*j*∈{..*i*}. (*a* (^)$_R$ *j*) *x*) ∈ *carrier D* **by** *simp*
from *nat-pow-closed-D* [*of Suc i x*] **show** (*a* (^)$_R$ *Suc i*) *x* ∈ *carrier D* **by** *simp*
qed
also from *nat-pow-closed-D* [*of Suc i x*] **have** ... = *a* ((⨂*j*∈{..*i*}. (*a* (^)$_R$ *j*) *x*) ⊗ (⨂*j*∈{*Suc i*}.(*a* (^)$_R$ *Suc i*) *x*))
by *simp*
also from *sym* [*OF finprod-Un-disjoint* [*of* {..*i*} {*Suc i*} (λ*j*::*nat*. (*a* (^)$_R$ *j*) *x*)]]
and *finite-nat-iff-bounded* [*of* {..*i*}] *finite-nat-iff-bounded* [*of* {*Suc i*}] **and** *Pi-def* [*of* {..*i*} (λ*j*::*nat*. *carrier D*)]
Pi-def [*of* {*Suc i*} (λ*j*::*nat*. *carrier D*)] **and** *nat-pow-closed-D* [*of Suc i x*] *nat-pow-closed-D* [*of - x*]
have ... = *a* (⨂*j*∈{..*i*} ∪ {*Suc i*}. (*a* (^)$_R$ *j*) *x*) **by** *simp*
also have ... = *a* (⨂*j*∈{..*Suc i*}. (*a* (^)$_R$ *j*) *x*)
proof −
have {..*i*} ∪ {*Suc i*} = {..*Suc i*} **by** *auto*
thus *?thesis* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed
qed

If we were able to interpret locales, now the idea would be to interpret the *locale nilpotent-term* with local nilpotent term α, as later defined in locale *alpha-beta*

6.4 Definition and some lemmas of perturbations

Perturbations are a homomorphism of D (not a differential homomorphism!) such that its addition with the differential is again a differential

constdefs (**structure** *D*)
pert :: - => ('*a* => '*a*) *set*
pert D == {δ. δ ∈ *hom-completion D D* &
diff-group (| *carrier* = *carrier D*, *mult* = *mult D*, *one* = *one D*, *diff* = (λ*x*. *if x* ∈ *carrier D then* ((*differ*) *x*) ⊗ (δ *x*) *else* **1**)|) }

locale *diff-group-pert* = *diff-group D* + *var* δ +
assumes *delta-pert*: δ ∈ *pert D*

lemma (**in** *diff-group-pert*) *diff-group-pert-is-diff-group*:
shows *diff-group* ⦇*carrier* = *carrier D*, *mult* = *mult D*, *one* = *one D*, *diff* = (λ*x*. *if x* ∈ *carrier D then* ((*differ*$_D$) *x*) ⊗$_D$ (δ *x*) *else* $\mathbf{1}_D$)⦈
using *diff-group-pert.delta-pert* [*of D* δ] **and** *prems* **unfolding** *diff-group-pert-def pert-def* **by** *simp*

lemma (**in** *diff-group-pert*) *pert-is-hom*: **shows** δ ∈ *hom-completion D D*
using *diff-group-pert.delta-pert* [*of D* δ] **and** *prems* **unfolding** *diff-group-pert-def pert-def* **by** *simp*

lemma (**in** *ring-endomorphisms*) *diff-group-pert-is-diff-group*: **assumes** *delta*: δ ∈ *pert D*
shows *diff-group* ⦇ *carrier* = *carrier D*, *mult* = *mult D*, *one* = *one D*, *diff* = (*differ*$_D$) ⊕$_R$ δ ⦈
using *diff-group-pert.diff-group-pert-is-diff-group* [*of D* δ] **and** *ring-R* **and** *prems*
unfolding *diff-group-pert-def diff-group-pert-axioms-def ring-endomorphisms-def*
by *simp*

lemma (**in** *ring-endomorphisms*) *pert-in-R* [*simp*]: **assumes** *delta*: δ ∈ *pert D*
shows δ ∈ *carrier R*
using *ring-R* **and** *diff-group-pert.pert-is-hom* [*of D* δ] **and** *prems*
unfolding *diff-group-pert-def diff-group-pert-axioms-def ring-endomorphisms-def*
by *simp*

lemma (**in** *ring-endomorphisms*) *diff-pert-in-R* [*simp*]: **assumes** *delta*: δ ∈ *pert D*
shows (*differ*$_D$) ⊕$_R$ δ ∈ *carrier R*
using *delta* **by** *simp*

The reason to introduce α by means of a *defines* command is to get the expected behavior when merging this locale with locale *local-nilpotent-term D R* α *bound-phi* in the definition of locale *local-nilpotent-alpha*

locale *alpha-beta* = *ring-endomorphisms* + *reduction* + *var* δ + *var* α +
assumes *delta-pert*: δ ∈ *pert D*
defines *alpha-def*: α == ⊖$_R$ (δ ⊗$_R$ *h*)

context *alpha-beta*
begin

definition *beta-def*: β = ⊖$_R$ (*h* ⊗$_R$ δ)

end

locale *local-nilpotent-alpha* = *alpha-beta* + *local-nilpotent-term D R* α *bound-phi*

The definition of Φ corresponds with the one given in the Basic Perturbation Lemma, Lemma 2.3.1 in Aransay's memoir

context *local-nilpotent-alpha*
begin

definition *phi-def*: Φ == *local-nilpotent-term.power-series D R α bound-phi*

end

lemma (**in** *alpha-beta*) *pert-in-R* [*simp*]: **shows** δ ∈ *carrier R*
  using *delta-pert* **and** *ring-R* **by** (*unfold pert-def*, *simp*)

lemma (**in** *alpha-beta*) *h-in-R* [*simp*]: **shows** *h* ∈ *carrier R*
  using *h-hom-compl* **and** *ring-R* **by** *simp*

lemma (**in** *alpha-beta*) *alpha-in-R*: **shows** α ∈ *carrier R*
  using *alpha-def* **and** *pert-in-R* **and** *h-in-R* **by** *simp*

lemma (**in** *alpha-beta*) *beta-in-R*: **shows** β ∈ *carrier R*
  using *beta-def* **and** *pert-in-R* **and** *h-in-R* **by** *simp*

lemma (**in** *alpha-beta*) *alpha-i-in-R*: **shows** $\alpha(\hat{\ })_R$ (*i*::*nat*) ∈ *carrier R*
  using *alpha-in-R* **and** *R.nat-pow-closed* **by** *simp*

lemma (**in** *alpha-beta*) *beta-i-in-R*: **shows** $\beta(\hat{\ })_R$ (*i*::*nat*) ∈ *carrier R*
  using *beta-in-R* **and** *nat-pow-closed* **by** *simp*

lemma (**in** *ring*) *power-minus-a-b*:
  assumes *a*: *a* ∈ *carrier R* **and** *b*: *b* ∈ *carrier R* **shows** (⊖ (*a* ⊗ *b*)) (ˆ) *Suc n* = ⊖ *a* ⊗ ((⊖ (*b* ⊗ *a*)) (ˆ) *n*) ⊗ *b*
proof (*induct n*)
  case *0*
  from *a* **and** *b* **and** *nat-pow-0* [*of a* ⊗ *b*] **show** ⊖ (*a* ⊗ *b*) (ˆ) *Suc 0* = ⊖ *a* ⊗ ⊖ (*b* ⊗ *a*) (ˆ) (*0*::*nat*) ⊗ *b* **by** *simp algebra*
next
  case (*Suc n*)
  assume *hypo*: (⊖ (*a* ⊗ *b*)) (ˆ) *Suc n* = ⊖ *a* ⊗ ⊖ (*b* ⊗ *a*) (ˆ) *n* ⊗ *b*
  have (⊖ (*a* ⊗ *b*)) (ˆ) *Suc* (*Suc n*) = ⊖ (*a* ⊗ *b*) (ˆ) (*Suc n*) ⊗ ⊖ (*a* ⊗ *b*) **by** *simp*
  also from *hypo* **have** . . . = ⊖ *a* ⊗ ⊖ (*b* ⊗ *a*) (ˆ) *n* ⊗ *b* ⊗ ⊖ (*a* ⊗ *b*) **by** *simp*
  also from *sym* [*OF l-minus* [*OF a b*]] **have** . . . = ⊖ *a* ⊗ ⊖ (*b* ⊗ *a*) (ˆ) *n* ⊗ *b* ⊗ (⊖ *a* ⊗ *b*) **by** *simp*
  also from *sym* [*OF m-assoc* [*OF b - b*, *of* ⊖ *a*]] **and** *a-inv-closed* [*OF a*] **and** *nat-pow-closed* [*of* ⊖ (*b* ⊗ *a*) *n*]
    and *a-inv-closed* [*of b* ⊗ *a*] **and** *a b r-minus* [*OF b a*]
  have . . . = ⊖ *a* ⊗ ⊖ (*b* ⊗ *a*) (ˆ) *n* ⊗ (⊖ (*b* ⊗ *a*) ⊗ *b*) **by** *algebra*
  also from *nat-pow-closed* [*of* ⊖ (*b* ⊗ *a*) *n*] **and** *a-inv-closed* [*of b* ⊗ *a*] **and** *a b*
    and *sym* [*OF m-assoc*[*of* (⊖ (*b* ⊗ *a*)) (ˆ) *n* (⊖ (*b* ⊗ *a*)) *b*]]
  have . . . = ⊖ *a* ⊗ (⊖ (*b* ⊗ *a*) (ˆ) *n* ⊗ (⊖ (*b* ⊗ *a*))) ⊗ *b* **by** *algebra*

also from *sym* [*OF nat-pow-Suc* [*of* $\ominus$ ($b \otimes a$) *n*]]
have ... = $\ominus$ *a* $\otimes$ ($\ominus$ ($b \otimes a$)) (^) (*Suc n*) $\otimes$ *b* **by** *simp*
finally show $\ominus$ ($a \otimes b$) (^) *Suc* (*Suc n*) = $\ominus$ *a* $\otimes$ $\ominus$ ($b \otimes a$) (^) *Suc n* $\otimes$ *b* **by** *simp*
qed

The following comment is already obsolete in the *Isabelle-11−Feb−2007* repository version

Comment: At the moment, the "definition" command is not inherited by locales defined from old ones; in the following lemma, there would be two ways of recovering the definition of β. The first one would be to give its long name *local-nilpotent-alpha.*β δ, and the other way is to use abbreviations.

Due to aesthetic reasons, we choose the second solution, while waiting to the "definition" command to be properly inherited

abbreviation (*in local-nilpotent-alpha*) β == *alpha-beta.*β *R h* δ

The following lemma proves that whenever α is a local nilpotent term, so will be β

lemma (**in** *local-nilpotent-alpha*) *nilp-alpha-nilp-beta*: **shows** *local-nilpotent-term D R* β (λx. (*LEAST n::nat.* (β (^)$_R$ *n*) *x* = $\mathbf{1}_D$))
proof (*unfold local-nilpotent-alpha-def local-nilpotent-term-def local-nilpotent-term-axioms-def*, *simp*, *intro conjI*)
from *prems* **show** *ring-endomorphisms D R* **by** *intro-locales*
from *beta-in-R* **show** $\beta \in$ *carrier R* **by** *simp*
show $\forall x \in$*carrier D*. (β (^)$_R$ (*LEAST n::nat.* (β (^)$_R$ *n*) *x* = **1**)) *x* = **1**
proof (*intro ballI*)
fix *x* **assume** *x-in-D*: $x \in$ *carrier D*
show (β (^)$_R$ (*LEAST n::nat.* (β (^)$_R$ *n*) *x* = **1**)) *x* = **1**
proof (*rule LeastI-ex* [*of* λn*::nat.* (β (^)$_R$ *n*) *x* = **1**])
from *a-local-nilpot* **and** *pert-in-R* **and** *ring-R* **and** *hom-completion-closed* [*OF - x-in-D*, *of* δ *D*]
have *alpha-nilpot*: (α (^)$_R$ *bound-phi* (δ *x*)) (δ *x*) = **1** **by** *simp*
from *beta-def* **have** (β (^)$_R$ (*Suc* (*bound-phi* (δ *x*)))) *x* = (($\ominus_R$ ($h \otimes_R \delta$)) (^)$_R$ (*Suc* (*bound-phi* (δ *x*)))) *x* **by** *simp*
also from *h-in-R* **and** *pert-in-R* **and** *power-minus-a-b* [*OF h-in-R pert-in-R*, *of bound-phi* (δ *x*)]
have ... = ($\ominus_R$ *h* $\otimes_R$ ($\ominus_R$ ($\delta \otimes_R h$) (^)$_R$ (*bound-phi* (δ *x*))) $\otimes_R$ δ) *x* **by** *simp*
also from *alpha-nilpot* **and** *alpha-def* **and** *ring-R* **have** ... = ($\ominus_R$ *h*) **1** **by** *simp*
also from *a-inv-closed* [*OF h-in-R*] **and** *ring-R* **and** *hom-completion-one* [*of D D* $\ominus_R$ *h*] **and** *D-diff-group*
have ... = **1** **by** (*unfold diff-group-def comm-group-def group-def*, *simp*)
finally have *beta-nil*: (β (^)$_R$ (*Suc* (*bound-phi* (δ *x*)))) *x* = **1** **by** *simp*
from *exI* [*of* λn*::nat.* (β (^)$_R$ *n*) *x* = **1** (*Suc* (*bound-phi* (δ *x*)))] **and** *beta-nil*
show $\exists$ *n::nat.* (β (^)$_R$ *n*) *x* = **1** **by** *simp*

qed
qed
qed

lemma (**in** *local-nilpotent-alpha*) *bound-psi-exists*: **shows** $\exists$ *bound-psi*. *local-nilpotent-term* *D R* β *bound-psi*
using *nilp-alpha-nilp-beta* **by** *iprover*

context *local-nilpotent-alpha*
begin

definition *bound-psi* $\equiv$ $(\lambda x.\ (LEAST\ n::nat.\ (\beta\ (\hat{\ })_R\ n)\ x = \mathbf{1}_D))$

The definition of Ψ below is equivalent to the one given in the statement of Lemma 2.3.1 in Aransay's memoir

definition *psi-def*: $\Psi \equiv$ *local-nilpotent-term.power-series D R* β *bound-psi*

end

6.5 Some properties of the endomorphisms Φ, Ψ, α and β

lemma (**in** *local-nilpotent-alpha*) *local-nilpotent-term-alpha*: **shows** *local-nilpotent-term* *D R* α *bound-phi*
using *prems* **by** (*unfold local-nilpotent-alpha-def local-nilpotent-term-def*, *simp*)

lemma (**in** *local-nilpotent-alpha*) *local-nilpotent-term-beta*: **shows** *local-nilpotent-term* *D R* β *bound-psi*
using *prems* **and** *nilp-alpha-nilp-beta* **and** *bound-psi-def* **by** (*unfold local-nilpotent-term-def*, *simp*)

lemma (**in** *local-nilpotent-alpha*) *phi-x-in-D* [*simp*]: **shows** $\Phi\ x \in$ *carrier D*
using *phi-def local-nilpotent-term.power-series-def* [*OF local-nilpotent-term-alpha*, *of x*]
D.finprod-closed [*of* {..*bound-phi x*} $(\lambda i::nat.\ (\alpha(\hat{\ })_R i)\ x)$] **and** *finite-nat-iff-bounded* [*of* {..*bound-phi x*}]
and *local-nilpotent-term.power-pi-D* [*OF local-nilpotent-term-alpha*, *of x bound-phi x*] **by** *simp*

lemma (**in** *local-nilpotent-alpha*) *phi-in-R* [*simp*]: **shows** $\Phi \in$ *carrier R*
using *phi-def local-nilpotent-term.power-series-in-R* [*OF local-nilpotent-term-alpha*] **by** *simp*

lemma (**in** *local-nilpotent-alpha*) *phi-in-hom*: **shows** $\Phi \in$ *hom-completion D D*
using *phi-in-R* **and** *ring-R* **by** *simp*

lemma (**in** *local-nilpotent-alpha*) *psi-in-R* [*simp*]: **shows** $\Psi \in$ *carrier R*
using *psi-def local-nilpotent-term.power-series-in-R* [*OF local-nilpotent-term-beta*] **by** *simp*

lemma (**in** *local-nilpotent-alpha*) *psi-in-hom*: **shows** $\Psi \in$ *hom-completion D D*
using *psi-in-R* **and** *ring-R* **by** *simp*

lemma (**in** *local-nilpotent-alpha*) *psi-x-in-D* [*simp*]: **shows** Ψ *x* $\in$ *carrier D*
using *psi-def local-nilpotent-term.power-series-def* [*OF local-nilpotent-term-beta*, *of x*]
D.finprod-closed [*of* {..*bound-psi x*} (λ*i*::*nat*. (α(^)$_R$*i*) *x*)] **and** *finite-nat-iff-bounded* [*of* {..*bound-psi x*}]
and *local-nilpotent-term.power-pi-D* [*OF local-nilpotent-term-beta*, *of x bound-psi x*] **by** *simp*

lemma (**in** *local-nilpotent-alpha*) *h-alpha-eq-beta-h*: $h \otimes_R \alpha(\hat{\ })_R(i{::}nat) = \beta\ (\hat{\ })_R\ i \otimes_R h$
proof (*induct i*)
case *0*
from *h-in-R* **and** *R.nat-pow-0* **show** $h \otimes_R \alpha\ (\hat{\ })_R\ (0{::}nat) = \beta\ (\hat{\ })_R\ (0{::}nat) \otimes_R h$ **by** *simp*
next
case (*Suc i*)
assume *hypo*: $h \otimes_R \alpha\ (\hat{\ })_R\ i = \beta\ (\hat{\ })_R\ i \otimes_R h$
from *R.nat-pow-mult* [*OF beta-in-R*, *of* (*1*::*nat*) *i*] **and** *R.nat-pow-1* [*OF beta-in-R*]
have $\beta\ (\hat{\ })_R\ (Suc\ i) \otimes_R h = \beta \otimes_R \beta\ (\hat{\ })_R\ i \otimes_R h$ **by** *simp*
also from *hypo* **and** *R.m-assoc* [*OF beta-in-R beta-i-in-R* [*of i*] *h-in-R*] **have** . . . $= \beta \otimes_R (h \otimes_R \alpha\ (\hat{\ })_R\ i)$ **by** *simp*
also from *sym* [*OF R.m-assoc* [*OF beta-in-R h-in-R alpha-i-in-R* [*of i*]]] **and** *beta-def* **and** *h-in-R pert-in-R alpha-i-in-R* [*of i*]
have . . . $= h \otimes_R \ominus_R (\delta \otimes_R h) \otimes_R \alpha\ (\hat{\ })_R\ i$ **by** *algebra*
also from *R.m-assoc* [*OF h-in-R - alpha-i-in-R* [*of i*], *of* $\ominus_R (\delta \otimes_R h)$] *pert-in-R h-in-R*
have . . . $= h \otimes_R (\ominus_R (\delta \otimes_R h) \otimes_R \alpha\ (\hat{\ })_R\ i)$ **by** *simp*
also from *alpha-def* **and** *sym* [*OF R.nat-pow-1* [*OF alpha-in-R*]] **and** *R.nat-pow-mult* [*OF alpha-in-R*, *of* (*1*::*nat*) *i*]
have . . . $= h \otimes_R \alpha\ (\hat{\ })_R\ (Suc\ i)$ **by** *simp*
finally show $h \otimes_R \alpha\ (\hat{\ })_R\ Suc\ i = \beta\ (\hat{\ })_R\ Suc\ i \otimes_R h$ **by** *simp*
qed

6.6 Lemmas 2.2.1 to 2.2.6

Lemma 2.2.1

lemma (**in** *local-nilpotent-alpha*) *lemma-2-2-1*: **shows** *bound-psi* (*h x*) $\leq$ *bound-phi x*
proof −
from *h-alpha-eq-beta-h* **and** *ring-R* **have** ($\beta\ (\hat{\ })_R$ *bound-phi x*) (*h x*) = (*h* (($\alpha\ (\hat{\ })_R$ *bound-phi x*) *x*)) **by** (*auto simp add*: *expand-fun-eq*)
also have . . . = *h* (**1**)
proof (*cases x* $\in$ *carrier D*)
case *True* **with** *local-nilpotent-term.a-local-nilpot* [*OF local-nilpotent-term-alpha*] **show** *?thesis* **by** *simp*
next

case *False* **from** *alpha-i-in-R* [*of bound-phi x*] **and** *ring-R* **and** *completion-closed2* [*OF - False*, *of* α (^)$_R$ *bound-phi x D*]
show *?thesis* **unfolding** *hom-completion-def* **by** *simp*
qed
also from *hom-completion-one* [*of D D h*] *h-in-R ring-R* **and** *D-diff-group* **have** ... = **1** **unfolding** *diff-group-def comm-group-def group-def* **by** *simp*
finally have *beta-x-h-x-eq-one*: (β (^)$_R$ *bound-phi x*) (*h x*) = **1** **by** *simp*
have *beta-h-x-h-x-eq-one*: (β (^)$_R$ *bound-psi* (*h x*)) (*h x*) = **1**
proof (*cases h x* $\in$ *carrier D*)
case *True* **with** *local-nilpotent-term.a-local-nilpot* [*OF local-nilpotent-term-beta*] **show** *?thesis* **by** *simp*
next
case *False* **with** *ring-R* **and** *h-in-R*
show *?thesis* **unfolding** *hom-completion-def completion-fun2-def completion-def hom-def Pi-def* **by** *auto*
qed
from *beta-x-h-x-eq-one* **and** *beta-h-x-h-x-eq-one*
and *local-nilpotent-term.bound-is-least* [*OF local-nilpotent-term-beta*, *of h x*]
and *Least-le* [*of* λi::*nat*. (β (^)$_R$ *i*) (*h x*) = **1** *bound-phi x*] **show** *?thesis* **by** *simp*
qed

Lemma 2.2.3 with endomorphisms applied to elements

lemma (**in** *local-nilpotent-alpha*) *lemma-2-2-3-elements*: **shows** ($h \circ \Phi$) x = ($\Psi \circ h$) x
proof –
let *?max* = *max* (*bound-phi x*) (*bound-psi x*)
from *ring-R* **have** ($h \circ \Phi$) x = h (Φ x) **by** *simp*
also from *phi-def* **have** ... = h (*local-nilpotent-term.power-series D R* α *bound-phi x*) **by** *simp*
also from *local-nilpotent-term.power-series-extended* [*OF local-nilpotent-term-alpha*, *of x ?max*]
and *le-maxI1* [*of bound-phi x bound-psi x*]
have ... = h ($\bigotimes$ $i \in$ {..*?max*}. (α (^)$_R$ *i*) *x*) **by** *simp*
also from *local-nilpotent-term.power-series-dist-l* [*OF local-nilpotent-term-alpha h-in-R*, *of x ?max*] **and** *h-in-R*
have ... = ($\bigotimes$ $i \in$ {..*?max*}. *h* ((α (^)$_R$ *i*) *x*)) **by** *simp*
also from *h-alpha-eq-beta-h* **and** *ring-R* **have** ... = ($\bigotimes$ $i \in$ {..*?max*}. ((β (^)$_R$ *i*) (*h x*))) **by** (*auto simp add*: *expand-fun-eq*)
also from *sym* [*OF local-nilpotent-term.power-series-extended* [*OF local-nilpotent-term-beta*, *of h x ?max*]]
and *lemma-2-2-1* [*of x*]
have ... = *local-nilpotent-term.power-series D R* β *bound-psi* (*h x*) **by** *arith*
also from *psi-def* **have** ... = Ψ (*h x*) **by** *simp*
also have ... = ($\Psi \circ h$) x **by** *simp*
finally show *?thesis* **by** *simp*
qed

Lemma 2.2.3 with endomorphisms

corollary (**in** *local-nilpotent-alpha*) *lemma-2-2-3*: **shows** $(h \circ \Phi) = (\Psi \circ h)$ **using** *lemma-2-2-3-elements* **by** (*simp add*: *expand-fun-eq*)

The following lemma is simple a renaming of the previous one; the idea is to give to the previous result the name it had before as a premise, to keep the files correspoding to the equational part of the proof working

lemma (**in** *local-nilpotent-alpha*) *psi-h-h-phi*: **shows** $\Psi \otimes_R h = h \otimes_R \Phi$ **using** *lemma-2-2-3* **and** *ring-R* **by** *simp*

lemma (**in** *local-nilpotent-alpha*) *alpha-delta-eq-delta-beta*: **shows** $\alpha(\hat{}\,)_R(i::nat) \otimes_R \delta = \delta \otimes_R \beta\ (\hat{}\,)_R\ i$
proof (*induct i*)
  case *0*
  from *pert-in-R* **and** *R.nat-pow-0* **show** $\alpha\ (\hat{}\,)_R\ (0::nat) \otimes_R \delta = \delta \otimes_R \beta\ (\hat{}\,)_R\ (0::nat)$ **by** *simp*
next
  case (*Suc i*)
  assume *hypo*: $\alpha\ (\hat{}\,)_R\ i \otimes_R \delta = \delta \otimes_R \beta\ (\hat{}\,)_R\ i$
   from *R.nat-pow-mult* [*OF alpha-in-R*, *of* (*1*::*nat*) *i*] **and** *R.nat-pow-1* [*OF alpha-in-R*]
  have $\alpha\ (\hat{}\,)_R\ (Suc\ i) \otimes_R \delta = \alpha \otimes_R \alpha\ (\hat{}\,)_R\ i \otimes_R \delta$ **by** *simp*
   also from *hypo* **and** *R.m-assoc* [*OF alpha-in-R alpha-i-in-R* [*of i*] *pert-in-R*] **have** $\ldots = \alpha \otimes_R (\delta \otimes_R \beta\ (\hat{}\,)_R\ i)$ **by** *simp*
  also from *sym* [*OF R.m-assoc* [*OF alpha-in-R pert-in-R beta-i-in-R* [*of i*]]] **and** *alpha-def* **and** *h-in-R pert-in-R beta-i-in-R* [*of i*]
  have $\ldots = \delta \otimes_R \ominus_R (h \otimes_R \delta) \otimes_R \beta\ (\hat{}\,)_R\ i$ **by** *algebra*
 also from *R.m-assoc* [*OF pert-in-R - beta-i-in-R* [*of i*], *of* $\ominus_R (h \otimes_R \delta)$] *pert-in-R h-in-R*
  have $\ldots = \delta \otimes_R (\ominus_R (h \otimes_R \delta) \otimes_R \beta\ (\hat{}\,)_R\ i)$ **by** *simp*
 also from *beta-def* **and** *sym* [*OF R.nat-pow-1* [*OF beta-in-R*]] **and** *R.nat-pow-mult* [*OF beta-in-R*, *of* (*1*::*nat*) *i*]
  have $\ldots = \delta \otimes_R \beta\ (\hat{}\,)_R\ (Suc\ i)$ **by** *simp*
  finally show $\alpha\ (\hat{}\,)_R\ Suc\ i \otimes_R \delta = \delta \otimes_R \beta\ (\hat{}\,)_R\ Suc\ i$ **by** *simp*
qed

Lemma 2.2.2 in Aransay's memoir

lemma (**in** *local-nilpotent-alpha*) *lemma-2-2-2*: **shows** *bound-phi* $(\delta\ x) \leq$ *bound-psi* x
proof −
  from *alpha-delta-eq-delta-beta* **and** *ring-R* **have** $(\alpha\ (\hat{}\,)_R\ bound\text{-}psi\ x)\ (\delta\ x) = (\delta\ ((\beta\ (\hat{}\,)_R\ bound\text{-}psi\ x)\ x))$ **by** (*auto simp add*: *expand-fun-eq*)
  also have $\ldots = \delta\ (\mathbf{1})$
  proof (*cases* $x \in$ *carrier D*)
   case *True* **with** *local-nilpotent-term.a-local-nilpot* [*OF local-nilpotent-term-beta*] **show** *?thesis* **by** *simp*
  next
   case *False* **from** *beta-i-in-R* [*of bound-psi x*] **and** *ring-R* **and** *completion-closed2* [*OF - False*, *of* $\beta\ (\hat{}\,)_R\ bound\text{-}psi\ x\ \ D$]
    show *?thesis* **unfolding** *hom-completion-def* **by** *simp*

qed
also from *hom-completion-one* [*of D D* δ] *pert-in-R ring-R* **and** *D-diff-group*
have ... = **1**
unfolding *diff-group-def comm-group-def group-def* **by** *simp*
finally have *alpha-x-pert-x-eq-one*: (α (^)$_R$ *bound-psi x*) (δ *x*) = **1** **by** *simp*
have *alpha-pert-x-pert-x-eq-one*: (α (^)$_R$ *bound-phi* (δ *x*)) (δ *x*) = **1**
proof (*cases* δ *x* $\in$ *carrier D*)
case *True* **with** *local-nilpotent-term.a-local-nilpot* [*OF local-nilpotent-term-alpha*]
show *?thesis* **by** *simp*
next
case *False* **with** *ring-R* **and** *pert-in-R*
show *?thesis* **unfolding** *hom-completion-def completion-fun2-def completion-def hom-def Pi-def* **by** *auto*
qed
from *alpha-x-pert-x-eq-one* **and** *alpha-pert-x-pert-x-eq-one*
and *local-nilpotent-term.bound-is-least* [*OF local-nilpotent-term-alpha*, *of* δ *x*]
and *Least-le* [*of* λ*i*::*nat*. (α (^)$_R$ *i*) (δ *x*) = **1** *bound-psi x*] **show** *?thesis* **by** *simp*
qed

Lemma 2.2.4 over endomorphisms applied to generic elements

lemma (**in** *local-nilpotent-alpha*) *lemma-2-2-4-elements*: **shows** ($\delta \circ \Psi$) *x* = ($\Phi \circ \delta$) *x*
proof −
let *?max* = *max* (*bound-psi x*) (*bound-phi x*)
from *ring-R* **have** ($\delta \circ \Psi$) *x* = δ (Ψ *x*) **by** *simp*
also from *psi-def* **have** ... = δ (*local-nilpotent-term.power-series D R* β *bound-psi x*) **by** *simp*
also from *local-nilpotent-term.power-series-extended* [*OF local-nilpotent-term-beta*, *of x ?max*]
and *le-maxI1* [*of bound-psi x bound-phi x*]
have ... = δ ($\bigotimes$ *i*$\in${..*?max*}. (β (^)$_R$ *i*) *x*) **by** *simp*
also from *local-nilpotent-term.power-series-dist-l* [*OF local-nilpotent-term-beta pert-in-R*, *of x ?max*] **and** *pert-in-R*
have ... = ($\bigotimes$ *i*$\in${..*?max*}. δ ((β (^)$_R$ *i*) *x*)) **by** *simp*
also from *alpha-delta-eq-delta-beta* **and** *ring-R* **have** ... = ($\bigotimes$ *i*$\in${..*?max*}. ((α (^)$_R$ *i*) (δ *x*))) **by** (*auto simp add*: *expand-fun-eq*)
also from *sym* [*OF local-nilpotent-term.power-series-extended* [*OF local-nilpotent-term-alpha*, *of* δ *x ?max*]]
and *lemma-2-2-2* [*of x*] **have** ... = *local-nilpotent-term.power-series D R* α *bound-phi* (δ *x*) **by** *arith*
also from *phi-def* **have** ... = ($\Phi \circ \delta$) *x* **by** *simp*
finally show *?thesis* **by** *simp*
qed

Lemma 2.2.4 over endomorphisms

corollary (**in** *local-nilpotent-alpha*) *lemma-2-2-4*: **shows** ($\delta \circ \Psi$) = ($\Phi \circ \delta$) **using** *lemma-2-2-4-elements* **by** (*simp add*: *expand-fun-eq*)

The following lemma is simple a renaming of the previous one; the idea is to give to the previous result the name it had before as a premise, to keep the files correspoding to the equational part of the proof working

lemma (**in** *local-nilpotent-alpha*) *delta-psi-phi-delta*: **shows** $\delta \otimes_R \Psi = \Phi \otimes_R \delta$ **using** *lemma-2-2-4* **and** *ring-R* **by** *simp*

Lemma 2.2.5 over a generic element of the domain

lemma (**in** *local-nilpotent-alpha*) *lemma-2-2-5-elements*: **shows** $\Psi\ x = (\mathbf{1}_R \ominus_R (h \otimes_R \delta \otimes_R \Psi))\ x$ **and** $\Psi\ x = (\mathbf{1}_R \ominus_R (h \otimes_R \Phi \otimes_R \delta))\ x$
and $\Psi\ x = (\mathbf{1}_R \ominus_R (\Psi \otimes_R h \otimes_R \delta))\ x$
proof −
from *psi-def* **have** $\Psi\ x =$ *local-nilpotent-term.power-series D R* β *bound-psi x* **by** *simp*
also from *local-nilpotent-term.power-series-extended* [*OF local-nilpotent-term-beta*, *of x Suc* (*bound-psi x*)]
have $\ldots = (\bigotimes j \in \{..Suc\ (bound\text{-}psi\ x)\}.\ (\beta\ (\hat{\ })_R\ j)\ x)$ **by** *simp*
also from *local-nilpotent-term.power-series-first-element* [*OF local-nilpotent-term-beta*, *of x Suc* (*bound-psi x*)]
have $\ldots = (\beta\ (\hat{\ })_R\ (0{::}nat))\ x \otimes (\bigotimes j \in \{1..Suc\ (bound\text{-}psi\ x)\}.\ (\beta\ (\hat{\ })_R\ j)\ x)$ **by** *simp*
also from *local-nilpotent-term.power-series-factor* [*OF local-nilpotent-term-beta*, *of x bound-psi x*]
have $\ldots = (\beta\ (\hat{\ })_R\ (0{::}nat))\ x \otimes (\beta\ (\bigotimes j \in \{..bound\text{-}psi\ x\}.\ (\beta\ (\hat{\ })_R\ j)\ x))$ **by** *simp*
also from *R.nat-pow-0* [*of* β] **and** *beta-def* **and** *psi-def*
and *local-nilpotent-term.power-series-def* [*OF local-nilpotent-term-beta*, *of x*] *ring-R*
have $\ldots = \mathbf{1}_R\ x \otimes (\ominus_R (h \otimes_R \delta) \otimes_R \Psi)\ x$ **by** *simp*
also have $\ldots = (\mathbf{1}_R \oplus_R (\ominus_R (h \otimes_R \delta) \otimes_R \Psi))\ x$
proof (*cases* $x \in$ *carrier D*)
case *True* **with** *ring-R* **show** *?thesis* **by** *simp*
next
case *False*
with *ring-R* **have** *one-x*: $\mathbf{1}_R\ x = \mathbf{1}$ **by** *simp*
moreover
from *h-in-R pert-in-R* **and** *psi-in-R* **have** $(\ominus_R (h \otimes_R \delta) \otimes_R \Psi) \in$ *carrier R* **by** *simp*
with *False* **and** *ring-R* **and** *completion-closed2* [*of* $(\ominus_R (h \otimes_R \delta) \otimes_R \Psi)$ *D D x*] **have** *h-pert-psi*: $(\ominus_R (h \otimes_R \delta) \otimes_R \Psi)\ x = \mathbf{1}$
by (*unfold hom-completion-def*, *simp*)
moreover
from *h-in-R pert-in-R* **and** *psi-in-R* **have** $\mathbf{1}_R \oplus_R (\ominus_R (h \otimes_R \delta) \otimes_R \Psi) \in$ *carrier R* **by** *simp*
with *False* **and** *ring-R* **and** *completion-closed2* [*of* $(\ominus_R (h \otimes_R \delta) \otimes_R \Psi)$ *D D x*]
have *one-h-pert-psi*: $(\mathbf{1}_R \oplus_R \ominus_R (h \otimes_R \delta) \otimes_R \Psi)\ x = \mathbf{1}$ **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
also from *R.one-closed* **and** *pert-in-R* **and** *h-in-R* **and** *psi-in-R* **have** $\ldots = (\mathbf{1}_R \ominus_R (h \otimes_R \delta \otimes_R \Psi))\ x$ **by** *algebra*

finally show *psi-eq*: $\Psi\ x = (\mathbf{1}_R \ominus_R (h \otimes_R \delta \otimes_R \Psi))\ x$ **by** *simp*
from *lemma-2-2-4* **and** *R.m-assoc* [*OF h-in-R pert-in-R psi-in-R*] **and** *sym* [*OF R.m-assoc* [*OF h-in-R phi-in-R pert-in-R*]] **and** *ring-R* **and** *psi-eq*
show *psi-eq2*: $\Psi\ x = (\mathbf{1}_R \ominus_R (h \otimes_R \Phi \otimes_R \delta))\ x$ **by** *simp*
with *lemma-2-2-3* **and** *ring-R* **show** $\Psi\ x = (\mathbf{1}_R \ominus_R (\Psi \otimes_R h \otimes_R \delta))\ x$ **by** *simp*
qed

Lemma 2.2.5 in generic terms

lemma (**in** *local-nilpotent-alpha*) *lemma-2-2-5*: **shows** $\Psi = \mathbf{1}_R \ominus_R (h \otimes_R \delta \otimes_R \Psi)$ **and** $\Psi = \mathbf{1}_R \ominus_R (h \otimes_R \Phi \otimes_R \delta)$
and $\Psi = \mathbf{1}_R \ominus_R (\Psi \otimes_R h \otimes_R \delta)$ **using** *lemma-2-2-5-elements* **by** (*simp-all add*: *expand-fun-eq*)

The following lemma is simple a renaming of the previous one; the idea is to give to the previous result the name it had before as a premise, to keep the proofs correspoding to the equational part of the proof working

lemma (**in** *local-nilpotent-alpha*) *psi-prop*: **shows** $\Psi = \mathbf{1}_R \ominus_R (h \otimes_R \delta \otimes_R \Psi)$ **and** $\Psi = \mathbf{1}_R \ominus_R (h \otimes_R \Phi \otimes_R \delta)$
and $\Psi = \mathbf{1}_R \ominus_R (\Psi \otimes_R h \otimes_R \delta)$ **using** *lemma-2-2-5* **.**

Lemma 2.2.6 over a generic element of the domain

lemma (**in** *local-nilpotent-alpha*) *lemma-2-2-6-elements*: **shows** $\Phi\ x = (\mathbf{1}_R \ominus_R (\delta \otimes_R h \otimes_R \Phi))\ x$ **and** $\Phi\ x = (\mathbf{1}_R \ominus_R (\delta \otimes_R \Psi \otimes_R h))\ x$
and $\Phi\ x = (\mathbf{1}_R \ominus_R (\Phi \otimes_R \delta \otimes_R h))\ x$
proof −
from *phi-def* **have** $\Phi\ x =$ *local-nilpotent-term.power-series D R* α *bound-phi x* **by** *simp*
also from *local-nilpotent-term.power-series-extended* [*OF local-nilpotent-term-alpha*, *of x Suc* (*bound-phi x*)]
have $\ldots = (\bigotimes j\in\{..Suc\ (bound\text{-}phi\ x)\}.\ (\alpha\ (\hat{\ })_R\ j)\ x)$ **by** *simp*
also from *local-nilpotent-term.power-series-first-element* [*OF local-nilpotent-term-alpha*, *of x Suc* (*bound-phi x*)]
have $\ldots = (\alpha\ (\hat{\ })_R\ (0{::}nat))\ x \otimes (\bigotimes j\in\{1..Suc\ (bound\text{-}phi\ x)\}.\ (\alpha\ (\hat{\ })_R\ j)\ x)$ **by** *simp*
also from *local-nilpotent-term.power-series-factor* [*OF local-nilpotent-term-alpha*, *of x bound-phi x*]
have $\ldots = (\alpha\ (\hat{\ })_R\ (0{::}nat))\ x \otimes (\alpha\ (\bigotimes j\in\{..bound\text{-}phi\ x\}.\ (\alpha\ (\hat{\ })_R\ j)\ x))$ **by** *simp*
also from *R.nat-pow-0* [*of* α] **and** *alpha-def* **and** *phi-def* **and** *local-nilpotent-term.power-series-def* [*OF local-nilpotent-term-alpha*, *of x*]
ring-R
have $\ldots = \mathbf{1}_R\ x \otimes (\ominus_R (\delta \otimes_R h) \otimes_R \Phi)\ x$ **by** *simp*
also have $\ldots = (\mathbf{1}_R \oplus_R (\ominus_R (\delta \otimes_R h) \otimes_R \Phi))\ x$
proof (*cases* $x \in$ *carrier D*)
case *True* **with** *ring-R* **show** *?thesis* **by** *simp*
next
case *False*
with *ring-R* **have** *one-x*: $\mathbf{1}_R\ x = \mathbf{1}$ **by** *simp*

moreover
from *h-in-R pert-in-R* **and** *phi-in-R* **have** $(\ominus_R (\delta \otimes_R h) \otimes_R \Phi) \in$ *carrier R* **by** *simp*
with *False* **and** *ring-R* **and** *completion-closed2* [*of* $(\ominus_R (\delta \otimes_R h) \otimes_R \Phi)$ *D D x*] **have** *h-pert-psi*: $(\ominus_R (\delta \otimes_R h) \otimes_R \Phi)\ x = \mathbf{1}$
unfolding *hom-completion-def* **by** *simp*
moreover
from *h-in-R pert-in-R* **and** *phi-in-R* **have** $\mathbf{1}_R \oplus_R (\ominus_R (\delta \otimes_R h) \otimes_R \Phi) \in$ *carrier R* **by** *simp*
with *False* **and** *ring-R* **and** *completion-closed2* [*of* $(\ominus_R (\delta \otimes_R h) \otimes_R \Phi)$ *D D x*]
have *one-h-pert-psi*: $(\mathbf{1}_R \oplus_R \ominus_R (\delta \otimes_R h) \otimes_R \Phi)\ x = \mathbf{1}$ **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
also from *R.one-closed* **and** *pert-in-R* **and** *h-in-R* **and** *phi-in-R* **have** $\ldots = (\mathbf{1}_R \ominus_R (\delta \otimes_R h \otimes_R \Phi))\ x$ **by** *algebra*
finally show *phi-eq*: $\Phi\ x = (\mathbf{1}_R \ominus_R (\delta \otimes_R h \otimes_R \Phi))\ x$ **by** *simp*
from *lemma-2-2-3* **and** *R.m-assoc* [*OF pert-in-R h-in-R phi-in-R*] **and** *sym* [*OF R.m-assoc* [*OF pert-in-R psi-in-R h-in-R*]] **and** *ring-R* **and** *phi-eq*
show *phi-eq2*: $\Phi\ x = (\mathbf{1}_R \ominus_R (\delta \otimes_R \Psi \otimes_R h))\ x$ **by** *simp*
with *lemma-2-2-4* **and** *ring-R* **show** $\Phi\ x = (\mathbf{1}_R \ominus_R (\Phi \otimes_R \delta \otimes_R h))\ x$ **by** *simp*
qed

Lemma 2.2.6

lemma (**in** *local-nilpotent-alpha*) *lemma-2-2-6*: **shows** $\Phi = (\mathbf{1}_R \ominus_R (\delta \otimes_R h \otimes_R \Phi))$ **and** $\Phi = (\mathbf{1}_R \ominus_R (\delta \otimes_R \Psi \otimes_R h))$
and $\Phi = (\mathbf{1}_R \ominus_R (\Phi \otimes_R \delta \otimes_R h))$ **using** *lemma-2-2-6-elements* **by** (*simp-all add*: *expand-fun-eq*)

The following lemma is simple a renaming of the previous one; the idea is to give to the previous result the name it had before as a premise, to keep the proofs correspoding to the equational part of the proof working

lemma (**in** *local-nilpotent-alpha*) *phi-prop*: **shows** $\Phi = \mathbf{1}_R \ominus_R (\delta \otimes_R h \otimes_R \Phi)$ **and** $\Phi = \mathbf{1}_R \ominus_R (\delta \otimes_R \Psi \otimes_R h)$
and $\Phi = \mathbf{1}_R \ominus_R (\Phi \otimes_R \delta \otimes_R h)$ **using** *lemma-2-2-6* **.**

end

7 Lemma 2.2.15 in Aransay's memoir

theory *lemma-2-2-15-local-nilpot*
imports
analytic-part-local
begin

We define a locale setting merging the specifications introduced for *lemma 2.2.14* and also the one created for the *local nilpotent term alpha*

A few definitions are also provided in this locale setting

locale *lemma-2-2-15* = *lemma-2-2-14 D R h* + *local-nilpotent-alpha D R C f g h* δ α *bound-phi*

context *lemma-2-2-15*
begin

definition h' **where** $h' == h \otimes_R \Phi$

definition p' **where** $p' == ((differ_D) \oplus_R \delta) \otimes_R h' \oplus_R h' \otimes_R ((differ_D) \oplus_R \delta)$

definition *diff'* **where** $diff' == differ \oplus_R \delta$

definition D' **where** $D' == (\!|\ carrier = carrier\ D,\ mult = mult\ D,\ one = one\ D,\ diff = differ \oplus_R \delta |\!)$

definition *ker-p'* **where** $ker\text{-}p' == kernel\ (\!|\ carrier = carrier\ D,\ mult = mult\ D,\ one = one\ D,\ diff = differ \oplus_R \delta |\!)$
$(\!|\ carrier = carrier\ D,\ mult = mult\ D,\ one = one\ D,\ diff = differ \oplus_R \delta |\!)\ p'$

definition *diff-group-ker-p'*
where $diff\text{-}group\text{-}ker\text{-}p' == (\!|carrier = kernel\ (\!|\ carrier = carrier\ D,\ mult = mult\ D,\ one = one\ D,\ diff = differ \oplus_R \delta |\!)$
$(\!|\ carrier = carrier\ D,\ mult = mult\ D,\ one = one\ D,\ diff = differ \oplus_R \delta |\!)\ p',\ mult = mult\ D,$
$one = one\ D,\ diff = completion\ ((\!|carrier = kernel\ (\!|\ carrier = carrier\ D,\ mult = mult\ D,\ one = one\ D,\ diff = differ \oplus_R \delta |\!)$
$(\!|\ carrier = carrier\ D,\ mult = mult\ D,\ one = one\ D,\ diff = differ \oplus_R \delta |\!)\ p',\ mult = mult\ D,\ one = one\ D,\ diff = differ \oplus_R \delta |\!))$
$(\!|\ carrier = carrier\ D,\ mult = mult\ D,\ one = one\ D,\ diff = differ \oplus_R \delta |\!)\ (differ \oplus_R \delta)\ |\!)$

definition *inc-ker-p'* **where** $inc\text{-}ker\text{-}p' == (\lambda x.\ if\ x \in kernel\ (\!|\ carrier = carrier\ D,\ mult = mult\ D,\ one = one\ D,\ diff = differ \oplus_R \delta |\!)$
$(\!|\ carrier = carrier\ D,\ mult = mult\ D,\ one = one\ D,\ diff = differ \oplus_R \delta |\!)\ \ p'$ *then* x *else* $\mathbf{1}_{D'}$)

end

lemma (**in** *lemma-2-2-15*) *h'-in-R* [*simp*]: **shows** $h' \in carrier\ R$ **using** *h'-def* **by** *simp*

lemma (**in** *lemma-2-2-15*) *pert-in-R* [*simp*]: **shows** $\delta \in carrier\ R$ **using** *delta-pert* **and** *pert-def* [*of D*] **by** *simp*

lemma (**in** *lemma-2-2-15*) *p'-in-R* [*simp*]: **shows** $p' \in carrier\ R$ **using** *p'-def* **and** *diff-pert-in-R* [*OF delta-pert*] **and** *h'-in-R* **by** *simp*

lemma (**in** *lemma-2-2-15*) *diff'-in-R* [*simp*]: **shows** $diff' \in carrier\ R$ **using** *diff'-def diff-in-R pert-in-R* **by** *simp*

The endomorphisms (not the differential endomorphisms) over a differential group happen to be the same ones as the homomorphisms over a perturbed version of this differential group

In other words, the definition of homomorphism over a differential group is independient of the differential

In the case of differential homomorphisms, this is not always true

lemma (**in** *ring-endomorphisms*) *hom-completion-eq*: **assumes** $\delta \in$ *pert D*
shows *hom-completion* (|*carrier* = *carrier D*, *mult* = *mult D*, *one* = *one D*, *diff* = *differ* $\oplus_R$ δ|)
(|*carrier* = *carrier D*, *mult* = *mult D*, *one* = *one D*, *diff* = *differ* $\oplus_R$ δ|) = *hom-completion D D*
using *ring-R* **unfolding** *hom-completion-def completion-fun2-def completion-def hom-def expand-fun-eq* **by** *auto*

lemma (**in** *ring-endomorphisms*) *ring-endomorphisms-pert*: **assumes** *delta*: $\delta \in$ *pert D*
shows *ring-endomorphisms* (|*carrier* = *carrier D*, *mult* = *mult D*, *one* = *one D*, *diff* = *differ* $\oplus_R$ δ|) *R*
(**is** *ring-endomorphisms ?D′ R*)
proof −
from *diff-group-pert-is-diff-group* [*OF delta*] **have** *diff-group-pert*: *diff-group ?D′* **by** *simp*
moreover from *prems* **have** *R-ring*: *ring R* **unfolding** *ring-endomorphisms-def* [*of D R*] **by** *simp*
moreover from *ring-R* **have** *R* = (|*carrier* = *hom-completion ?D′ ?D′*, *mult* = *op* ∘,
one = λx. *if* $x \in$ *carrier ?D′ then id x else* $\mathbf{1}_{?D'}$,
zero = λx. *if* $x \in$ *carrier ?D′ then* $\mathbf{1}_{?D'}$ *else* $\mathbf{1}_{?D'}$,
add = $\lambda f\ g\ x$. *if* $x \in$ *carrier ?D′ then f x* $\otimes_{?D'}$ *g x else* $\mathbf{1}_{?D'}$|)
proof −
from *ring-R* **and** *hom-completion-eq* [*OF delta*] **have** *carrier R* = *hom-completion ?D′ ?D′* **by** *simp*
moreover from *ring-R* **have** *mult R* = *op* ∘ **by** *simp*
moreover from *ring-R* **have** *one R* = (λx. *if* $x \in$ *carrier ?D′ then id x else* $\mathbf{1}_{?D'}$) **by** (*simp add*: *expand-fun-eq*)
moreover from *ring-R* **have** *zero R* = (λx. *if* $x \in$ *carrier ?D′ then* $\mathbf{1}_{?D'}$ *else* $\mathbf{1}_{?D'}$) **by** *simp*
moreover from *ring-R* **have** *add R* = ($\lambda f\ g\ x$. *if* $x \in$ *carrier ?D′ then f x* $\otimes_{?D'}$ *g x else* $\mathbf{1}_{?D'}$) **by** (*simp add*: *expand-fun-eq*)
ultimately show *?thesis* **by** *auto*

qed
ultimately show *?thesis* **unfolding** *ring-endomorphisms-def ring-endomorphisms-axioms-def Ring.ring-def diff-group-def* **by** *simp*
qed

The two following lemmas prove that $h' \otimes_R h' = \mathbf{0}_R$ and $h' \otimes_R$ *diff′* $\otimes_R$

$h' = h'$; these are the properties that will allow us to introduce *reduction D diff-group-ker-p* $(\mathbf{1}_R \ominus_R p)$ *inc-ker-p h* in order to define the reduction needed for *Lemma 2.2.15*

lemma (**in** *lemma-2-2-15*) *h'-nil*: **shows** $h' \otimes_R h' = \mathbf{0}_R$
proof −
  from *h'-def* **have** $h' \otimes_R h' = (h \otimes_R \Phi) \otimes_R (h \otimes_R \Phi)$ **by** *simp*
  also from *psi-h-h-phi* **and** *h-in-R phi-in-R psi-in-R* **and** *R.m-assoc* [*of* Ψ h (h $\otimes_R$ Φ)] **and** *R.m-assoc* [*of* h h Φ]
  have $\ldots = \Psi \otimes_R (h \otimes_R h \otimes_R \Phi)$ **by** *simp*
  also from *h-nil* **have** $\ldots = \mathbf{0}_R$ **by** *simp*
  finally show *?thesis* **by** *simp*
qed

lemma (**in** *lemma-2-2-15*) *h'd'h'-h'*: **shows** $h' \otimes_R \mathit{diff}' \otimes_R h' = h'$

proof −
  have $h' \otimes_R \mathit{diff}' \otimes_R h' = (h \otimes_R \Phi) \otimes_R \mathit{diff}' \otimes_R (h \otimes_R \Phi)$ **unfolding** *h'-def* **by** *simp*
  also from *psi-h-h-phi* **have** $\ldots = (\Psi \otimes_R h) \otimes_R \mathit{diff}' \otimes_R (h \otimes_R \Phi)$ **by** *simp*
  also from *h-in-R phi-in-R psi-in-R diff'-def diff-in-R pert-in-R*
  have $\ldots = (\Psi \otimes_R (h \otimes_R (\mathit{differ}_D) \otimes_R h \otimes_R \Phi)) \oplus_R (\Psi \otimes_R h) \otimes_R ((\delta \otimes_R h) \otimes_R \Phi)$ **by** *algebra*
  also have $\ldots = (\Psi \otimes_R (h \otimes_R \Phi)) \oplus_R (\Psi \otimes_R h) \otimes_R (\mathbf{1}_R \ominus_R \Phi)$
  proof −
    from *phi-prop* **have** $\Phi = \mathbf{1}_R \ominus_R \delta \otimes_R h \otimes_R \Phi$ **by** *simp*
    with *h-in-R phi-in-R pert-in-R* **have** $\mathbf{1}_R \ominus_R \Phi = \mathbf{1}_R \ominus_R (\mathbf{1}_R \ominus_R \delta \otimes_R h \otimes_R \Phi)$ **by** *algebra*
    with *h-in-R phi-in-R pert-in-R* **have** *r-h-p*: $\mathbf{1}_R \ominus_R \Phi = \delta \otimes_R h \otimes_R \Phi$ **by** *algebra*
    from *hdh-h* **have** *l-h-p*: $h \otimes_R (\mathit{differ}_D) \otimes_R h = h$ **by** *simp*
    from *r-h-p* **and** *l-h-p* **show** *?thesis* **by** *simp*
  qed
  also from *h-in-R phi-in-R psi-in-R diff-in-R pert-in-R* **have** $\ldots = \Psi \otimes_R h \otimes_R \Phi \oplus_R \Psi \otimes_R h \ominus_R \Psi \otimes_R h \otimes_R \Phi$ **by** *algebra simp*

  also from *h-in-R phi-in-R psi-in-R diff-in-R pert-in-R* **have** $\ldots = \Psi \otimes_R h$ **by** *algebra*
  also from *psi-h-h-phi* **have** $\ldots = h \otimes_R \Phi$ **by** *simp*
  also from *h'-def* **have** $\ldots = h'$ **by** *simp*
  finally show *?thesis* **by** *simp*
qed

The following lemma is an instantiation of *lemma-2-2-14*, where $D' =$ (|*carrier* = *carrier D*, *mult* = *op* $\otimes$, *one* = $\mathbf{1}$, *diff* = *differ* $\oplus_R$ δ|) $R = R$, and finally $h = h \otimes_R \Phi$.

Therefore, the premises of locale *lemma-2-2-14* have to be verified

It is not neccesary to explicitly prove that *diff-group-ker-p′* is a differential group, since it is one of the premises in the definition of reduction

lemma (**in** *lemma-2-2-15*) *lemma-2-2-15*: **shows** *reduction D′ diff-group-ker-p′* $(\mathbf{1}_R \ominus_R p')$ *inc-ker-p′ h′*
proof −

from *diff-group-pert-is-diff-group* **and** *delta-pert* **have** *diff-group-D′*: *diff-group D′* **unfolding** *D′-def* **by** *simp*
moreover
from *h′-in-R* **and** *ring-R* **and** *hom-completion-eq* [*OF delta-pert*] **have** $h' \in$ *hom-completion D′ D′* **unfolding** *D′-def* **by** *simp*
moreover
from *h′-nil* **have** *h′-nilpot*: $h' \otimes_R h' = \mathbf{0}_R$ **by** *simp*
moreover
from *h′d′h′-h′* **have** $h' \otimes_R$ *diff′* $\otimes_R h' = h'$ **by** *simp*
moreover
from *ring-endomorphisms-pert* [*OF delta-pert*] **and** *D′-def* **have** *ring-D′*: *ring-endomorphisms D′ R* **unfolding** *ring-endomorphisms-def* **by** *simp*

ultimately have *lemma-2-2-14*: *lemma-2-2-14 D′ R h′*
unfolding *lemma-2-2-14-def lemma-2-2-14-axioms-def diff-group-def ring-endomorphisms-def diff′-def D′-def* **by** *simp*
show *?thesis*
using *lemma-2-2-14.lemma-2-2-14* [*OF lemma-2-2-14*]
unfolding *lemma-2-2-14.p-def* [*OF lemma-2-2-14*] *lemma-2-2-14.inc-ker-p-def* [*OF lemma-2-2-14*]
lemma-2-2-14.diff-group-ker-p-def [*OF lemma-2-2-14*]
unfolding *diff-group-ker-p′-def inc-ker-p′-def inc-ker-p-def p′-def D′-def* **by** *simp*
qed

end

8 Proposition 2.2.16 and Lemma 2.2.17 in Aransay's memoir

theory *lemma-2-2-17-local-nilpot*
imports
lemma-2-2-15-local-nilpot
begin

8.1 Previous definitions

Locale *proposition-2-2-16* does not introduce new facts; only some new definitions are given in the locale

locale *proposition-2-2-16* = *lemma-2-2-15*

context *proposition-2-2-16*
begin

definition π **where** $\pi = \mathbf{1}_R \ominus_R p$

definition π' **where** $\pi' = \mathbf{1}_R \ominus_R p'$

end

The following lemma has been extracted from the proof of *Proposition 2.2.16* as stated in the memoir

lemma (**in** *proposition-2-2-16*) *hp′-h* [*simp*]: **shows** $h \otimes_R p' = h$ **and** $p' \otimes_R h = h$

proof −
show $h \otimes_R p' = h$
proof −
from *p′-def* **and** *diff′-def* **have** $h \otimes_R p' = h \otimes_R ((differ \oplus_R \delta) \otimes_R h' \oplus_R h' \otimes_R (differ \oplus_R \delta))$ **by** *simp*
also from *diff-in-R* **and** *pert-in-R* **and** *h′-in-R* **and** *h-in-R*
have $\ldots = h \otimes_R ((differ \oplus_R \delta) \otimes_R h') \oplus_R h \otimes_R (h' \otimes_R (differ \oplus_R \delta))$ **by** *algebra*
also from *diff-in-R* **and** *pert-in-R* **and** *h′-in-R* **and** *h-in-R*
have $\ldots = h \otimes_R (differ \otimes_R h' \oplus_R \delta \otimes_R h') \oplus_R h \otimes_R h' \otimes_R (differ \oplus_R \delta)$ **by** *algebra*
also from *diff-in-R* **and** *pert-in-R* **and** *h′-in-R* **and** *h-in-R* **and** *h′-def*
have $\ldots = h \otimes_R (differ \otimes_R (h \otimes_R \Phi)) \oplus_R h \otimes_R (\delta \otimes_R (h \otimes_R \Phi)) \oplus_R h \otimes_R (h \otimes_R \Phi) \otimes_R (differ \oplus_R \delta)$ **by** *algebra*
also from *diff-in-R* **and** *pert-in-R* **and** *h-in-R* **and** *phi-in-R*
have $\ldots = h \otimes_R differ \otimes_R h \otimes_R \Phi \oplus_R h \otimes_R (\delta \otimes_R h \otimes_R \Phi) \oplus_R h \otimes_R h \otimes_R \Phi \otimes_R (differ \oplus_R \delta)$ **by** *algebra*
also have $\ldots = h \otimes_R \Phi \oplus_R h \otimes_R (\mathbf{1}_R \ominus_R \Phi)$
proof −
from *phi-prop* **have** $\Phi = \mathbf{1}_R \ominus_R \delta \otimes_R h \otimes_R \Phi$ **by** *simp*
with *phi-in-R* **and** *h-in-R* **and** *pert-in-R* **have** $\mathbf{1}_R \ominus_R \Phi = \mathbf{1}_R \ominus_R (\mathbf{1}_R \ominus_R \delta \otimes_R h \otimes_R \Phi)$ **by** *algebra*
with *phi-in-R* **and** *h-in-R* **and** *pert-in-R* **have** $\mathbf{1}_R \ominus_R \Phi = \delta \otimes_R h \otimes_R \Phi$ **by** *algebra*
with *diff-in-R* **and** *pert-in-R* **and** *h′-in-R* **and** *h-in-R* **and** *phi-in-R* **and** *hdh-h* **and** *h-nil*
show *?thesis* **by** *algebra*
qed
also from *h-in-R* **and** *phi-in-R* **and** *R.r-one* [*OF h-in-R*] **have** $\ldots = h$ **by** *algebra*
finally show $h \otimes_R p' = h$ **by** *simp*
qed
next
show $p' \otimes_R h = h$
proof −

from *p′-def* **and** *diff′-def* **have** $p' \otimes_R h = ((\mathit{differ} \oplus_R \delta) \otimes_R h' \oplus_R h' \otimes_R (\mathit{differ} \oplus_R \delta)) \otimes_R h$ **by** *simp*
also from *diff-in-R* **and** *h′-in-R* **and** *h-in-R* **and** *pert-in-R*
have $\ldots = ((\mathit{differ} \oplus_R \delta) \otimes_R h') \otimes_R h \oplus_R (h' \otimes_R (\mathit{differ} \oplus_R \delta)) \otimes_R h$ **by** *algebra*
also from *diff-in-R* **and** *h′-in-R* **and** *h-in-R* **and** *pert-in-R*
have $\ldots = (\mathit{differ} \oplus_R \delta) \otimes_R (h' \otimes_R h) \oplus_R ((h' \otimes_R \mathit{differ}) \oplus_R (h' \otimes_R \delta)) \otimes_R h$ **by** *algebra*
also from *h′-def* **and** *diff-in-R* **and** *h′-in-R* **and** *h-in-R* **and** *pert-in-R*
have $\ldots = (\mathit{differ} \oplus_R \delta) \otimes_R (h \otimes_R \Phi \otimes_R h) \oplus_R h' \otimes_R \mathit{differ} \otimes_R h \oplus_R h' \otimes_R \delta \otimes_R h$ **by** *algebra*
also from *psi-h-h-phi* *h′-def* **and** *diff-in-R* **and** *h′-in-R* **and** *h-in-R* **and** *pert-in-R*
have $\ldots = (\mathit{differ} \oplus_R \delta) \otimes_R (\Psi \otimes_R h \otimes_R h) \oplus_R h \otimes_R \Phi \otimes_R \mathit{differ} \otimes_R h \oplus_R h \otimes_R \Phi \otimes_R \delta \otimes_R h$ **by** *algebra*
also have $\ldots = h \otimes_R \Phi \otimes_R \mathit{differ} \otimes_R h \oplus_R (\mathbf{1}_R \ominus_R \Psi) \otimes_R h$
proof −
from *psi-prop* **have** $\Psi = (\mathbf{1}_R \ominus_R h \otimes_R \Phi \otimes_R \delta)$ **by** *simp*
with *psi-in-R* **and** *h-in-R* **and** *phi-in-R* **and** *pert-in-R* **have** $\mathbf{1}_R \ominus_R \Psi = \mathbf{1}_R \ominus_R (\mathbf{1}_R \ominus_R h \otimes_R \Phi \otimes_R \delta)$ **by** *algebra*
with *phi-in-R* **and** *h-in-R* **and** *pert-in-R* **have** $\mathbf{1}_R \ominus_R \Psi = h \otimes_R \Phi \otimes_R \delta$ **by** *algebra*
with *diff-in-R* **and** *pert-in-R* **and** *h-in-R* **and** *psi-in-R* **and** *phi-in-R* **and** *hdh-h* **and** *h-nil* **and** *R.m-assoc* [*of* Ψ *h h*]
show *?thesis* **by** *algebra*
qed
also from *psi-h-h-phi* **have** $\ldots = (\Psi \otimes_R h) \otimes_R \mathit{differ} \otimes_R h \oplus_R (\mathbf{1}_R \ominus_R \Psi) \otimes_R h$ **by** *algebra*
also from *phi-in-R* **and** *h-in-R* **and** *diff-in-R* **have** $\ldots = \Psi \otimes_R (h \otimes_R \mathit{differ} \otimes_R h) \oplus_R (\mathbf{1}_R \ominus_R \Psi) \otimes_R h$ **by** (*simp add*: *R.m-assoc*)
also from *hdh-h* **have** $\ldots = \Psi \otimes_R h \oplus_R (\mathbf{1}_R \ominus_R \Psi) \otimes_R h$ **by** *simp*
also from *psi-in-R* **and** *h-in-R* **and** *R.l-one* [*OF h-in-R*] **have** $\ldots = h$ **by** *algebra*
finally show $p' \otimes_R h = h$ **by** *simp*
qed
qed

Another rewriting step that will be later used

lemma (**in** *lemma-2-2-14*) *ph-h*[*simp*]: **shows** $p \otimes_R h = h$ **and** $h \otimes_R p = h$
proof −
from *p-def* **have** $p \otimes_R h = ((\mathit{differ} \otimes_R h \oplus_R h \otimes_R \mathit{differ}) \otimes_R h)$ **by** *simp*
also from *diff-in-R* **and** *h-in-R* **and** *hdh-h* **and** *h-nil* **have** $\ldots = h$ **by** *algebra*
finally show $p \otimes_R h = h$ **by** *simp*
next
from *p-def* **have** $h \otimes_R p = (h \otimes_R (\mathit{differ} \otimes_R h \oplus_R h \otimes_R \mathit{differ}))$ **by** *simp*
also from *diff-in-R* **and** *h-in-R* **have** $\ldots = h \otimes_R (\mathit{differ} \otimes_R h) \oplus_R h \otimes_R (h \otimes_R \mathit{differ})$ **by** *algebra*
also from *diff-in-R* **and** *h-in-R* **have** $\ldots = h \otimes_R \mathit{differ} \otimes_R h \oplus_R h \otimes_R h \otimes_R \mathit{differ}$ **by** *algebra*

also from *h-in-R* **and** *diff-in-R* **and** *hdh-h* **and** *h-nil* **have** ... = h **by** *algebra*
finally show $h \otimes_R p = h$ **by** *simp*
qed

8.2 Proposition 2.2.16

The following lemma corresponds to the *Proposition 2.2.16* as stated in Aransay's memoir

The previous lemmas $h \otimes_R p' = h$

$p' \otimes_R h = h$ and $p \otimes_R h = h$

$h \otimes_R p = h$ are now used

lemma (**in** *proposition-2-2-16*) *proposition-2-2-16*[*simp*]:
shows *h-π'*: $h \otimes_R \pi' = \mathbf{0}_R$ **and** *π'-h*: $\pi' \otimes_R h = \mathbf{0}_R$ **and** *π-h'*: $\pi \otimes_R h' = \mathbf{0}_R$ **and** *h'-π*: $h' \otimes_R \pi = \mathbf{0}_R$
proof −
from *π'-def* **and** *h-in-R* **and** *p'-in-R* **and** *R.r-one* [*OF h-in-R*] **and** *hp'-h* **show** $h \otimes_R \pi' = \mathbf{0}_R$ **by** *algebra*
next
from *π'-def* **and** *h-in-R* **and** *p'-in-R* **and** *R.l-one* [*OF h-in-R*] **and** *hp'-h* **show** $\pi' \otimes_R h = \mathbf{0}_R$ **by** *algebra*
next
from *π-def* **and** *h'-def* **have** $\pi \otimes_R h' = (\mathbf{1}_R \ominus_R p) \otimes_R (h \otimes_R \Phi)$ **by** *simp*
also from *p-in-R* **and** *h-in-R* **and** *phi-in-R* **have** ... $= (\mathbf{1}_R \ominus_R p) \otimes_R h \otimes_R \Phi$ **by** *algebra*
also from *ph-h* **and** *p-in-R* **and** *h-in-R* **and** *phi-in-R* **and** *R.l-one* [*OF h-in-R*] **have** ... $= \mathbf{0}_R$ **by** *algebra*
finally show $\pi \otimes_R h' = \mathbf{0}_R$ **by** *simp*
next
from *π-def* **and** *h'-def* **have** $h' \otimes_R \pi = (h \otimes_R \Phi) \otimes_R (\mathbf{1}_R \ominus_R p)$ **by** *simp*
also from *psi-h-h-phi* **have** ... $= (\Psi \otimes_R h) \otimes_R (\mathbf{1}_R \ominus_R p)$ **by** *simp*
also from *psi-in-R* **and** *p-in-R* **and** *h-in-R* **have** ... $= \Psi \otimes_R (h \otimes_R (\mathbf{1}_R \ominus_R p))$ **by** *algebra*
also from *p-in-R* **and** *h-in-R* **and** *psi-in-R* **and** *ph-h* **and** *R.r-one* [*OF h-in-R*] **have** ... $= \mathbf{0}_R$ **by** *algebra*
finally show $h' \otimes_R \pi = \mathbf{0}_R$ **by** *simp*
qed

lemma (**in** *proposition-2-2-16*) *p'-projector*: **shows** $p' \otimes_R p' = p'$

proof −
have $p' \otimes_R p' = (\mathit{diff}' \otimes_R h' \oplus_R h' \otimes_R \mathit{diff}') \otimes_R (\mathit{diff}' \otimes_R h' \oplus_R h' \otimes_R \mathit{diff}')$ **unfolding** *p'-def diff'-def* **by** *simp*
also have ... $= (\mathit{diff}' \otimes_R h' \oplus_R h' \otimes_R \mathit{diff}')$ (**is** - = *?d'h'* $\oplus_R$ *?h'd'*)

proof (*rule ring.idemp-prod*)
from *prems* **show** *ring R* **unfolding** *proposition-2-2-16-def lemma-2-2-15-def lemma-2-2-14-def* [*of D R h*] *ring-endomorphisms-def* **by** *simp*

from *diff'-in-R* **and** *h'-in-R* **show** *?d'h'* ∈ *carrier R* **by** *simp*
from *diff'-in-R* **and** *h'-in-R* **show** *?h'd'* ∈ *carrier R* **by** *simp*
from *diff'-in-R h'-in-R* **and** *h'd'h'-h'* **and** *R.m-assoc* [*of diff' h' ?d'h'*] **show** *?d'h'* $\otimes_R$ *?d'h'* = *?d'h'* **by** *algebra*
from *diff'-in-R h'-in-R* **and** *h'd'h'-h'* **and** *sym* [*OF R.m-assoc* [*of diff' h' diff'*]] **and** *sym* [*OF R.m-assoc* [*of h' ?d'h' diff'*]]
show *?h'd'* $\otimes_R$ *?h'd'* = *?h'd'* **by** *algebra*
from *diff'-in-R h'-in-R* **and** *h'-nil* **and** *R.m-assoc* [*of diff' h' ?h'd'*] **and** *sym* [*OF R.m-assoc* [*of h' h' diff'*]]
show *?d'h'* $\otimes_R$ *?h'd'* = $\mathbf{0}_R$ **by** *algebra*
from *diff'-in-R h'-in-R* **and** *R.m-assoc* [*of h' diff' ?d'h'*] **and** *sym* [*OF R.m-assoc* [*of diff' diff' h'*]]
and *ring-endomorphisms.diff-nilpot* [*of D' R*] **and** *ring-endomorphisms-pert* [*OF delta-pert*]
show *?h'd'* $\otimes_R$ *?d'h'* = $\mathbf{0}_R$ **unfolding** *D'-def diff'-def* **by** *simp*
qed
also have *diff'* $\otimes_R$ *h'* $\oplus_R$ *h'* $\otimes_R$ *diff'* = *p'* **unfolding** *p'-def diff'-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

The following lemmas *π-projector* and *π'-projector* correspond to one of the parts of the proof of Lemma 2.2.17, as stated in the memoir; here they have been extracted as independent results, because later they will be used to get some other results

lemma (**in** *proposition-2-2-16*) *π-in-R* [*simp*]: **shows** π ∈ *carrier R* **using** *minus-closed* [*OF R.one-closed p-in-R*] **and** *π-def* **and** *ring-R* **by** *simp*

lemma (**in** *proposition-2-2-16*) *π'-in-R* [*simp*]: **shows** π' ∈ *carrier R* **using** *minus-closed* [*OF R.one-closed p'-in-R*] **and** *π'-def* **and** *ring-R* **by** *simp*

lemma (**in** *proposition-2-2-16*) *π-projector*: **shows** π $\otimes_R$ π = π
proof −
from *π-def* **and** *p-in-R* **and** *minus-closed* [*OF R.one-closed p-in-R*] **and** *r-distr* [*of* $\mathbf{1}_R \ominus_R$ *p* $\mathbf{1}_R \ominus_R$ *p*] **and** *a-inv-closed* [*OF p-in-R*]
and *R.r-one* [*OF minus-closed* [*OF R.one-closed p-in-R*]]
have π $\otimes_R$ π = ($\mathbf{1}_R \ominus_R$ *p*) $\ominus_R$ ($\mathbf{1}_R \ominus_R$ *p*) $\otimes_R$ *p* **by** *algebra*
also from *p-in-R* **and** *minus-closed* [*OF R.one-closed p-in-R*] **and** *p-projector* **and** *l-distr* [*of* $\mathbf{1}_R \ominus_R$ *p p*] **and** *a-inv-closed* [*OF p-in-R*]
and *R.l-one* [*OF p-in-R*] **have** . . . = ($\mathbf{1}_R \ominus_R$ *p*) $\ominus_R$ *p* $\oplus_R$ *p* **by** *algebra*
also from *π-def* **and** *p-in-R* **have** . . . = π **by** *algebra*
finally show *?thesis* **by** *simp*
qed

lemma (**in** *proposition-2-2-16*) *π'-projector*: **shows** π' $\otimes_R$ π' = π'
proof −
from *π'-def* **and** *p'-in-R* **and** *minus-closed* [*OF R.one-closed p'-in-R*] **and** *r-distr* [*of* $\mathbf{1}_R \ominus_R$ *p'* $\mathbf{1}_R \ominus_R$ *p'*]
and *a-inv-closed* [*OF p'-in-R*] **and** *R.r-one* [*OF minus-closed* [*OF R.one-closed p'-in-R*]]

have $\pi' \otimes_R \pi' = (\mathbf{1}_R \ominus_R p') \ominus_R (\mathbf{1}_R \ominus_R p') \otimes_R p'$ **by** *algebra*
also from *p′-in-R* **and** *minus-closed* [*OF R.one-closed p′-in-R*] **and** *p′-projector* **and** *l-distr* [*of* $\mathbf{1}_R \ominus_R p'$ p']
and *a-inv-closed* [*OF p′-in-R*] **and** *R.l-one* [*OF p′-in-R*] **have** $\ldots = (\mathbf{1}_R \ominus_R p') \ominus_R p' \oplus_R p'$ **by** *algebra*
also from *π′-def* **and** *p′-in-R* **have** $\ldots = \pi'$ **by** *algebra*
finally show *?thesis* **by** *simp*
qed

8.3 Lemma 2.2.17

Lemma 2.2.17 proves the existence of an isomorphism between the differential subgroups *diff-group-im-π* and *diff-group-im-π′*

The isomorphism will be explicitly given

Lemma *im-π-ker-p* corresponds to the first part of the proof of Lemma 2.2.17 in Aransay's memoir; in this part, we prove both *im* π' = *kernel* D' D' p', where D' is the differential group perturbed, i.e., D' = (|*carrier* = *carrier* D, *mult* = *op* ⊗, *one* = **1**, *diff* = *differ* $\oplus_R$ δ|), and also *im* π = *kernel D D p*

The reason to prove these equalities between sets is that later, it will be easier to prove the existence of an isomorphism between *diff-group-im-π* and *diff-group-im-π′* than between the kernel sets

The two following proofs in lemma *im-π-ker-p* are quite similar, but maybe trying to extract the common parts and obtaining both goals just by instantiation of the obtained common lemma would have been even, at least, longer

lemma (**in** *proposition-2-2-16*) *im-π-ker-p*: **shows** *image* π (*carrier D*) = *kernel D D p* **and** *image* π′ (*carrier D′*) = *kernel D′ D′ p′*
proof −
show *image* π (*carrier D*) = *kernel D D p*
proof (*intro equalityI*)
show π ‘ *carrier D* ≤ *kernel D D p*
proof
fix *x*
assume *x*: *x* ∈ π ‘ *carrier D* **then obtain** *y* **where** *y*: *y* ∈ *carrier D* **and** *π-y*: π (*y*) = *x* **by** *auto*
show *x* ∈ *kernel D D p*
proof (*unfold kernel-def*, *simp*, *intro conjI*)
from *π-in-R* **and** *ring-R* **have** π ∈ *hom-completion D D* **by** *simp*
with *hom-completion-closed* [*OF - y*, *of* π *D*] **and** *π-y* **show** *x* ∈ *carrier D* **by** *simp*
next

from *π-y* **have** *p x* = *p* (π *y*) **by** *simp*

also have ... = $p\ (p\ (\pi\ y) \otimes \pi\ (\pi\ y))$
proof −
from *π-def* **and** *p-in-R* **have** $\mathbf{1}_R = p \oplus_R \pi$ **by** *algebra*
then have $\mathbf{1}_R\ (\pi\ y) = (p \oplus_R \pi)\ (\pi\ y)$ **by** *simp*
with y **and** *hom-completion-closed* [*OF - y*, *of* π *D*] **and** *ring-R* **and** *π-in-R* **have** $\pi\ y = p\ (\pi\ y) \otimes \pi\ (\pi\ y)$ **by** *simp*
then show *?thesis* **by** *simp*
qed
also from *ring-R* **and** *y* **and** *p-in-R π-in-R* **and** *hom-completion-mult* [*of p D D p* (*π y*) *π* (*π y*)]
and *hom-completion-closed* [*of π D D y*] **and** *hom-completion-closed* [*of p D D π y*] **and** *hom-completion-closed* [*of π D D π y*]
have ... = $p\ (p\ (\pi\ y)) \otimes p\ (\pi\ (\pi\ y))$ **by** *simp*
also from *ring-R* **and** *p-projector* **and** *π-projector* **have** ... = $p\ (\pi\ y) \otimes p\ (\pi\ y)$ **by** (*simp add*: *expand-fun-eq*)
also have ... = $\mathbf{1}_D$
proof −
from *π-def* **and** *p-in-R* **and** *p-projector* **and** *R.r-one* [*OF p-in-R*] **have** $p \otimes_R \pi = \mathbf{0}_R$ **by** *algebra*
with *ring-R* **and** *y* **have** $p\ (\pi\ y) = \mathbf{1}_D$ **by** (*simp add*: *expand-fun-eq*)
then show *?thesis* **by** *simp*
qed
finally show $p\ x = \mathbf{1}_D$ **by** *simp*
qed
qed
show *kernel D D p* ≤ *π ‘ carrier D*
proof (*unfold image-def*, *auto*)
fix *x* **assume** *x*: *x* ∈ *kernel D D p* **then have** *x-in-D*: *x* ∈ *carrier D* **unfolding** *kernel-def* **by** *simp*
from *π-def* **and** *p-in-R* **have** $\mathbf{1}_R = \pi \oplus_R p$ **by** *algebra*
then have $\mathbf{1}_R\ x = (\pi \oplus_R p)\ x$ **by** *simp*
with *x-in-D* **and** *ring-R* **have** $x = \pi\ x \otimes_D p\ x$ **by** (*simp add*: *expand-fun-eq*)
with *ring-R π-in-R* **and** *hom-completion-closed* [*of π D D x*] **and** *x* **and** *D.r-one* [*of π x*] **have** $x = \pi\ x$ **unfolding** *kernel-def* **by** *simp*
with *x-in-D* **show** ∃ *y*∈*carrier D*. *x* = *π y* **by** *auto*
qed
qed
next
show *π′ ‘ carrier D′* = *kernel D′ D′ p′*
proof (*intro equalityI*)
show *π′ ‘ carrier D′* ≤ *kernel D′ D′ p′*
proof
fix *x*
assume *x*: *x* ∈ *π′ ‘ carrier D′* **then obtain** *y* **where** *y*: *y* ∈ *carrier D′* **and** *π′-y*: *π′* (*y*) = *x* **by** *auto*
show *x* ∈ *kernel D′ D′ p′*
proof (*unfold D′-def kernel-def*, *auto*)
from *π′-in-R* **and** *ring-R* **have** *π′* ∈ *hom-completion D D* **by** *simp*
with *y* **and** *hom-completion-closed* [*of π′ D D y*] **and** *π′-y* **and** *D′-def*

show $x \in$ *carrier D* **by** *simp*
next
from π'-*y* **have** $p'\ x = p'\ (\pi'\ y)$ **by** *simp*
also have $\ldots = p'\ (p'\ (\pi'\ y) \otimes \pi'\ (\pi'\ y))$
proof $-$
from π'-*def* **and** p'-*in-R* **have** $\mathbf{1}_R = p' \oplus_R \pi'$ **by** *algebra*
then have $\mathbf{1}_R\ (\pi'\ y) = (p' \oplus_R \pi')\ (\pi'\ y)$ **by** *simp*
with *y* **and** D'-*def* **and** *hom-completion-closed* [*of* π' *D D y*] **and** *ring-R* **and** π'-*in-R* **have** $\pi'\ y = p'\ (\pi'\ y) \otimes \pi'\ (\pi'\ y)$ **by** *simp*
then show *?thesis* **by** *simp*
qed
also from *ring-R* **and** *y* **and** D'-*def* **and** p'-*in-R* π'-*in-R* **and** *hom-completion-mult* [*of* p' *D D* $p'\ (\pi'\ y)\ \ \pi'\ (\pi'\ y)$]
and *hom-completion-closed* [*of* π' *D D y*] **and** *hom-completion-closed* [*of* p' *D D* π' *y*]
and *hom-completion-closed* [*of* π' *D D* π' *y*]
have $\ldots = p'\ (p'\ (\pi'\ y)) \otimes p'\ (\pi'\ (\pi'\ y))$ **by** *simp*
also from *ring-R* **and** p'-*projector* **and** π'-*projector* **have** $\ldots = p'\ (\pi'\ y) \otimes p'\ (\pi'\ y)$ **by** (*simp add*: *expand-fun-eq*)
also have $\ldots = \mathbf{1}_D$
proof $-$
from π'-*def* **and** p'-*in-R* **and** p'-*projector* **and** *R.r-one* [*OF* p'-*in-R*] **have** $p' \otimes_R \pi' = \mathbf{0}_R$ **by** *algebra*
with *ring-R* **and** *y* **and** D'-*def* **have** $p'\ (\pi'\ y) = \mathbf{1}_D$ **by** (*simp add*: *expand-fun-eq*)
then show *?thesis* **by** *simp*
qed
finally show $p'\ x = \mathbf{1}_D$ **by** *simp*
qed
qed
show *kernel* $D'\ D'\ p' \leq \pi'$ ‘ *carrier* D'
proof (*unfold image-def*, *auto*)
fix *x* **assume** *x*: $x \in$ *kernel* $D'\ D'\ p'$ **then have** *x-in-D*: $x \in$ *carrier D* **unfolding** *kernel-def* D'-*def* **by** *simp*
from π'-*def* **and** p'-*in-R* **have** $\mathbf{1}_R = \pi' \oplus_R p'$ **by** *algebra*
then have $\mathbf{1}_R\ x = (\pi' \oplus_R p')\ x$ **by** *simp*
with *x-in-D* **and** *ring-R* **have** $x = \pi'\ x \otimes_D p'\ x$ **by** (*simp add*: *expand-fun-eq*)
with *ring-R* **and** π'-*in-R* **and** *hom-completion-closed* [*of* π' *D D x*] **and** *x* **and** *D.r-one* [*of* π' *x*] **have** $x = \pi'\ x$
unfolding *kernel-def* D'-*def* **by** *simp*
with *x-in-D* **show** $\exists\, y \in$*carrier* D'. $x = \pi'\ y$ **unfolding** D'-*def* **by** *auto*
qed
qed
qed

The following definition is similar to the one of isomorphism given in Isabelle, but here we add the premise that the homomorphism has to be also a completion. This is mainly to keep the coherence with the previous work

constdefs

iso-compl :: - => - => ('*a* => '*b*) *set* (**infixr** $\cong_{compl}$ *60*)
$D \cong_{compl} C$ == {*h*. *h* ∈ *hom-completion D C* & *bij-betw h* (*carrier D*) (*carrier C*)}

The following is an introduction lemma for isomorphisms between groups; maybe it could be introduced in the *Group.thy* file, avoiding the premise on completions!!

lemma *iso-complI*: **assumes** *closed*: $\bigwedge x.\ x \in carrier\ D \Longrightarrow h\ x \in carrier\ C$
and *mult*: $\bigwedge x\ y.\ [\![x \in carrier\ D;\ y \in carrier\ D]\!] \Longrightarrow h\ (x \otimes_D y) = h\ x \otimes_C h\ y$
and *complect*: $\exists\, g.\ h = (\lambda x.\ if\ x \in carrier\ D\ then\ g\ x\ else\ \mathbf{1}_C)$
and *inj-on*: $\bigwedge x\ y.\ [\![\ x \in carrier\ D;\ \ y \in carrier\ D;\ \ h\ (x) = h\ (y)\ ]\!] \Longrightarrow x{=}y$
and *image*: $\bigwedge y.\ y{\in}carrier\ C \Longrightarrow \exists\, x \in carrier\ D.\ y = h\ (x)$
shows $h \in D \cong_{compl} C$
using *prems*
unfolding *iso-compl-def* **unfolding** *hom-diff-def* **apply** (*simp add*: *expand-fun-eq*)
unfolding *hom-completion-def* **apply** *simp*
unfolding *completion-fun2-def completion-def* **apply** (*simp add*: *expand-fun-eq*)
unfolding *hom-def* **unfolding** *Pi-def* **apply** *simp*
unfolding *bij-betw-def inj-on-def* **apply** *simp*
unfolding *image-def* **by** *auto*

Lemmas $\pi\pi'\pi$-π and $\pi'\pi\pi'$-π have been also extracted from the proof of *Lemma 2.2.17* as stated in the memoir

They are used in order to prove injectivity and surjection of π and π'

lemma (**in** *proposition-2-2-16*) $\pi\pi'\pi$-π: **shows** $\pi \otimes_R \pi' \otimes_R \pi = \pi$
proof −
from π'-*def* **have** $\pi \otimes_R \pi' \otimes_R \pi = \pi \otimes_R (\mathbf{1}_R \ominus_R p') \otimes_R \pi$ **by** *simp*
also from π'-*in-R* π-*in-R* p'-*in-R* *R.r-one* [*OF* π-*in-R*] **have** ... $= (\pi \ominus_R \pi \otimes_R p') \otimes_R \pi$ **by** *algebra*
also from π'-*in-R* π-*in-R* p'-*in-R* **and** π-*projector* **have** ... $= \pi \ominus_R \pi \otimes_R\ p' \otimes_R \pi$ **by** *algebra*
also from p'-*def* **have** ... $= \pi \ominus_R \pi \otimes_R ((differ \oplus_R \delta) \otimes_R h' \oplus_R h' \otimes_R (differ \oplus_R \delta)) \otimes_R \pi$
(**is** - $= \pi \ominus_R \pi \otimes_R (?diff' \otimes_R h' \oplus_R h' \otimes_R ?diff') \otimes_R \pi$) **by** *simp*
also from π-*in-R* **and** *diff-pert-in-R* [*OF delta-pert*] **and** h'-*in-R* **and** *sym* [*OF R.m-assoc* [*of* π h' *?diff'*]]
have ... $= \pi \ominus_R (\pi \otimes_R (?diff' \otimes_R h') \oplus_R \pi \otimes_R h' \otimes_R ?diff') \otimes_R \pi$ **by** *algebra*
also from *proposition-2-2-16* **and** π-*in-R* **and** *diff-pert-in-R* [*OF delta-pert*] **and** h'-*in-R*
have ... $= \pi \ominus_R (\pi \otimes_R (?diff' \otimes_R h')) \otimes_R \pi$ **by** *algebra*
also from π-*in-R* **and** *diff-pert-in-R* [*OF delta-pert*] **and** h'-*in-R* **and** *R.m-assoc* [*of* π $?diff' \otimes_R h'$ π] **and** *R.m-assoc* [*of* *?diff'* h' π]
and *proposition-2-2-16* **have** ... $= \pi$ **by** *algebra*
finally show $\pi \otimes_R \pi' \otimes_R \pi = \pi$ **by** *simp*
qed

lemma (**in** *proposition-2-2-16*) $\pi'\pi\pi'$-π': **shows** $\pi' \otimes_R \pi \otimes_R \pi' = \pi'$
proof −

from *π-def* **have** $\pi' \otimes_R \pi \otimes_R \pi' = \pi' \otimes_R (\mathbf{1}_R \ominus_R p) \otimes_R \pi'$ **by** *simp*
also from *π′-in-R π-in-R p-in-R R.r-one [OF π′-in-R]* **have** $\ldots = (\pi' \ominus_R \pi' \otimes_R p) \otimes_R \pi'$ **by** *algebra*
also from *π′-in-R π-in-R p-in-R* **and** *π′-projector* **have** $\ldots = \pi' \ominus_R \pi' \otimes_R p \otimes_R \pi'$ **by** *algebra*
also from *p-def* **have** $\ldots = \pi' \ominus_R \pi' \otimes_R (\mathit{differ} \otimes_R h \oplus_R h \otimes_R \mathit{differ}) \otimes_R \pi'$ **by** *simp*
also from *π′-in-R* **and** *diff-in-R* **and** *h-in-R* **and** *sym [OF R.m-assoc [of π′ h differ]]*
have $\ldots = \pi' \ominus_R (\pi' \otimes_R (\mathit{differ} \otimes_R h) \oplus_R \pi' \otimes_R h \otimes_R \mathit{differ}) \otimes_R \pi'$ **by** *algebra*
also from *proposition-2-2-16* **and** *π′-in-R* **and** *diff-in-R* **and** *h-in-R* **have** $\ldots = \pi' \ominus_R (\pi' \otimes_R (\mathit{differ} \otimes_R h)) \otimes_R \pi'$ **by** *algebra*
also from *π′-in-R* **and** *diff-in-R* **and** *h-in-R* **and** *R.m-assoc [of π′ differ* $\otimes_R$ *h π′]* **and** *R.m-assoc [of differ h π′]* **and** *proposition-2-2-16*
have $\ldots = \pi'$ **by** *algebra*
finally show $\pi' \otimes_R \pi \otimes_R \pi' = \pi'$ **by** *simp*
qed

The following locale definition only introduces some new definitions of constants; they will improve the presentation of the results

locale *lemma-2-2-17 = proposition-2-2-16*

context *lemma-2-2-17*
begin

definition *im-π* **where** *im-π == image π (carrier D)*

definition *im-π′* **where** *im-π′ == image π′ (carrier D′)*

definition *diff-group-im-π* **where** *diff-group-im-π == (|carrier = image π (carrier D), mult = mult D, one = one D,*
diff = completion (|carrier = image π (carrier D), mult = mult D, one = one D, diff = diff D|) D (diff D)|)

definition *diff-group-im-π′* **where** *diff-group-im-π′ == (|carrier = image π′ (carrier D′), mult = mult D, one = one D,*
diff = completion (|carrier = image π′ (carrier D′), mult = mult D, one = one D, diff = (differ $\oplus_R$ *δ)|)*
(|carrier = carrier D, mult = mult D, one = one D, diff = (differ $\oplus_R$ *δ)|) (differ* $\oplus_R$ *δ)|)*

definition *diff-im-π-def*: *diff-im-π == completion (|carrier = image π (carrier D), mult = mult D, one = one D, diff = diff D|) D (diff D)*

definition *diff-im-π′-def*: *diff-im-π′ == completion (|carrier = image π′ (carrier D′), mult = mult D, one = one D,*
diff =(differ $\oplus_R$ *δ) |) (| carrier = carrier D, mult = mult D, one = one D, diff = differ* $\oplus_R$ *δ|) (differ* $\oplus_R$ *δ)*

definition τ **where** τ == *completion*
(| *carrier* = *image* π (*carrier* D), *mult* = *mult* D, *one* = *one* D,
diff = *completion* (|*carrier* = *image* π (*carrier* D), *mult* = *mult* D, *one* = *one* D, *diff* = *diff* D|) D (*diff* D)|)
(|*carrier* = *image* π' (*carrier* D'), *mult* = *mult* D, *one* = *one* D,
diff = *completion* (|*carrier* = *image* π' (*carrier* D'), *mult* = *mult* D, *one* = *one* D, *diff* = *differ* $\oplus_R$ δ|)
(|*carrier* = *carrier* D, *mult* = *mult* D, *one* = *one* D, *diff* = *differ* $\oplus_R$ δ|) (*differ* $\oplus_R$ δ) |) π'

The following definition of τ' corresponds to the inverse of τ

definition
τ' **where** τ' == *completion*
(|*carrier* = *image* π' (*carrier* D'), *mult* = *mult* D, *one* = *one* D,
diff = *completion* (|*carrier* = *image* π' (*carrier* D'), *mult* = *mult* D, *one* = *one* D, *diff* = *differ* $\oplus_R$ δ|)
(|*carrier* = *carrier* D, *mult* = *mult* D, *one* = *one* D, *diff* = *differ* $\oplus_R$ δ|) (*differ* $\oplus_R$ δ) |)
(| *carrier* = *image* π (*carrier* D), *mult* = *mult* D, *one* = *one* D,
diff = *completion* (|*carrier* = *image* π (*carrier* D), *mult* = *mult* D, *one* = *one* D, *diff* = *diff* D|) D (*diff* D)|) π

end

As with *Lemma 2.2.14*, we divide the proof of *Lemma 2.2.17* in four parts. First we prove that there are two homomorphisms, one in each direction, satisfying that they are isomorphisms. Then, in other two lemmas, we prove that their compositions, also in both directions, are equal to the corresponding identities

lemma (**in** *lemma-2-2-17*) *lemma-2-2-17-first-part*: **shows** $\tau \in$ (*diff-group-im-π* $\cong_{compl}$ *diff-group-im-π′*)
proof (*intro iso-complI*)
fix x **assume** $x \in$ *carrier diff-group-im-π* **then have** x: $x \in \pi$ ‘ (*carrier* D) **unfolding** *diff-group-im-π-def* **by** *simp*
then obtain y **where** y: $y \in$ *carrier* D **and** *π-y*: $\pi\ y = x$ **by** *auto*
with *ring-R* **and** *π-in-R π′-in-R* **and** *hom-completion-closed* [*of* π D D y] *hom-completion-closed* [*of* π' D D π y]
have $\pi'\ x \in \pi'$ ‘ *carrier* D **unfolding** *image-def* **by** *auto*
with x **show** $\tau\ x \in$ *carrier diff-group-im-π′* **unfolding** *diff-group-im-π′-def* **and** *τ-def* **and** *D′-def*
unfolding *completion-def image-def* **by** *auto*
next
fix x y **assume** $x \in$ *carrier diff-group-im-π* **and** $y \in$ *carrier diff-group-im-π*
then have x: $x \in \pi$ ‘ (*carrier* D) **and** y: $y \in \pi$ ‘ (*carrier* D) **unfolding** *diff-group-im-π-def* **by** *simp-all*
then obtain x' y' **where** x': $x' \in$ *carrier* D **and** y': $y' \in$ *carrier* D **and** *π-x′*: $\pi\ x' = x$ **and** *π-y′*: $\pi\ y' = y$ **by** *auto*

with *ring-R* **and** π'*-in-R* **and** π*-in-R* **and** *hom-completion-closed* [*of* π *D D x*′] *hom-completion-closed* [*of* π *D D y*′]
and *D.m-closed* [*of* (π *x*′) (π *y*′)] **and** *hom-completion-mult* [*of* π' *D D x y*] **and** *x y* **and** *sym* [*OF hom-completion-mult* [*of* π *D D x*′ *y*′]]
show τ ($x \otimes_{\textit{diff-group-im-}\pi}$ y) = τ $x \otimes_{\textit{diff-group-im-}\pi'}$ τ y **unfolding** τ*-def* *diff-group-im-*π*-def* *diff-group-im-*π'*-def* *completion-def* **by** *simp*
next
from *exI* [*of* - τ] **show** $\exists$ *g*. τ = (λ*x*. *if* $x \in$ *carrier diff-group-im-*π *then g x else* $\mathbf{1}_{\textit{diff-group-im-}\pi'}$)
unfolding τ*-def* *diff-group-im-*π*-def* *diff-group-im-*π'*-def* *completion-def* **by** *auto*
next
fix *x y* **assume** $x \in$ *carrier diff-group-im-*π **and** $y \in$ *carrier diff-group-im-*π **and** τ*-eq*: τ *x* = τ *y*
then have *x*: $x \in \pi$ ‘ (*carrier D*) **and** *y*: $y \in \pi$ ‘ (*carrier D*) **unfolding** *diff-group-im-*π*-def* **by** *simp-all*
then obtain *x*′ *y*′ **where** *x*′: *x*′ $\in$ *carrier D* **and** *y*′: *y*′ $\in$ *carrier D* **and** π*-x*′: π *x*′ = *x* **and** π*-y*′: π *y*′ = *y* **by** *auto*
with τ*-eq* **and** *x y* **have** π' (π *x*′) = π' (π *y*′) **unfolding** τ*-def* *completion-def* *image-def* **by** *auto*
then have π (π' (π *x*′)) = π (π' (π *y*′)) **by** *simp*
with *ring-R* **and** π*-in-R* **and** π'*-in-R* **and** $\pi\pi'\pi$*-*π **have** π *x*′ = π *y*′ **by** (*simp add*: *expand-fun-eq*)
with π*-x*′ **and** π*-y*′ **show** *x* = *y* **by** *simp*
next
fix *y* **assume** $y \in$ *carrier diff-group-im-*π'
then have $y \in \pi'$ ‘ (*carrier D*′) **unfolding** *diff-group-im-*π'*-def* **by** *simp*
with *D*′*-def* **obtain** *y*′ **where** *y*′: *y*′ $\in$ *carrier D* **and** π'*-y*′: π' *y*′ = *y* **by** *auto*
with $\pi'\pi\pi'$*-*π' **and** *ring-R* **and** π*-in-R* **and** π'*-in-R* **have** π' (π (π' *y*′)) = π' *y*′ **by** (*auto simp add*: *expand-fun-eq*)
with π'*-y*′ **and** *y*′ **and** *ring-R* π*-in-R* π'*-in-R* **and** *hom-completion-closed* [*of* π' *D D y*′] *hom-completion-closed* [*of* π *D D* π' *y*′]
have π' (π *y*) = *y* **and** π $y \in \pi'$ ‘ (*carrier D*) **unfolding** *image-def* **by** *auto*
with *diff-group-im-*π*-def* τ*-def* **show** $\exists x \in$ *carrier diff-group-im-*π. *y* = τ *x* **unfolding** *completion-def* **by** *auto*
qed

lemma-2-2-17-second-part proves that $\tau' \in$ *diff-group-im-*π' $\cong_{\textit{compl}}$ *diff-group-im-*π

lemma (**in** *lemma-2-2-17*) *lemma-2-2-17-second-part*: **shows** $\tau' \in$ (*diff-group-im-*π' $\cong_{\textit{compl}}$ *diff-group-im-*π)
(**is** $\tau' \in$ (*?IM-*π' $\cong_{\textit{compl}}$ *?IM-*π))
proof (*intro iso-complI*)
fix *x* **assume** $x \in$ *carrier ?IM-*π' **with** *diff-group-im-*π'*-def* **have** *x*: $x \in \pi'$ ‘ *carrier D*′ **by** *simp*
with *D*′*-def* **obtain** *y* **where** *y*: $y \in$ *carrier D* **and** π*-y*: π' *y* = *x* **by** *auto*
with *ring-R* **and** π*-in-R* π'*-in-R* **and** *hom-completion-closed* [*of* π' *D D y*] *hom-completion-closed* [*of* π *D D* π' *y*]
have π $x \in \pi$ ‘ *carrier D* **unfolding** *image-def* **by** *auto*
with *x* **show** τ' $x \in$ *carrier diff-group-im-*π **unfolding** *diff-group-im-*π*-def* τ'*-def* **unfolding** *completion-def image-def* **by** *auto*

next
fix $x\ y$ **assume** $x \in$ *carrier ?IM-*π' **and** $y \in$ *carrier ?IM-*π'
then have x: $x \in \pi'$ ‘ (*carrier* D') **and** y: $y \in \pi'$ ‘ (*carrier* D') **unfolding** *diff-group-im-*π'*-def* **by** *simp-all*
then obtain $x'\ y'$ **where** x': $x' \in$ *carrier* D **and** y': $y' \in$ *carrier* D **and** π*-*x': $\pi'\ x' = x$ **and** π*-*y': $\pi'\ y' = y$ **unfolding** D'*-def* **by** *auto*
with *ring-R* **and** π'*-in-R* **and** π*-in-R* **and** *hom-completion-closed* [*of* π' D D x'] *hom-completion-closed* [*of* π' D D y']
and D*.m-closed* [*of* $\pi'\ x'\ \pi'\ y'$] **and** *hom-completion-mult* [*of* π D D x y] **and** $x\ y$ **and** *sym* [*OF hom-completion-mult* [*of* π' D D x' y']]
show $\tau'\ (x \otimes_{?IM\text{-}\pi'} y) = \tau'\ x \otimes_{?IM\text{-}\pi}\ \tau'\ y$ **unfolding** τ'*-def diff-group-im-*π*-def diff-group-im-*π'*-def* D'*-def*
unfolding *completion-def* **by** *simp*
next
from *exI* [*of* - τ'] **show** $\exists\, g.\ \tau' = (\lambda x.$ *if* $x \in$ *carrier diff-group-im-*π' *then* $g\ x$ *else* $\mathbf{1}_{diff\text{-}group\text{-}im\text{-}\pi}$)
unfolding τ'*-def diff-group-im-*π*-def diff-group-im-*π'*-def completion-def* **by** *auto*
next
fix $x\ y$ **assume** $x \in$ *carrier diff-group-im-*π' **and** $y \in$ *carrier diff-group-im-*π' **and** τ'*-eq*: $\tau'\ x = \tau'\ y$
then have x: $x \in \pi'$‘ (*carrier* D') **and** y: $y \in \pi'$‘ (*carrier* D') **unfolding** *diff-group-im-*π'*-def* **by** *simp-all*
then obtain $x'\ y'$ **where** x': $x' \in$ *carrier* D **and** y': $y' \in$ *carrier* D **and** π'*-*x': $\pi'\ x' = x$ **and** π'*-*y': $\pi'\ y' = y$
unfolding D'*-def* **by** *auto*
with τ'*-eq* **and** $x\ y$ **have** $\pi\ (\pi'\ x') = \pi\ (\pi'\ y')$ **unfolding** τ'*-def completion-def image-def* **by** *auto*
then have $\pi'\ (\pi\ (\pi'\ x')) = \pi'\ (\pi\ (\pi'\ y'))$ **by** *simp*
with *ring-R* **and** π*-in-R* **and** π'*-in-R* **and** $\pi'\pi\pi'$*-*π' **have** $\pi'\ x' = \pi'\ y'$ **by** (*simp add*: *expand-fun-eq*)
with π'*-*x' **and** π'*-*y' **show** $x = y$ **by** *simp*
next
fix y **assume** $y \in$ *carrier ?IM-*π
then have $y \in \pi$ ‘ (*carrier* D) **unfolding** *diff-group-im-*π*-def* **by** *simp*
then obtain y' **where** y': $y' \in$ *carrier* D **and** π*-*y': $\pi\ y' = y$ **by** *auto*
with $\pi\pi'\pi$*-*π **and** *ring-R* **and** π*-in-R* **and** π'*-in-R* **have** $\pi\ (\pi'\ (\pi\ y')) = \pi\ y'$ **by** (*auto simp add*: *expand-fun-eq*)
with π*-*y' **and** y' **and** *ring-R* π*-in-R* π'*-in-R* **and** *hom-completion-closed* [*of* π D D y'] *hom-completion-closed* [*of* π' D D π y']
have $\pi\ (\pi'\ y) = y$ **and** $\pi'\ y \in \pi'$‘ (*carrier* D') **unfolding** D'*-def image-def* **by** *auto*
then show $\exists\, x \in$*carrier diff-group-im-*π'. $y = \tau'\ x$ **unfolding** *diff-group-im-*π'*-def* τ'*-def completion-def* **by** *auto*
qed

In *lemma-2-2-17-first-part* and *lemma-2-2-17-second-part* we have proved the isomorphism between *diff-group-im-*π and *diff-group-im-*π'; now, with the help of π ‘ *carrier* D = *kernel* D D p

π' ‘ *carrier* D' = *kernel* D' D' p', where we have proved both that *im* π = *ker p* and also that *im* π' = *ker* p', we prove that *ker p* and *ker* p' are also isomorphic. Then we obtain the statement as it is presented in *Lemma 2.2.17* in Aransay's memoir

lemma (**in** *lemma-2-2-17*) *lemma-2-2-17-kernel*: **shows** $\tau \in$ (*diff-group-ker-p* $\cong_{compl}$ *diff-group-ker-p*$'$)
and $\tau' \in$ (*diff-group-ker-p*$'$ $\cong_{compl}$ *diff-group-ker-p*)
using *im-*π*-ker-p lemma-2-2-17-first-part lemma-2-2-17-second-part*
unfolding *diff-group-ker-p-def diff-group-ker-p*$'$*-def diff-group-im-*π*-def diff-group-im-*π'*-def* D'*-def* **by** *simp-all*

lemma (**in** *lemma-2-2-17*) *lemma-2-2-17-third-part*:
shows $\tau \circ \tau' = (\lambda x.$ *if* $x \in$ *carrier diff-group-im-*π' *then id x else* $\mathbf{1}_{diff\text{-}group\text{-}im\text{-}\pi'}$)
(**is** $\tau \circ \tau'$ = *?id-image-*π')
proof (*rule ext*)
fix *x*
show $(\tau \circ \tau')$ *x* = *?id-image-*π' *x*
proof (*cases* $x \in$ *carrier diff-group-im-*π')
case *True* **then have** *x*: $x \in \pi'$‘(*carrier D*) **unfolding** *diff-group-im-*π'*-def* **and** D'*-def* **by** *simp*
then obtain *y* **where** *y*: $y \in$ *carrier D* **and** π'*-y*: π' *y* = *x* **by** *auto*
with *x* **have** $(\tau \circ \tau')$ *x* = τ (π (π' *y*)) **unfolding** τ'*-def* D'*-def completion-def* **by** *simp*
also from π'*-in-R* **and** *ring-R hom-completion-closed* [*of* π' *D D y*] **and** *y* **and** *imageI* [*of* π' *y carrier D* π]
have ... = π' (π (π' *y*)) **unfolding** τ*-def completion-def* **by** *simp*
also with $\pi'\pi\pi'$*-*π' **and** π*-in-R* **and** π'*-in-R* **and** *ring-R* **have** ... = π' *y* **unfolding** *expand-fun-eq* **by** *simp*
also with π'*-y* **and** *True* **have** ... = *?id-image-*π' *x* **by** *simp*
finally show $(\tau \circ \tau')$ *x* = *?id-image-*π' *x* **by** *simp*
next
case *False* **then have** $(\tau \circ \tau')$ *x* = τ **1** **unfolding** τ'*-def diff-group-im-*π'*-def completion-def* **by** *simp*
also from *group-hom.hom-one* [*of D D* π] **and** π*-in-R* **and** π'*-in-R* **and** *ring-R* **and** *imageI* [*OF D.one-closed*, *of* π]
and *group-hom.hom-one* [*of D D* π'] **and** *D-diff-group* **have** ... = **1**
unfolding *group-hom-def group-hom-axioms-def diff-group-def comm-group-def group-def hom-completion-def completion-def* τ*-def* **by** *simp*
also from *False* **have** ... = *?id-image-*π' *x* **unfolding** *diff-group-im-*π'*-def* **by** *simp*
finally show $(\tau \circ \tau')$ *x* = *?id-image-*π' *x* **by** *simp*
qed
qed

lemma (**in** *lemma-2-2-17*) *lemma-2-2-17-fourth-part*:

shows $\tau' \circ \tau = (\lambda x.\ \textit{if } x \in \textit{carrier diff-group-im-}\pi \textit{ then id } x \textit{ else } \mathbf{1}_{\textit{diff-group-im-}\pi})$
(**is** $\tau' \circ \tau$ = *?id-image-π*)
proof (*rule ext*)
fix *x*
show $(\tau' \circ \tau)\ x$ = *?id-image-π x*
proof (*cases x* $\in$ *carrier diff-group-im-π*)
case *True* **then have** *x*: *x* $\in$ π ‘ (*carrier D*) **unfolding** *diff-group-im-π-def* **by** *simp*
then obtain *y* **where** *y*: *y* $\in$ *carrier D* **and** *π-y*: π *y* = *x* **by** *auto*
with *x* **have** $(\tau' \circ \tau)\ x = \tau'\ (\pi'\ (\pi\ y))$ **unfolding** *τ-def* **by** *simp*
also from *π-in-R* **and** *ring-R hom-completion-closed* [*of* π *D D y*] **and** *y* **and** *imageI* [*of* π *y carrier D* π']
have ... = $\pi\ (\pi'\ (\pi\ y))$ **unfolding** *τ'-def D'-def completion-def* **by** *simp*
also with *$\pi\pi'\pi$-π* **and** *π-in-R* **and** *π'-in-R* **and** *ring-R* **have** ... = π *y* **unfolding** *expand-fun-eq* **by** *simp*
also with *π-y* **and** *True* **have** ... = *?id-image-π x* **by** *simp*
finally show $(\tau' \circ \tau)\ x$ = *?id-image-π x* **by** *simp*
next
case *False* **then have** $(\tau' \circ \tau)\ x = \tau'\ \mathbf{1}$ **unfolding** *τ-def diff-group-im-π-def completion-def* **by** *simp*
also from *group-hom.hom-one* [*of D D* π'] **and** *π'-in-R* **and** *π-in-R* **and** *ring-R* **and** *imageI* [*OF D.one-closed*, *of* π']
and *group-hom.hom-one* [*of D D* π] *D-diff-group* **and** *prems* **have** ... = $\mathbf{1}$
unfolding *τ'-def group-hom-def group-hom-axioms-def diff-group-def comm-group-def group-def hom-completion-def completion-def* **by** *simp*
also from *False* **have** ... = *?id-image-π x* **unfolding** *diff-group-im-π-def* **by** *simp*
finally show $(\tau' \circ \tau)\ x$ = *?id-image-π x* **by** *simp*
qed
qed

In the following lemma, again we transfer the result obtained in $\tau \circ \tau' = (\lambda x.\ \textit{if } x \in \textit{carrier diff-group-im-}\pi' \textit{ then id } x \textit{ else } \mathbf{1}_{\textit{diff-group-im-}\pi'})$ and $\tau' \circ \tau = (\lambda x.\ \textit{if } x \in \textit{carrier diff-group-im-}\pi \textit{ then id } x \textit{ else } \mathbf{1}_{\textit{diff-group-im-}\pi})$ from the image sets to the kernel sets

lemma (**in** *lemma-2-2-17*) *lemma-2-2-17-identities*: **shows** $\tau' \circ \tau = (\lambda x.\ \textit{if } x \in \textit{carrier diff-group-ker-p then id } x \textit{ else } \mathbf{1}_{\textit{diff-group-ker-p}})$
and $\tau \circ \tau' = (\lambda x.\ \textit{if } x \in \textit{carrier diff-group-ker-p}' \textit{ then id } x \textit{ else } \mathbf{1}_{\textit{diff-group-ker-p}'})$
using *lemma-2-2-17-third-part lemma-2-2-17-fourth-part im-π-ker-p*
unfolding *diff-group-ker-p-def diff-group-ker-p$'$-def* **apply** *auto*
unfolding *diff-group-im-π-def diff-group-im-π'-def D$'$-def* **apply** *auto*
by (*auto simp add*: *expand-fun-eq*)

We now define what we consider inverse isomorphisms between differential groups (actually the definition also holds for monoids) by means of homomorphism

The previous definition, *op* $\cong_{\textit{invdiff}}$, defined an isomorphism by means of *differential* homomorphisms

constdefs
iso-inv-compl :: (*'a*, *'c*) *monoid-scheme* => (*'b*, *'d*) *monoid-scheme* => ((*'a* => *'b*) × (*'b* => *'a*)) *set* (**infixr** $\cong_{invcompl}$ *60*)
$D \cong_{invcompl} C$ == {(*f*, *g*). $f \in (D \cong_{compl} C)$ & $g \in (C \cong_{compl} D)$ & ($f \circ g$ = *completion C C id*) & ($g \circ f$ = *completion D D id*)}

lemma *iso-inv-complI*: **assumes** *f*: $f \in (D \cong_{compl} C)$ **and** *g*: $g \in (C \cong_{compl} D)$ **and** *fg-id*: ($f \circ g$ = *completion C C id*)
and *gf-id*: ($g \circ f$ = *completion D D id*) **shows** $(f, g) \in (D \cong_{invcompl} C)$
using *f g fg-id gf-id* **unfolding** *iso-inv-compl-def* **by** *simp*

lemma *iso-inv-diff-impl-iso-inv-compl*: **assumes** *f-g*: $(f, g) \in (D \cong_{invdiff} C)$ **shows** $(f, g) \in (D \cong_{invcompl} C)$
using *f-g* **unfolding** *iso-inv-diff-def iso-diff-def iso-inv-compl-def iso-compl-def hom-diff-def hom-completion-def* **by** *auto*

lemma *iso-inv-compl-iso-compl*: **assumes** *f-f'*: $(f, f') \in (D \cong_{invcompl} C)$ **shows** $f \in (D \cong_{compl} C)$
using *f-f'* **unfolding** *iso-inv-compl-def* **by** *simp*

lemma *iso-inv-compl-iso-compl2*: **assumes** *f-f'*: $(f, f') \in (D \cong_{invcompl} C)$ **shows** $f' \in (C \cong_{compl} D)$
using *f-f'* **unfolding** *iso-inv-compl-def* **by** *simp*

lemma *iso-inv-compl-id*: **assumes** *f-f'*: $(f, f') \in (D \cong_{invcompl} C)$ **shows** $f' \circ f$ = *completion D D id*
using *f-f'* **unfolding** *iso-inv-compl-def* **by** *simp*

lemma *iso-inv-compl-id2*: **assumes** *f-f'*: $(f, f') \in (D \cong_{invcompl} C)$ **shows** $f \circ f'$ = *completion C C id*
using *f-f'* **unfolding** *iso-inv-compl-def* **by** *simp*

lemma (**in** *lemma-2-2-17*) *lemma-2-2-17*: **shows** $(\tau, \tau') \in$ (*diff-group-ker-p* $\cong_{invcompl}$ *diff-group-ker-p'*)
using *lemma-2-2-17-identities* **and** *lemma-2-2-17-kernel* **and** *iso-inv-complI* **unfolding** *completion-def* **by** *auto*

end

9 Lemma 2.2.18 in Aransay's memoir

theory *lemma-2-2-18-local-nilpot*
imports
lemma-2-2-17-local-nilpot
begin

Lemma 2.2.18 is generic, in the sense that the previous definitions and premises from locales *lemma-2-2-11* to *lemma-2-2-17* are not needed. Only the notion of differential groups and isomorphism of abelian groups are introduced.

As far as we are in a generic setting, with homomorphisms instead of endomorphisms, the automation of the ring of endomorphisms is lost, and proofs become a bit more obscure

Composition of completions is again a completion

lemma *hom-completion-comp-closed*: **includes** *group A + group B + group C*
assumes *f*: *f* ∈ *hom-completion A B* **and** *g*: *g* ∈ *hom-completion B C*
shows *g* ∘ *f* ∈ *hom-completion A C*
using *f g*
unfolding *hom-completion-def hom-def Pi-def* **apply** *auto*
unfolding *completion-fun2-def completion-def* **apply** *simp*
apply (*intro exI* [*of - g* ∘ *f*], *auto simp add*: *expand-fun-eq*)
apply (*rule group-hom.hom-one*)
unfolding *group-hom-def group-hom-axioms-def hom-def Pi-def* **by** (*simp add*: *prems*)

lemma *iso-inv-compl-coherent-iso-inv-diff*: **assumes** *fg*: (*f*, *g*) ∈ ($F \cong_{invcompl} G$)
and *f-coherent*: *f* ∘ *diff F* = *diff G* ∘ *f*
and *g-coherent*: *g* ∘ *diff G* = *diff F* ∘ *g* **shows** (*f*, *g*) ∈ ($F \cong_{invdiff} G$)
using *fg* **and** *f-coherent* **and** *g-coherent* **unfolding** *iso-inv-diff-def iso-inv-compl-def iso-diff-def iso-compl-def hom-diff-def* **by** *simp*

9.1 Lemma 2.2.18

The following lemma corresponds to *Lemma 2.2.18* in the memoir

It illustrates quite precisely the difficulties of proving facts about homomorphisms and endomorphisms when we loose the automation supplied in the previous lemmas

The difficulties are due to the neccesity of operating with endomorphisms and homomorphisms between different domains, *A* and *B*

A suitable environment would be the one defined by the ring *End* (*A*), the ring *End* (*B*), the commutative group *hom* (*A*, *B*) and the commutative group *hom* (*A*, *B*), but then the question would be how to supply this structure with any automation

In my opinion, the definition *comm-group ?G* ≡ *comm-monoid ?G* ∧ *group ?G* should be relaxed; in its actual version, when unfolded, the characterization *group G* ∧ *comm-monoid G* is obtained, which unfolded again produces *group-axioms G* ∧ *monoid G* ∧ *comm-monoid-axioms G* ∧ *monoid G*, which is redundant. Two possible solutions would be to define *comm-group G* = *group G* ∧ *comm-monoid-axioms G* or also *comm-group G* = *group-axioms G* ∧ *comm-monoid G*

lemma *lemma-2-2-18*: **assumes** *A*: *diff-group A* **and** *B*: *comm-group B* **and** *F-F'*: $(F, F') \in (A \cong_{invcompl} B)$
shows *diff-group* (|*carrier* = *carrier B*, *mult* = *mult B*, *one* = *one B*, *diff* = $F \circ (diff\ A) \circ F'$|)
(**is** *diff-group ?B'*)
and $(F, F') \in (A \cong_{invdiff}$ (|*carrier* = *carrier B*, *mult* = *mult B*, *one* = *one B*, *diff* = $F \circ (diff\ A) \circ F'$|))
(**is** - $\in A \cong_{invdiff}$ *?B'*)
proof −
show *diff-group ?B'*
proof (*unfold diff-group-def diff-group-axioms-def comm-group-def group-def comm-monoid-def*, *intro conjI*)
from *B* **show** *monoid ?B'* **unfolding** *comm-group-def group-def monoid-def* **by** *simp*
from *B* **show** *monoid ?B'* **unfolding** *comm-group-def group-def monoid-def* **by** *simp*
from *B* **show** *comm-monoid-axioms ?B'* **unfolding** *comm-group-def comm-monoid-def comm-monoid-axioms-def* **by** *simp*
from *B* **show** *group-axioms ?B'* **unfolding** *comm-group-def group-def group-axioms-def Units-def* **by** *simp*
next
from *F-F'* **have** *F'-hom*: $F' \in$ *hom-completion B A* **unfolding** *iso-inv-compl-def iso-compl-def* **by** *simp*
from *diff-group.diff-hom* [*OF A*] **have** *diff*: $differ_A \in$ *hom-completion A A* **by** *simp*
from *hom-completion-comp-closed* [*OF - - - F'-hom diff*] **and** *A* **and** *B* **have** *diff-F'*: $differ_A \circ F' \in$ *hom-completion B A*
unfolding *diff-group-def comm-group-def group-def* **by** *simp*
from *F-F'* **have** *F-hom*: $F \in$ *hom-completion A B* **unfolding** *iso-inv-compl-def iso-compl-def* **by** *simp*
from *hom-completion-comp-closed* [*OF - - - diff-F' F-hom*] **and** *A* **and** *B* **have** $differ_{?B'} \in$ *hom-completion B B*
unfolding *diff-group-def comm-group-def group-def* **by** (*simp add*: *o-assoc*)
then show $differ_{?B'} \in$ *hom-completion ?B' ?B'*
unfolding *hom-completion-def completion-fun2-def completion-def hom-def Pi-def expand-fun-eq* **by** *auto*
next
from *sym* [*OF o-assoc* [*of* $F \circ differ_A\ F'\ F \circ differ_A \circ F'$]] **and** *sym* [*OF o-assoc* [*of F* $differ_A$ *F'*]]
and *o-assoc* [*of F' F* $(differ_A \circ F')$]
have $differ_{?B'} \circ differ_{?B'} = F \circ differ_A \circ ((F' \circ F) \circ (differ_A \circ F'))$ **by** *simp*
also from *iso-inv-compl-id* [*OF F-F'*] **have** $\ldots = F \circ differ_A \circ ((\lambda x.\ if\ x \in carrier\ A\ then\ id\ x\ else\ \mathbf{1}_A) \circ (differ_A \circ F'))$
unfolding *iso-inv-compl-def completion-def* **by** *simp*
also from *o-assoc* [*of* $(\lambda x.\ if\ x \in carrier\ A\ then\ id\ x\ else\ \mathbf{1}_A)$ $differ_A$ *F'*] **have** $\ldots = F \circ differ_A \circ (differ_A \circ F')$
proof −
from *diff-group.diff-hom* [*OF A*] **have** $(\lambda x.\ if\ x \in carrier\ A\ then\ id\ x\ else$

$\mathbf{1}_A$) ∘ (*differ* $_A$) = (*differ* $_A$)
unfolding *hom-completion-def completion-fun2-def completion-def hom-def Pi-def expand-fun-eq* **by** *auto*
with *o-assoc* [*of* (λx. *if* $x \in$ *carrier A then id x else* $\mathbf{1}_A$) *differ* $_A$ *F'*] **show** *?thesis* **by** *simp*
qed
also from *sym* [*OF o-assoc* [*of F differ* $_A$ *differ* $_A$ ∘ *F'*]] **and** *o-assoc* [*of differ* $_A$ *differ* $_A$ *F'*]
have ... = *F* ∘ ((*differ* $_A$ ∘ *differ* $_A$) ∘ *F'*) **by** *simp*
also from *diff-group.diff-nilpot* [*OF A*] **have** ... = *F* ∘ ((λx. $\mathbf{1}_A$) ∘ *F'*) **by** *simp*
also have ... = *F* ∘ (λx. $\mathbf{1}_A$) **by** (*simp add*: *expand-fun-eq*)
also from *F-F' A B* **have** ... = (λx. $\mathbf{1}_B$) **by** (*unfold iso-inv-compl-def iso-compl-def expand-fun-eq*, *auto*)
(*intro hom-completion-one*, *unfold group-def diff-group-def comm-group-def*, *simp-all*)
also have ... = (λx. $\mathbf{1}_{?B'}$) **by** *simp*
finally show *differ* $_{?B'}$ ∘ *differ* $_{?B'}$ = (λx. $\mathbf{1}_{?B'}$) **by** *simp*
qed
next
from *F-F'* **have** *F-F'-iso-compl*: (*F*, *F'*) ∈ *A* $\cong_{invcompl}$ *?B'*
unfolding *iso-inv-compl-def iso-compl-def completion-def expand-fun-eq hom-completion-def hom-def Pi-def completion-fun2-def* **by** *auto*
moreover have *F-coherent*: *F* ∘ *diff A* = *diff ?B'* ∘ *F*
proof −
have *diff A* = *diff A* ∘ *completion A A id*
proof (*rule ext*)
fix *x*
show (*differ* $_A$) *x* = (*differ* $_A$ ∘ *completion A A id*) *x*
proof (*cases x* ∈ *carrier A*)
case *True* **with** *diff-group.diff-hom* [*OF A*] **show** (*differ* $_A$) *x* = (*differ* $_A$ ∘ *completion A A id*) *x*
unfolding *hom-completion-def completion-fun2-def completion-def* **by** *simp*
next
case *False* **with** *diff-group.diff-hom* [*OF A*]
have *l-h-s*: (*differ* $_A$) *x* = $\mathbf{1}_A$ **unfolding** *hom-completion-def completion-fun2-def completion-def* **by** *auto*
from *False* **have** (*completion A A id*) *x* = $\mathbf{1}_A$ **by** (*unfold completion-def*, *simp*)
with *hom-completion-one* [*OF - - diff-group.diff-hom* [*OF A*]] **and** *A* **have** *r-h-s*: (*differ* $_A$ ∘ *completion A A id*) *x* = $\mathbf{1}_A$
unfolding *diff-group-def comm-group-def group-def* **by** *simp*
from *r-h-s* **and** *l-h-s* **show** (*differ* $_A$) *x* = (*differ* $_A$ ∘ *completion A A id*) *x* **by** *simp*
qed
qed
also from *iso-inv-compl-id* [*OF F-F'*] **and** *o-assoc* [*of diff A F' F*] **have** ... = *diff A* ∘ *F'* ∘ *F* **by** *simp*
finally have *diff A* = *diff A* ∘ *F'* ∘ *F* **by** *simp*

with *o-assoc* [*of F diff A* ∘ *F' F*] **and** *o-assoc* [*of F diff A F'*] **show** *?thesis* **by** *simp*
qed
moreover have *F'-coherent*: *F'* ∘ *diff ?B'* = *diff A* ∘ *F'*
proof −
have *diff A* = *completion A A id* ∘ *diff A*
proof (*rule ext*)
fix *x*
show (*differ*$_A$) *x* = (*completion A A id* ∘ *differ*$_A$) *x*
proof (*cases x* ∈ *carrier A*)
case *True* **with** *diff-group.diff-hom* [*OF A*] **and** *diff-group.diff-hom* [*OF A*] **and** *hom-completion-closed* [*of diff A A A x*] **show** (*differ*$_A$) *x* = (*completion A A id* ∘ *differ*$_A$) *x*
unfolding *hom-completion-def completion-fun2-def completion-def* **by** *simp*
next
case *False* **with** *diff-group.diff-hom* [*OF A*] **have** *l-h-s*: (*differ*$_A$) *x* = $\mathbf{1}_A$
unfolding *hom-completion-def completion-fun2-def completion-def* **by** *auto*
with *l-h-s* **have** *r-h-s*: (*completion A A id* ∘ *differ*$_A$) *x* = $\mathbf{1}_A$ **unfolding** *completion-def* **by** *simp*
from *r-h-s* **and** *l-h-s* **show** (*differ*$_A$) *x* = (*completion A A id* ∘ *differ*$_A$) *x* **by** *simp*
qed
qed
also from *iso-inv-compl-id* [*OF F-F'*] **and** *sym* [*OF o-assoc* [*of F' F diff A*]] **have** . . . = *F'* ∘ *F* ∘ *diff A* **by** *simp*
finally have *diff A* = *F'* ∘ *F* ∘ *diff A* **by** *simp*
then show *?thesis* **by** (*auto simp add*: *o-assoc*)
qed
ultimately show (*F*, *F'*) ∈ *A* $\cong_{invdiff}$ *?B'* **by** (*intro iso-inv-compl-coherent-iso-inv-diff*)
qed

end

10 Lemma 2.2.19 in Aransay's memoir

theory *lemma-2-2-19-local-nilpot*
imports
lemma-2-2-18-local-nilpot
begin

Lemma 2.2.19, as well as *Lemma 2.2.18*, is generic in the sense that the previous definitions and premises from locales *lemma-2-2-11* to *lemma-2-2-17* are not needed. Only the definition of reduction is used

lemma (**in** *diff-group*) *diff-group-is-group*: **shows** *group D* **using** *prems* **unfolding** *diff-group-def comm-group-def* **by** *simp*

lemma *hom-diffs-comp-closed*: **includes** *diff-group A* **includes** *diff-group B* **includes** *diff-group C*
assumes *f*: $f \in$ *hom-diff A B* **and** *g*: $g \in$ *hom-diff B C*
shows $g \circ f \in$ *hom-diff A C*
proof (*unfold hom-diff-def hom-completion-def*, *auto*)
from *f* **and** *g* **have** *f-compl*: $f \in$ *completion-fun2 A B* **and** *g-compl*: $g \in$ *completion-fun2 B C*
unfolding *hom-diff-def hom-completion-def* **by** *auto*
with *A.diff-group-is-group B.diff-group-is-group C.diff-group-is-group hom-diff-is-hom-completion* [*OF f*]
and *hom-diff-is-hom-completion* [*OF g*] **and** *hom-completion-one* [*of B C g*] **and** *completion-closed2* [*OF f-compl*]
show $g \circ f \in$ *completion-fun2 A C*
unfolding *completion-fun2-def completion-def expand-fun-eq* **apply** *simp* **apply** (*intro exI* [*of - g* $\circ$ *f*]) **by** *auto*
next
show $g \circ f \in$ *hom A C*
proof (*intro homI*)
fix *x*
assume *x*: $x \in$ *carrier A* **from** *hom-diff-closed* [*OF f x*] **and** *hom-diff-closed* [*OF g*, *of f x*] **show** $(g \circ f)\ x \in$ *carrier C* **by** *simp*
next
fix *x y*
assume *x*: $x \in$ *carrier A* **and** *y*: $y \in$ *carrier A* **from** *f g* **and** *hom-diff-mult* [*OF f x y*] **and** *hom-diff-mult* [*OF g*, *of f x f y*]
and *hom-diff-closed* [*OF f x*] *hom-diff-closed* [*OF f y*] **show** $(g \circ f)\ (x \otimes_A y) = (g \circ f)\ x \otimes_C (g \circ f)\ y$ **by** (*unfold hom-diff-def*, *simp*)
qed
next
from *hom-diff-coherent* [*OF f*] **and** *hom-diff-coherent* [*OF g*] **and** *o-assoc* [*of g f* $differ_A$] **and** *o-assoc* [*of g* $differ_B$ *f*]
and *o-assoc* [*of* $differ_C$ *g f*] **show** $g \circ f \circ differ_A = differ_C \circ (g \circ f)$ **by** *simp*
qed

10.1 Lemma 2.2.19

The following lemma corresponds to *Lemma 2.2.19* as stated in Aransay's memoir

lemma (**in** *reduction*) *lemma-2-2-19*: **assumes** *B*: *diff-group B* **and** *F-F'-isom*: $(F, F') \in (C \cong_{invdiff} B)$
shows *reduction D B* $(F \circ f)\ (g \circ F')\ h$
proof (*intro reductionI*)
from *prems* **show** *diff-group D* **by** (*unfold reduction-def*, *simp*)
from *B* **show** *diff-group B* **by** *simp*
next
from *hom-diffs-comp-closed* [*OF D-diff-group C-diff-group B f-hom-diff*, *of F*] **and** *iso-diff-hom-diff* [*of F C B*]
and *iso-inv-diff-iso-diff* [*OF F-F'-isom*] **show** $F \circ f \in$ *hom-diff D B* **by** *simp*
next

from *hom-diffs-comp-closed* [*OF B C-diff-group D-diff-group - g-hom-diff*, *of F'*]
and *iso-diff-hom-diff* [*of F' B C*]
and *iso-inv-diff-iso-diff2* [*OF F-F'-isom*] **show** $g \circ F' \in$ *hom-diff B D* **by** *simp*
next
from *h-hom-compl* **show** $h \in$ *hom-completion D D* **by** *simp*
next
from *sym* [*OF o-assoc* [*of F f* $(g \circ F')$]] **and** *o-assoc* [*of f g F'*] **and** *fg*
have $F \circ f \circ (g \circ F') = F \circ ((\lambda x.$ *if* $x \in$ *carrier C then id x else* $\mathbf{1}_C) \circ F')$ **by** *simp*
also from *iso-inv-diff-iso-diff2* [*OF F-F'-isom*] **and** *iso-diff-hom-diff* [*of F' B C*] **have** $\ldots = F \circ F'$
unfolding *hom-diff-def hom-completion-def completion-fun2-def completion-def hom-def Pi-def expand-fun-eq* **by** *auto*
also from *iso-inv-diff-id2* [*OF F-F'-isom*] **have** $\ldots = (\lambda x.$ *if* $x \in$ *carrier B then id x else* $\mathbf{1}_B)$ **unfolding** *completion-def* **by** *simp*
finally show $F \circ f \circ (g \circ F') = (\lambda x.$ *if* $x \in$ *carrier B then id x else* $\mathbf{1}_B)$ **by** *simp*
next
from *sym* [*OF o-assoc* [*of g F'* $(F \circ f)$]] **and** *o-assoc* [*of F' F f*] **and** *iso-inv-diff-id* [*OF F-F'-isom*]
have $g \circ F' \circ (F \circ f) = g \circ ((\lambda x.$ *if* $x \in$ *carrier C then id x else* $\mathbf{1}_C) \circ f)$ **unfolding** *completion-def* **by** *simp*
also from *f-hom-diff* **have** $\ldots = g \circ f$
unfolding *hom-diff-def hom-completion-def completion-fun2-def completion-def hom-def Pi-def expand-fun-eq* **by** *auto*
finally have $g \circ F' \circ (F \circ f) = g \circ f$ **by** *simp*
with *gf-dh-hd* **show** $(\lambda x.$ *if* $x \in$ *carrier D then* $(g \circ F' \circ (F \circ f))\ x$
$\otimes$ (*if* $x \in$ *carrier D then* (*differ* $\circ h)\ x \otimes (h \circ$ *differ*) x *else* $\mathbf{1}$) *else* $\mathbf{1}) =$
$(\lambda x.$ *if* $x \in$ *carrier D then id x else* $\mathbf{1})$ **by** (*simp only*: *expand-fun-eq*) *simp*
next
from *fh* **and** *sym* [*OF o-assoc* [*of F f h*]] **have** $F \circ f \circ h = F \circ (\lambda x.$ *if* $x \in$ *carrier D then* $\mathbf{1}_C$ *else* $\mathbf{1}_C)$ **by** *simp*
also from *B* **and** *iso-diff-hom-diff* [*of F C B*] **and** *iso-inv-diff-iso-diff* [*OF F-F'-isom*] **and** *C-diff-group*
have $\ldots = (\lambda x.$ *if* $x \in$ *carrier D then* $\mathbf{1}_B$ *else* $\mathbf{1}_B)$
by (*unfold expand-fun-eq*, *simp*) (*intro hom-completion-one*, *unfold hom-diff-def diff-group-def comm-group-def group-def*, *auto*)
finally show $F \circ f \circ h = (\lambda x.$ *if* $x \in$ *carrier D then* $\mathbf{1}_B$ *else* $\mathbf{1}_B)$ **by** *simp*
next
from *hg* **and** *o-assoc* [*of h g F'*] **have** $h \circ (g \circ F') = (\lambda x.$ *if* $x \in$ *carrier C then* $\mathbf{1}$ *else* $\mathbf{1}) \circ F'$ **by** *simp*
also have $\ldots = (\lambda x.$ *if* $x \in$ *carrier B then* $\mathbf{1}$ *else* $\mathbf{1})$ **by** (*unfold expand-fun-eq*, *simp*)
finally show $h \circ (g \circ F') = (\lambda x.$ *if* $x \in$ *carrier B then* $\mathbf{1}$ *else* $\mathbf{1})$ **by** *simp*
next
from *hh* **show** $h \circ h = (\lambda x.$ *if* $x \in$ *carrier D then* $\mathbf{1}$ *else* $\mathbf{1})$ **by** *simp*
qed

end

11 Proof of the Basic Perturbation Lemma

theory *Basic-Perturbation-Lemma-local-nilpot*
imports
lemma-2-2-19-local-nilpot
begin

In the following locale we define an abbreviation that we will use later in proofs, and we also join the results obtained in locale *lemma-2-2-17* with the ones reached in *lemma-2-2-11*. The combination of both locales give us the set of premises in the Basic Perturbation Lemma (from now on, BPL) statement

locale *BPL* = *lemma-2-2-17* + *lemma-2-2-11*

context *BPL*
begin

definition f' **where** f' == (*completion*
(|*carrier* = *kernel D D p*, *mult* = *op* ⊗, *one* = **1**,
diff = *completion* (|*carrier* = *kernel D D p*, *mult* = *op* ⊗, *one* = **1**, *diff* = *differ*|)
D (*differ*)|) *C f*)

end

lemma (**in** *BPL*) *π-gf*: **shows** $g \circ f = \pi$
proof −
let *?gf* = $g \circ f$

from *g-f-hom-diff* **have** *?gf* ∈ *hom-completion D D* **unfolding** *hom-diff-def* **by** *simp*
with *ring-R* **have** *gf-in-R*: *?gf* ∈ *carrier R* **by** *simp*
from *gf-dh-hd* **and** *ring-R* **have** *?gf* $\oplus_R$ (*differ* $\otimes_R$ *h* $\oplus_R$ *h* $\otimes_R$ *differ*) = $\mathbf{1}_R$ **by** *simp*
then have *?gf* $\oplus_R$ (*differ* $\otimes_R$ *h* $\oplus_R$ *h* $\otimes_R$ *differ*) $\ominus_R$ (*differ* $\otimes_R$ *h* $\oplus_R$ *h* $\otimes_R$ *differ*) = $\mathbf{1}_R$ $\ominus_R$ (*differ* $\otimes_R$ *h* $\oplus_R$ *h* $\otimes_R$ *differ*) **by** *simp*
with *gf-in-R* **and** *R.one-closed* **and** *diff-in-R* **and** *h-in-R* **have** *?gf* = $\mathbf{1}_R$ $\ominus_R$ (*differ* $\otimes_R$ *h* $\oplus_R$ *h* $\otimes_R$ *differ*) **by** *algebra*
with *π-def* **and** *p-def* **and** *ring-R* **show** $g \circ f = \pi$ **by** *simp*
qed

11.1 BPL proof

The following lemma corresponds to the first part of *Lemma 2.2.20* (i.e., the BPL) in Aransay's memoir

The proof of the BPL is divided into two parts, as it is also in Aransay's memoir.

In the first part, proved in *BPL-reduction*, from the given premises, we buid a new reduction from $D' = (\!|$ *carrier* = *carrier* D, ..., *diff* = *differ*$_D$ $\oplus_R$ $\delta|\!)$ into C', where $C' = (\!|$ *carrier* = *carrier* C, ... , *diff* = $f' \circ (\tau' \circ$ *differ*$_{\textit{diff-group-ker-p}'}$ $\circ\ \tau) \circ g|\!)$ $(f' \circ (\tau' \circ (\mathbf{1}_R \ominus_R p')))$

The reduction is given by the triple $f' \circ (\tau' \circ \mathbf{1}_R \ominus_R p')$, *inc-ker-p'* $\circ\ \tau \circ g$, h'

In the second part of the proof of the BPL, here stored in lemma *BPL-simplifications*, the expressions $f' \circ (\tau' \circ \mathbf{1}_R \ominus_R p')$, *inc-ker-p'* $\circ\ \tau \circ g$ and $f' \circ (\tau' \circ$ *differ*$_{\textit{diff-group-ker-p}'}$ $\circ\ \tau) \circ g$ are simplified, obtaining the ones in the BPL statement

By finally joining *BPL-reduction* and *BPL-simplifications*, we complete the proof of the BPL

11.2 Existence of a reduction

lemma (**in** *BPL*) *BPL-reduction*:
 shows *reduction* D'
 $(\!|$ *carrier* = *carrier* C, *mult* = *mult* C, *one* = *one* C, *diff* = $f' \circ (\tau' \circ$ *differ*$_{\textit{diff-group-ker-p}'}$ $\circ\ \tau) \circ g|\!)$ $(f' \circ (\tau' \circ (\mathbf{1}_R \ominus_R p')))$
 (*inc-ker-p'* $\circ\ \tau \circ g$) h'
proof −

 from *lemma-2-2-15* **have** *red-D'-ker-p'*: *reduction* D' *diff-group-ker-p'* $(\mathbf{1}_R \ominus_R p')$ *inc-ker-p'* h' **by** *simp*

 have *iso-inv-diff-ker-p'-ker-p*:
 $(\tau', \tau) \in$ *diff-group-ker-p'* $\cong_{\textit{invdiff}}$ $(\!|$ *carrier* = *kernel* D D p, *mult* = *mult* D, *one* = *one* D, *diff* = $\tau' \circ$ *differ*$_{\textit{diff-group-ker-p}'}$ $\circ\ \tau|\!)$
 (**is** $(\tau', \tau) \in$ *diff-group-ker-p'* $\cong_{\textit{invdiff}}$ *?ker-p-pert*)
 and *diff-group-ker-p-pert*: *diff-group* $(\!|$ *carrier* = *kernel* D D p, *mult* = *mult* D, *one* = *one* D, *diff* = $\tau' \circ$ *differ*$_{\textit{diff-group-ker-p}'}$ $\circ\ \tau|\!)$
 (**is** *diff-group ?ker-p-pert*)
 proof −
 from *lemma-2-2-17* **have** *iso-inv-compl*: $(\tau', \tau) \in$ *diff-group-ker-p'* $\cong_{\textit{invcompl}}$ *diff-group-ker-p* **unfolding** *iso-inv-compl-def* **by** *simp*
 from *lemma-2-2-15* **have** *diff-group-ker-p'*: *diff-group diff-group-ker-p'* **unfolding** *reduction-def diff-group-def* **by** *simp*
 from *lemma-2-2-14* **have** *comm-group-ker-p*: *comm-group diff-group-ker-p* **unfolding** *reduction-def diff-group-def comm-group-def* **by** *simp*
 from *lemma-2-2-18* [*OF diff-group-ker-p' comm-group-ker-p iso-inv-compl*]
 show $(\tau', \tau) \in$ *diff-group-ker-p'* $\cong_{\textit{invdiff}}$ *?ker-p-pert* **and** *diff-group ?ker-p-pert*

unfolding *diff-group-ker-p-def* **by** *simp-all*
qed

from *reduction.lemma-2-2-19* [*OF red-D′-ker-p′ diff-group-ker-p-pert iso-inv-diff-ker-p′-ker-p*]
have *red-D′-ker-p-pert*: *reduction* D' *?ker-p-pert* $(\tau' \circ (\mathbf{1}_R \ominus_R p'))$ (*inc-ker-p′* $\circ$ τ) h' **by** *simp*

have *ker-p-isom-C*: $(f', g) \in$ (*diff-group-ker-p* $\cong_{invdiff}$ C)
proof −
from *iso-inv-diff-rev* [*OF lemma-2-2-11*] **have** *im-gf-isom-C*: (*completion diff-group-im-gf* C f, g) $\in$ (*diff-group-im-gf* $\cong_{invdiff}$ C) **by** *simp*
moreover from π*-gf* **have** $g \circ f = \pi$ **by** *simp*
moreover from *im-*π*-ker-p* **have** *im-*π*-ker-p*: π ‘ *carrier* D = *kernel* D D p **by** *simp*
ultimately show *?thesis* **unfolding** *diff-group-im-gf-def im-gf-def diff-group-ker-p-def* f'*-def* **by** *simp*
qed
then have *ker-p-pert-isom-C*: $(f', g) \in$ (*?ker-p-pert* $\cong_{invcompl}$ C)
unfolding *iso-inv-diff-def iso-inv-compl-def diff-group-ker-p-def iso-diff-def iso-compl-def hom-diff-def hom-completion-def*
hom-def Pi-def completion-fun2-def completion-def **by** (*auto simp add*: *expand-fun-eq*)
from *red-D′-ker-p-pert* **have** *diff-group-ker-p-pert*: *diff-group ?ker-p-pert* **unfolding** *reduction-def diff-group-def* **by** *simp*
from *C-diff-group* **have** C: *comm-group* C **unfolding** *diff-group-def comm-group-def* **by** *simp*
from *lemma-2-2-18* [*OF diff-group-ker-p-pert C ker-p-pert-isom-C*]
have f'*-g-isom*: $(f', g) \in$ (*?ker-p-pert* $\cong_{invdiff}$
(| *carrier* = *carrier* C, *mult* = *op* $\otimes_C$, *one* = $\mathbf{1}_C$, *diff* = $f' \circ$ ($\tau' \circ$ *differ*$_{diff\text{-}group\text{-}ker\text{-}p'}$ $\circ$ τ) $\circ$ g|))
(**is** $(f', g) \in$ (*?ker-p-pert* $\cong_{invdiff}$ *?C′*))
and *diff-group-C-pert*:
diff-group (|*carrier* = *carrier* C, *mult* = *op* $\otimes_C$, *one* = $\mathbf{1}_C$, *diff* = $f' \circ$ ($\tau' \circ$ *differ*$_{diff\text{-}group\text{-}ker\text{-}p'}$ $\circ$ τ) $\circ$ g |)
(**is** *diff-group ?C′*) **by** *simp-all*

from *reduction.lemma-2-2-19* [*OF red-D′-ker-p-pert diff-group-C-pert* f'*-g-isom*]
show *reduction* D' *?C′* $(f' \circ (\tau' \circ \mathbf{1}_R \ominus_R p'))$ (*inc-ker-p′* $\circ$ τ $\circ$ g) h' **by** *simp*
qed

11.3 BPL previous simplifications

In order to prove the simplifications required in the second part of the proof, i.e. lemma *BPL-simplifications*, we first have to prove some results concerning the composition of some of the homomorphisms and endomorphisms we have already introduced.

Therefore, we have the ring R and we prove that it behaves as expected with some homomorphisms from *Hom* (D C) and *Hom* (C D), where the operation to relate them is the composition

We will prove some properties such as distributivity of composition with respect to addition of endomorphisms and the like

The results are stated in generic terms

lemma (**in** *ring-endomorphisms*) *add-dist-comp*: **assumes** C: *diff-group* C **and** g: $g \in$ *hom-completion* C D **and** a: $a \in$ *carrier* R
and b: $b \in$ *carrier* R **shows** $(a \oplus_R b) \circ g = (\lambda x.$ *if* $x \in$ *carrier* C *then* $(a \circ g)$ $x \otimes (b \circ g)$ x *else* $\mathbf{1})$
using *ring-R* **and** g **and** a **and** b **and** *hom-completion-closed* [*OF* g] **and** *completion-closed2* [*of* g C D] **and** *group-hom.hom-one* [*of* D D a]
group-hom.hom-one [*of* D D b] **and** *D-diff-group*
unfolding *hom-completion-def completion-def diff-group-def comm-group-def group-hom-def group-hom-axioms-def*
by (*auto simp add*: *expand-fun-eq*)

lemma (**in** *ring-endomorphisms*) *comp-hom-compl*: **assumes** C: *diff-group* C **and** g: $g \in$ *hom-completion* C D **and** a: $a \in$ *carrier* R
shows $a \circ g = (\lambda x.$ *if* $x \in$ *carrier* C *then* $(a \circ g)$ x *else* $\mathbf{1})$
using *ring-R* **and** g **and** a **and** *hom-completion-closed* [*OF* g] **and** *completion-closed2* [*of* g C D]
and *group-hom.hom-one* [*of* D D a] **and** *D-diff-group*
unfolding *hom-completion-def completion-def diff-group-def comm-group-def group-hom-def group-hom-axioms-def*
by (*auto simp add*: *expand-fun-eq*)

lemma (**in** *ring-endomorphisms*) *one-comp-g*: **assumes** C: *diff-group* C **and** g: $g \in$ *hom-completion* C D
shows $\mathbf{1}_R \circ g = g$
using *ring-R* **and** g **and** *hom-completion-closed* [*OF* g] **and** *completion-closed2* [*of* g C D]
unfolding *hom-completion-def completion-fun2-def completion-def* **apply** *simp*
by (*auto simp add*: *expand-fun-eq*)

lemma (**in** *ring-endomorphisms*) *minus-dist-comp*: **assumes** C: *diff-group* C **and** g: $g \in$ *hom-completion* C D **and** a: $a \in$ *carrier* R
and b: $b \in$ *carrier* R **shows** $(a \ominus_R b) \circ g = (\lambda x.$ *if* $x \in$ *carrier* C *then* $(a \circ g)$ $x \otimes ((\ominus_R b) \circ g)$ x *else* $\mathbf{1})$
using *add-dist-comp* [*OF* C g a *a-inv-closed* [*OF* b]] **unfolding** *a-minus-def* [*OF* a b] **by** *simp*

lemma (**in** *ring-endomorphisms*) *minus-comp-g*: **assumes** C: *diff-group* C **and** g: $g \in$ *hom-completion* C D **and** a: $a \in$ *carrier* R
and b: $b \in$ *carrier* R **and** *a-eq-b*: $a = b$ **shows** $(\ominus_R a) \circ g = (\ominus_R b) \circ g$
proof −
from a **and** b **and** *a-eq-b* **have** $\ominus_R a = \ominus_R b$ **by** *algebra*

then show *?thesis* **by** *simp*
qed

lemma (**in** *ring-endomorphisms*) *minus-comp-g2*: **assumes** *C*: *diff-group C* **and** *g*: *g* ∈ *hom-completion C D* **and** *a*: *a* ∈ *carrier R*
and *b*: *b* ∈ *carrier R* **and** *a-eq-b*: *a* ∘ *g* = *b* ∘ *g* **shows** $(\ominus_R\ a) \circ g = (\ominus_R\ b) \circ g$
using *a-eq-b* **and** *minus-interpret* [*OF a*] *minus-interpret* [*OF b*] **and** *g* **by** (*simp add*: *expand-fun-eq*)

lemma (**in** *ring-endomorphisms*) *l-add-dist-comp*: **includes** *diff-group C* **assumes** *f*: *f* ∈ *hom-completion D C* **and** *a*: *a* ∈ *carrier R*
and *b*: *b* ∈ *carrier R* **shows** $f \circ (a \oplus_R b) = (\lambda x.\ \textit{if}\ x \in \textit{carrier}\ D\ \textit{then}\ (f \circ a)\ x \otimes_C (f \circ b)\ x\ \textit{else}\ \mathbf{1}_C)$
using *prems ring-R* **and** *f* **and** *a* **and** *b* **and** *hom-completion-closed* [*of a D D*] *hom-completion-closed* [*of b D D*]
and *hom-completion-mult* [*of f D C*] *group-hom.hom-one* [*of D C f*] **and** *D.diff-group-is-group C.diff-group-is-group*
unfolding *hom-completion-def* [*of D C*] *group-hom-def group-hom-axioms-def* **by** (*simp add*: *expand-fun-eq*)

lemma (**in** *ring-endomorphisms*) *l-comp-hom-compl*: **assumes** *C*: *diff-group C* **and** *f*: *f* ∈ *hom-completion D C* **and** *a*: *a* ∈ *carrier R*
shows $f \circ a = (\lambda x.\ \textit{if}\ x \in \textit{carrier}\ D\ \textit{then}\ (f \circ a)\ x\ \textit{else}\ \mathbf{1}_C)$
using *ring-R* **and** *f* **and** *a* **and** *hom-completion-closed* [*of a*] **and** *completion-closed2* [*of a D D*] **and** *group-hom.hom-one* [*of D C f*] **and** *C*
and *D-diff-group*
unfolding *diff-group-def comm-group-def hom-completion-def group-hom-def group-hom-axioms-def* **by** (*simp add*: *expand-fun-eq*)

lemma (**in** *ring-endomorphisms*) *l-minus-dist-comp*: **includes** *diff-group C* **assumes** *f*: *f* ∈ *hom-completion D C* **and** *a*: *a* ∈ *carrier R*
and *b*: *b* ∈ *carrier R* **shows** $f \circ (a \ominus_R b) = (\lambda x.\ \textit{if}\ x \in \textit{carrier}\ D\ \textit{then}\ (f \circ a)\ x \otimes_C (f \circ (\ominus_R\ b))\ x\ \textit{else}\ \mathbf{1}_C)$
using *l-add-dist-comp* [*OF - f a a-inv-closed* [*OF b*]] **unfolding** *a-minus-def* [*OF a b*] **by** (*simp add*: *prems*)

lemma (**in** *ring-endomorphisms*) *l-minus-comp-f*: **assumes** *C*: *diff-group C* **and** *f*: *f* ∈ *hom-completion D C* **and** *a*: *a* ∈ *carrier R*
and *b*: *b* ∈ *carrier R* **and** *a-eq-b*: *f* ∘ *a* = *f* ∘ *b* **shows** $f \circ (\ominus_R\ a) = f \circ (\ominus_R\ b)$
proof −
from *HomGroupsCompletion.hom-completion-groups-mult-comm-group* [*of D C*] **and** *C* **and** *D-diff-group*
have *Hom-D-C*: *comm-group* ⦇*carrier* = *hom-completion D C*, *mult* = $\lambda f\ g\ x.\ \textit{if}\ x \in \textit{carrier}\ D\ \textit{then}\ f\ x \otimes_C g\ x\ \textit{else}\ \mathbf{1}_C$,
one = $\lambda x.\ \textit{if}\ x \in \textit{carrier}\ D\ \textit{then}\ \mathbf{1}_C\ \textit{else}\ \mathbf{1}_C$⦈ **unfolding** *diff-group-def*

comm-group-def **by** *simp*
let *?hom-D-C* = $(\!|$*carrier* = *hom-completion D C*, *mult* = $\lambda f\ g\ x.$ *if* $x \in$ *carrier D then* $f\ x \otimes_C g\ x$ *else* $\mathbf{1}_C$,
one = $\lambda x.$ *if* $x \in$ *carrier D then* $\mathbf{1}_C$ *else* $\mathbf{1}_C|\!)$
from *ring-R* **and** *group-hom.hom-one* [*of D C f*] **and** *f* **and** *C* **and** *D-diff-group* **have** *one-fzero*: *one ?hom-D-C* = $f \circ \mathbf{0}_R$
unfolding *diff-group-def comm-group-def group-def hom-completion-def group-hom-def group-hom-axioms-def* **by** (*simp add*: *expand-fun-eq*)
also from *R.r-neg* [*OF a*] **have** ... = $f \circ (a \oplus_R \ominus_R a)$ **by** *simp*
also from *l-add-dist-comp* [*OF C f a a-inv-closed* [*OF a*]] **have** ... = ($\lambda x.$ *if* $x \in$ *carrier D then* $(f \circ a)\ x \otimes_C (f \circ \ominus_R a)\ x$ *else* $\mathbf{1}_C$)
by *simp*
also from *a-eq-b* **have** ... = ($\lambda x.$ *if* $x \in$ *carrier D then* $(f \circ b)\ x \otimes_C (f \circ \ominus_R a)\ x$ *else* $\mathbf{1}_C$) **by** (*simp add*: *expand-fun-eq*)
also from *Hom-D-C* **have** ... = $(f \circ b) \otimes_{?hom\text{-}D\text{-}C} (f \circ \ominus_R a)$ **by** *simp*
finally have *fb-f-minus-a*: *one ?hom-D-C* = $(f \circ b) \otimes_{?hom\text{-}D\text{-}C} (f \circ \ominus_R a)$ **by** *simp*
from *one-fzero* **have** *one ?hom-D-C* = $f \circ \mathbf{0}_R$ **by** *simp*
also from *R.l-neg* [*OF b*] **have** ... = $f \circ (\ominus_R b \oplus_R b)$ **by** *simp*
also from *l-add-dist-comp* [*OF C f a-inv-closed* [*OF b*] *b*] **and** *Hom-D-C* **have** ... = $(f \circ \ominus_R b) \otimes_{?hom\text{-}D\text{-}C} (f \circ b)$ **by** *simp*
finally have *f-minus-b-fb*: *one ?hom-D-C* = $(f \circ \ominus_R b) \otimes_{?hom\text{-}D\text{-}C} (f \circ b)$ **by** *simp*
from *monoid.inv-unique* [*of ?hom-D-C* $(f \circ \ominus_R b)$ $f \circ b$ $(f \circ \ominus_R a)$] **and** *a-inv-closed* [*OF a*] *b a-inv-closed* [*OF b*]
and *sym* [*OF f-minus-b-fb*] **and** *sym* [*OF fb-f-minus-a*] **and** *ring-R*
and *lemma-2-2-18-local-nilpot.hom-completion-comp-closed* [*of D D C* $\ominus_R$ *a f*]
and *lemma-2-2-18-local-nilpot.hom-completion-comp-closed* [*of D D C b f*]
and *lemma-2-2-18-local-nilpot.hom-completion-comp-closed* [*of D D C* $\ominus_R$ *b f*]
and *C* **and** *D-diff-group* **and** *f* **and** *Hom-D-C*
show $(f \circ \ominus_R a) = (f \circ \ominus_R b)$ **unfolding** *diff-group-def comm-group-def group-def* **by** *simp*
qed

The following properties are used later in lemma *BPL-simplifications*; just in order to make the proof of *BPL-simplification* shorter, we have extracted them, as far as they are not generic properties that can be used in other different settings

lemma (**in** *BPL*) *inc-ker-pτ-eq-τ*: **shows** *inc-ker-p'* $\circ\ \tau = \tau$
proof –
have *image* τ (π ‘ *carrier D*) $\subseteq$ *kernel D' D' p'*
proof (*unfold image-def τ-def*, *auto*)
fix *x* **assume** *x*: $x \in$ *carrier D* **with** *π-in-R* **and** *ring-R* **and** *hom-completion-closed* [*of π D D x*] **have** $\pi\ x \in$ *carrier D* **by** *simp*
with *im-π-ker-p* **and** *D'-def* **show** π' ($\pi\ x$) $\in$ *kernel D' D' p'* **unfolding** *image-def* **by** *auto*
qed
with *inc-ker-p'-def* **and** *τ-def* **and** *D'-def* **show** *inc-ker-p'* $\circ\ \tau = \tau$ **unfolding** *completion-def expand-fun-eq* **by** *auto*

qed

lemma (**in** *BPL*) τ*g-eq-*π'*g*: **shows** $\tau \circ g = \pi' \circ g$
proof –
from π*-gf* **have** *im-g-im-*π: *image g* (*carrier C*) $\subseteq$ π‘ *carrier D*
proof (*unfold image-def*, *auto simp add*: *expand-fun-eq*)
fix *x* **assume** *x*: $x \in$ *carrier C* **with** *fg* **have** *fg-x*: f (g x) = x **by** (*simp add*: *expand-fun-eq*)
with *hom-diff-closed* [*OF g-hom-diff x*] **have** g (f (g x)) = g x **and** g $x \in$ *carrier D* **by** (*simp-all*)
with π*-gf* **show** $\exists$ *y*$\in$*carrier D*. g x = π (*y*) **by** (*force simp add*: *expand-fun-eq*)
qed
show $\tau \circ g = \pi' \circ g$
proof (*rule ext*)
fix *x*
show ($\tau \circ g$) x = ($\pi' \circ g$) x
proof (*cases* $x \in$ *carrier C*)
case *True* **with** *im-g-im-*π **and** τ*-def* **and** *g-hom-diff* **show** ($\tau \circ g$) x = (π' $\circ$ g) x
unfolding *completion-def hom-diff-def hom-completion-def hom-def Pi-def* **by** (*auto simp add*: *expand-fun-eq*)
next
case *False* **with** *g-hom-diff* **and** *completion-closed2* [*of g C D x*] **have** *g-x*: g x = $\mathbf{1}_D$
unfolding *hom-diff-def hom-completion-def* **by** *simp*
with *lemma-2-2-17* **and** *hom-completion-one* [*of diff-group-ker-p diff-group-ker-p*$'$ τ] **and** *lemma-2-2-14* **and** *lemma-2-2-15*
have *r-h-s*: τ (g x) = $\mathbf{1}_D$
unfolding *reduction-def diff-group-def comm-group-def group-def*
unfolding τ*-def iso-inv-compl-def iso-compl-def diff-group-ker-p-def diff-group-ker-p*$'$*-def* **by** *simp*
from *C-diff-group D-diff-group* **and** *g-x* **and** π'*-in-R* **and** *ring-R* **and** *hom-completion-one* [*of D D* π']
have *l-h-s*: π' (g x) = $\mathbf{1}_D$ **unfolding** *diff-group-def comm-group-def group-def* **by** *simp*
from *r-h-s* **and** *l-h-s* **show** ($\tau \circ g$) x = ($\pi' \circ g$) x **by** *simp*
qed
qed
qed

lemma (**in** *BPL*) *diff*$'$*h*$'$*g-eq-zero*: **shows** (*diff*$'$ $\otimes_R$ h') $\circ$ g = (λx. *if* $x \in$ *carrier C then* $\mathbf{1}$ *else* $\mathbf{1}$)
proof –
from *ring-R* **have** (*diff*$'$ $\otimes_R$ h') $\circ$ g = (*diff*$'$ $\circ$ h') $\circ$ g **by** *simp*
also have ... = *diff*$'$ $\circ$ (h' $\circ$ g) **by** (*simp add*: *o-assoc*)
also from h'*-def* **and** *psi-h-h-phi* **and** *ring-R* **have** ... = *diff*$'$ $\circ$ (($\Psi \circ h$) $\circ$ g) **by** *simp*
also have ... = *diff*$'$ $\circ$ ($\Psi \circ$ ($h \circ g$)) **by** (*simp add*: *o-assoc*)
also from *hg* **have** ... = *diff*$'$ $\circ$ ($\Psi \circ$ (λx. *if* $x \in$ *carrier C then* $\mathbf{1}$ *else* $\mathbf{1}$)) **by**

simp
also from *psi-in-R* **and** *diff'-in-R* **and** *ring-R* **and** *hom-completion-one* [*of D D diff'*] *hom-completion-one* [*of D D* Ψ]
and *C-diff-group* **and** *D-diff-group*
have ... = (λ*x*. *if x* ∈ *carrier C then* **1** *else* **1**) **unfolding** *diff-group-def comm-group-def group-def* **by** (*simp add*: *expand-fun-eq*)
finally show *?thesis* **by** *simp*
qed

lemma (**in** *BPL*) *h'diff'g-eq-psihdeltag*: **shows** $(h' \otimes_R \mathit{diff}') \circ g = (\Psi \otimes_R h \otimes_R \delta) \circ g$
proof −
from *ring-R* **and** *sym* [*OF o-assoc* [*of h' diff' g*]] **have** $(h' \otimes_R \mathit{diff}') \circ g = h' \circ (\mathit{diff}' \circ g)$ **by** *simp*
also from *add-dist-comp* [*of C g differ* δ] **and** *diff'-def* **and** *diff-in-R* **and** *pert-in-R* **and** *C-diff-group* **and** *g-hom-diff*
have ... = $h' \circ (\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ C\ \mathit{then}\ (\mathit{differ}_D \circ g)\ x \otimes_D (\delta \circ g)\ x\ \mathit{else}\ \mathbf{1})$ **unfolding** *hom-diff-def* **by** *simp*
also from *hom-diff-coherent* [*OF g-hom-diff*]
have ... = $h' \circ (\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ C\ \mathit{then}\ (g \circ \mathit{differ}_C)\ x \otimes_D (\delta \circ g)\ x\ \mathit{else}\ \mathbf{1})$ **by** (*simp add*: *expand-fun-eq*)
also have ... = $(\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ C\ \mathit{then}\ (h' \circ (g \circ \mathit{differ}_C))\ x \otimes (h' \circ (\delta \circ g))\ x\ \mathit{else}\ \mathbf{1})$
proof (*auto simp add*: *expand-fun-eq*)
fix *x* **assume** *x*: *x* ∈ *carrier C*
from *ring-R* **and** *h'-in-R* **and** *hom-completion-mult* [*of h' D D* $(g\ ((\mathit{differ}_C)\ x))\ \delta\ (g\ x)$]
and *hom-completion-closed* [*OF diff-group.diff-hom* [*OF C-diff-group*] *x*]
and *hom-diff-closed* [*OF g-hom-diff*, *of* (differ_C) *x*]
and *hom-diff-closed* [*OF g-hom-diff x*] **and** *pert-in-R hom-completion-closed* [*of* δ *D D g x*]
show $h'\ (g\ ((\mathit{differ}_C)\ x) \otimes \delta\ (g\ x)) = h'\ (g\ ((\mathit{differ}_C)\ x)) \otimes h'\ (\delta\ (g\ x))$ **by** *simp*
next
from *D.diff-group-is-group* **have** *D*: *group D* **by** *simp*
with *ring-R* **and** *h'-in-R* **show** *h'* **1** = **1** **by** (*intro hom-completion-one*, *simp-all*)
qed
also have ... = $(\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ C\ \mathit{then}\ (h' \circ g \circ \mathit{differ}_C)\ x \otimes (h' \circ (\delta \circ g))\ x\ \mathit{else}\ \mathbf{1})$ **by** (*simp add*: *expand-fun-eq o-assoc*)
also from *h'-def* **and** *ring-R* **have** ... = $(\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ C\ \mathit{then}\ (h \circ \Phi \circ g \circ \mathit{differ}_C)\ x \otimes (h \circ \Phi \circ (\delta \circ g))\ x\ \mathit{else}\ \mathbf{1})$
by (*simp add*: *expand-fun-eq*)
also from *psi-h-h-phi* **and** *ring-R* **have** ... = $(\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ C\ \mathit{then}\ (\Psi \circ h \circ g \circ \mathit{differ}_C)\ x \otimes (\Psi \circ h \circ (\delta \circ g))\ x\ \mathit{else}\ \mathbf{1})$
by (*simp add*: *expand-fun-eq*)
also have ... = $(\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ C\ \mathit{then}\ (\Psi \circ h \circ (\delta \circ g))\ x\ \mathit{else}\ \mathbf{1})$
proof (*auto simp add*: *expand-fun-eq*)
fix *x* **assume** *x*: *x* ∈ *carrier C*

from *hg* **and** *psi-in-R* **and** *hom-completion-one* [*of D D* Ψ] **and** *ring-R* **and** *D.diff-group-is-group*
have Ψ (h (g ((*differ*$_C$) x))) = **1** **by** (*simp add*: *expand-fun-eq*)
moreover have Ψ (h (δ (g x))) $\in$ *carrier D*
proof −
from *pert-in-R* **and** *h-in-R* **and** *psi-in-R* **have** $\Psi \otimes_R h \otimes_R \delta \in$ *carrier R* **by** *algebra*
with *ring-R* **have** *psi-h-pert*: $\Psi \circ h \circ \delta \in$ *hom-completion D D* **by** *simp*
from *hom-completion-closed* [*OF psi-h-pert*, *of g x*] **and** *hom-diff-closed* [*OF g-hom-diff x*]
show Ψ (h (δ (g x))) $\in$ *carrier D* **by** *simp*
qed
ultimately show Ψ (h (g ((*differ*$_C$) x))) $\otimes$ Ψ (h (δ (g x))) = Ψ (h (δ (g x))) **by** (*simp add*: *D.r-one*)
qed
also from *ring-R* **have** ... = (λx. *if* $x \in$ *carrier C then* ($\Psi \otimes_R h \otimes_R \delta \circ g$) x *else* **1**) **by** (*simp add*: *expand-fun-eq*)
also from *comp-hom-compl* [*of C g* $\Psi \otimes_R h \otimes_R \delta$] **and** *pert-in-R h-in-R psi-in-R* **and** *g-hom-diff C-diff-group*
have ... = ($\Psi \otimes_R h \otimes_R \delta$) $\circ$ g **unfolding** *hom-diff-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma (**in** *BPL*) *p′g-eq-psihdeltag*: **shows** $p' \circ g$ = ($\Psi \otimes_R h \otimes_R \delta$) $\circ$ g
proof −
from *p′-def* **and** *ring-R* **and** *diff′-def* **have** $p' \circ g$ = ((*diff*′ $\otimes_R$ h') $\oplus_R$ (h' $\otimes_R$ *diff*′)) $\circ$ g **by** (*simp add*: *expand-fun-eq*)
also from *add-dist-comp* [*of C g* (*diff*′ $\otimes_R$ h') (h' $\otimes_R$ *diff*′)] **and** *g-hom-diff* **and** *diff′-in-R h′-in-R* **and** *C-diff-group*
have ... = (λx. *if* $x \in$ *carrier C then* ((*diff*′ $\otimes_R$ h') $\circ$ g) x $\otimes$ ((h' $\otimes_R$ *diff*′) $\circ$ g) x *else* **1**) **by** (*unfold hom-diff-def*, *simp*)
also from *diff′h′g-eq-zero* **and** *h′diff′g-eq-psihdeltag* **have** ... = (λx. *if* $x \in$ *carrier C then* ($\Psi \otimes_R h \otimes_R \delta \circ g$) x *else* **1**)
proof (*auto simp add*: *expand-fun-eq*, *intro D.l-one*)
fix x **assume** x: $x \in$ *carrier C*
from *pert-in-R* **and** *h-in-R* **and** *psi-in-R* **have** $\Psi \otimes_R h \otimes_R \delta \in$ *carrier R* **by** *algebra*
with *ring-R* **have** *psi-h-pert*: $\Psi \otimes_R h \otimes_R \delta \in$ *hom-completion D D* **by** *simp*
from *hom-completion-closed* [*OF psi-h-pert*, *of g x*] **and** *hom-diff-closed* [*OF g-hom-diff x*]
show ($\Psi \otimes_R h \otimes_R \delta$) ($g$ x) $\in$ *carrier D* **by** *simp*
qed
also from *comp-hom-compl* [*of C g* $\Psi \otimes_R h \otimes_R \delta$] **and** *pert-in-R h-in-R psi-in-R* **and** *g-hom-diff C-diff-group*
have ... = ($\Psi \otimes_R h \otimes_R \delta$) $\circ$ g **by** (*unfold hom-diff-def*, *simp*)
finally show *?thesis* **by** *simp*
qed

lemma (**in** *BPL*) *τ′π′-eq-ππ′*: **shows** $\tau' \circ \pi' = \pi \circ \pi'$

proof (*rule ext*)
fix *x*
show $(\tau' \circ \pi')\ x = (\pi \circ \pi')\ x$
proof (*cases* $x \in$ *carrier D*)
case *True* **with** π'*-in-R ring-R* **and** *hom-completion-closed* [*of* π' *D D x*]
have π'*-im*: $\pi'\ x \in \pi'$‘(*carrier D*) **and** π'*-D*: $\pi'\ x \in$ *carrier D* **unfolding** *image-def* **by** *auto*
with τ'*-def* **and** D'*-def* **show** $(\tau' \circ \pi')\ x = (\pi \circ \pi')\ x$ **by** (*simp add*: *expand-fun-eq*)
next
case *False* **with** π'*-in-R* **and** *ring-R* **and** *completion-closed2* [*of* π' *D D x*] **have** π'*-x*: $\pi'\ x = \mathbf{1}$ **unfolding** *hom-completion-def* **by** *simp*
with *lemma-2-2-17* **and** *hom-completion-one* [*of diff-group-ker-p' diff-group-ker-p* τ'] **and** *lemma-2-2-14* **and** *lemma-2-2-15*
have *r-h-s*: $\tau'\ (\pi'\ x) = \mathbf{1}_D$
unfolding *reduction-def diff-group-def comm-group-def group-def* τ'*-def*
unfolding *iso-inv-compl-def iso-compl-def diff-group-ker-p-def diff-group-ker-p'-def* **by** *simp*
from π*-in-R* **and** *ring-R* **and** *hom-completion-one* [*of D D* π] **and** π'*-x* **and** *D.diff-group-is-group*
have *l-h-s*: $\pi\ (\pi'\ x) = \mathbf{1}$ **by** *simp*
from *r-h-s* **and** *l-h-s* **show** *?thesis* **by** *simp*
qed
qed

lemma (**in** *BPL*) $f'\pi$*-eq-f*π: **shows** $f' \circ \pi = f \circ \pi$
proof (*rule ext*)
fix *x*
show $(f' \circ \pi)\ x = (f \circ \pi)\ x$
proof (*cases* $x \in$ *carrier D*)
case *True* **with** π*-in-R* **and** *ring-R* **and** *hom-completion-closed* [*of* π *D D*] **have** π*-im*: $\pi\ x \in \pi$‘(*carrier D*)
and π*-D*: $\pi\ x \in$ *carrier D* **by** (*unfold image-def*, *auto*)
then have $\pi\ x \in$ *kernel D D p* **using** *im-*π*-ker-p* **by** *simp*
then show $(f' \circ \pi)\ x = (f \circ \pi)\ x$ **unfolding** f'*-def* **by** (*simp add*: *expand-fun-eq*)
next
case *False* **with** π*-in-R* **and** *ring-R* **and** *completion-closed2* [*of* π *D D x*] **have** π*-x*: $\pi\ x = \mathbf{1}$ **unfolding** *hom-completion-def* **by** *simp*
from *iso-inv-diff-rev* [*OF lemma-2-2-11*] **and** π*-gf im-*π*-ker-p*
have $(f', g) \in$ (*diff-group-im-gf* $\cong_{invdiff}$ *C*) **unfolding** f'*-def diff-group-im-gf-def im-gf-def* **by** *simp*
with *hom-completion-one* [*of diff-group-im-gf C* f'] **and** π*-x* **and** *C.diff-group-is-group* **and** *image-g-f-diff-group*
have *r-h-s*: $f'\ (\pi\ x) = \mathbf{1}_C$ **unfolding** *diff-group-im-gf-def im-gf-def diff-group-def comm-group-def*
iso-inv-diff-def iso-diff-def hom-diff-def **by** *simp*
from *hom-diff-is-hom-completion* [*OF f-hom-diff*] **and** *hom-completion-one* [*of D C f*] **and** π*-x*
and *D.diff-group-is-group C.diff-group-is-group* **have** *l-h-s*: $f\ (\pi\ x) = \mathbf{1}_C$ **by**

simp
from *r-h-s* **and** *l-h-s* **show** *?thesis* **by** *simp*
qed
qed

lemma (**in** *BPL*) *fπ-eq-f*: **shows** $f \circ \pi = f$
proof −
from *sym* [*OF π-gf*] **have** $f \circ \pi = f \circ g \circ f$ **by** (*simp add*: *o-assoc*)
also from *fg* **have** $\ldots = (\lambda x.\ if\ x \in carrier\ C\ then\ id\ x\ else\ \mathbf{1}_C) \circ f$ **by** *simp*
also from *f-hom-diff* **have** $\ldots = f$
unfolding *hom-diff-def hom-completion-def completion-fun2-def completion-def hom-def Pi-def* **by** (*auto simp add*: *expand-fun-eq*)
finally have $f \circ \pi = f$ **by** *simp*
then show *?thesis* **by** (*simp add*: *expand-fun-eq*)
qed

lemma (**in** *BPL*) *fh′diff′-eq-zero*: **shows** $f \circ (h' \otimes_R diff') = (\lambda x.\ if\ x \in carrier\ D\ then\ \mathbf{1}_C\ else\ \mathbf{1}_C)$
proof −
from *ring-R* **have** $f \circ (h' \otimes_R diff') = f \circ h' \circ diff'$ **by** (*simp add*: *o-assoc*)
also from *ring-R* **and** *h′-def* **and** *o-assoc* [*of f h Φ*] **have** $\ldots = f \circ h \circ \Phi \circ diff'$ **by** *simp*
also from *fh* **have** $\ldots = (\lambda x.\ if\ x \in carrier\ D\ then\ \mathbf{1}_C\ else\ \mathbf{1}_C)$ **by** (*simp add*: *expand-fun-eq*)
finally show $f \circ (h' \otimes_R diff') = (\lambda x.\ if\ x \in carrier\ D\ then\ \mathbf{1}_C\ else\ \mathbf{1}_C)$ **by** *simp*
qed

lemma (**in** *BPL*) *fdiff′h′-eq-fdeltahphi*: **shows** $f \circ (diff' \otimes_R h') = f \circ \delta \otimes_R h \otimes_R \Phi$
proof −
from *diff′-def* **have** $f \circ (diff' \otimes_R h') = f \circ (differ \oplus_R \delta) \otimes_R h'$ **by** *simp*
also from *diff-in-R* **and** *pert-in-R* **and** *h′-in-R* **have** $\ldots = f \circ (differ \otimes_R h' \oplus_R \delta \otimes_R h')$ **by** *algebra*
also from *l-add-dist-comp* [*OF C-diff-group*, *of f differ* $\otimes_R$ *h′ δ* $\otimes_R$ *h′*] **and** *diff-in-R* **and** *pert-in-R* **and** *h′-in-R* **and** *f-hom-diff*
have $\ldots = (\lambda x.\ if\ x \in carrier\ D\ then\ (f \circ differ \otimes_R h')\ x \otimes_C (f \circ \delta \otimes_R h')\ x\ else\ \mathbf{1}_C)$ **unfolding** *hom-diff-def* **by** *simp*
also from *ring-R* **have** $\ldots = (\lambda x.\ if\ x \in carrier\ D\ then\ (f \circ differ \circ h')\ x \otimes_C (f \circ \delta \otimes_R h')\ x\ else\ \mathbf{1}_C)$ **by** (*simp add*: *expand-fun-eq*)
also from *hom-diff-coherent* [*OF f-hom-diff*]
have $\ldots = (\lambda x.\ if\ x \in carrier\ D\ then\ (differ_C \circ f \circ h')\ x \otimes_C (f \circ \delta \otimes_R h')\ x\ else\ \mathbf{1}_C)$ **by** (*simp add*: *expand-fun-eq*)
also from *h′-def* **and** *ring-R* **have** $\ldots = (\lambda x.\ if\ x \in carrier\ D\ then\ (differ_C \circ f \circ h \circ \Phi)\ x \otimes_C (f \circ \delta \otimes_R h')\ x\ else\ \mathbf{1}_C)$
by (*simp add*: *expand-fun-eq*)
also from *fh* **and** *hom-completion-one* [*OF - - diff-hom*] *C.diff-group-is-group* **and** *l-one* **and** *hom-completion-closed* [*of h′ D D*]
and *hom-completion-closed* [*of δ D D*] **and** *hom-diff-closed* [*OF f-hom-diff*]

and *ring-R* **and** *h′-in-R* **and** *pert-in-R*
have ... = ($\lambda x.$ *if* $x \in$ *carrier D then* ($f \circ \delta \otimes_R h'$) x *else* $\mathbf{1}_C$) **by** (*simp add*: *expand-fun-eq*)
also from *l-comp-hom-compl* [*OF C-diff-group*, *of f* $\delta \otimes_R h'$] **and** *f-hom-diff* **have** ... = $f \circ \delta \otimes_R h'$ **unfolding** *hom-diff-def* **by** *simp*
also from *h′-def* **and** *pert-in-R* **and** *h-in-R* **and** *phi-in-R* **have** ... = $f \circ \delta \otimes_R h \otimes_R \Phi$ **by** *algebra*
finally show *?thesis* **by** *simp*
qed

lemma (**in** *BPL*) *fp′-eq-fdeltahphi*: **shows** $f \circ p' = f \circ \delta \otimes_R h \otimes_R \Phi$
proof −
from *p′-def* **and** *diff′-def* **and** *l-add-dist-comp* [*OF C-diff-group*, *of f diff′* $\otimes_R$ *h′ h′* $\otimes_R$ *diff′*] *f-hom-diff*
and *diff′-in-R pert-in-R h′-in-R*
have $f \circ p'$ = ($\lambda x.$ *if* $x \in$ *carrier D then* ($f \circ$ *diff′* $\otimes_R h'$) $x \otimes_C$ ($f \circ h' \otimes_R$ *diff′*) x *else* $\mathbf{1}_C$) **by** (*unfold hom-diff-def*, *simp*)
also from *fh′diff′-eq-zero* **and** *fdiff′h′-eq-fdeltahphi* **and** *r-one* **and** *hom-completion-closed* [*of h D D*] **and** *hom-completion-closed* [*of* Φ *D D*]
and *hom-completion-closed* [*of* δ *D D*] **and** *hom-diff-closed* [*OF f-hom-diff*] **and** *ring-R* **and** *h-in-R* **and** *phi-in-R* **and** *pert-in-R*
have ... = ($\lambda x.$ *if* $x \in$ *carrier D then* ($f \circ \delta \otimes_R h \otimes_R \Phi$) x *else* $\mathbf{1}_C$) **by** (*simp add*: *expand-fun-eq*)
also from *l-comp-hom-compl* [*OF C-diff-group*, *of f* $\delta \otimes_R h \otimes_R \Phi$] **and** *h-in-R* **and** *phi-in-R* **and** *pert-in-R f-hom-diff*
have ... = $f \circ \delta \otimes_R h \otimes_R \Phi$ **unfolding** *hom-diff-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma (**in** *BPL*) *diff-ker-p′π′-eq-diff′π′*: **shows** $\mathit{differ}_{\mathit{diff\text{-}group\text{-}ker\text{-}p'}} \circ \pi' = \mathit{diff}' \circ \pi'$
proof (*rule ext*)
fix x
show ($\mathit{differ}_{\mathit{diff\text{-}group\text{-}ker\text{-}p'}} \circ \pi'$) x = ($\mathit{diff}' \circ \pi'$) x
proof (*cases* $x \in$ *carrier D*)
case *True* **from** *im-π-ker-p* **and** *D′-def* **have** *im-π′-ker-p′*: *image* π' (*carrier D*) $\subseteq$ *kernel D′ D′ p′* **by** *simp*
with *True* **show** ($\mathit{differ}_{\mathit{diff\text{-}group\text{-}ker\text{-}p'}} \circ \pi'$) x = ($\mathit{diff}' \circ \pi'$) x
unfolding *diff-group-ker-p′-def diff′-def D′-def completion-def image-def* **by** (*auto simp add*: *expand-fun-eq*)
next
case *False* **with** *π′-in-R* **and** *ring-R* **and** *completion-closed2* [*of* π' *D D x*] **have** *π′-x*: $\pi'\ x = \mathbf{1}$ **unfolding** *hom-completion-def* **by** *simp*
with *diff′-in-R* **and** *ring-R* **and** *hom-completion-one* [*of D D diff′*] **and** *D.diff-group-is-group*
have *l-h-s*: ($\mathit{diff}' \circ \pi'$) $x = \mathbf{1}$ **by** *simp*
from *reduction.C-diff-group* [*OF lemma-2-2-15*] **and** *diff-group.diff-hom*
have *diff-ker-p′*: $\mathit{differ}_{\mathit{diff\text{-}group\text{-}ker\text{-}p'}} \in$ *hom-completion diff-group-ker-p′ diff-group-ker-p′* **by** *auto*

from π'-*x* **and** *hom-completion-one* [*OF - - diff-ker-p'*] **and** *reduction.C-diff-group* [*OF lemma-2-2-15*]
have *r-h-s*: $(differ_{diff\text{-}group\text{-}ker\text{-}p'} \circ \pi')\ x = \mathbf{1}$ **unfolding** *diff-group-ker-p'-def diff-group-def comm-group-def group-def* **by** *simp*
from *l-h-s* **and** *r-h-s* **show** *?thesis* **by** *simp*
qed
qed

lemma (**in** *BPL*) $\tau'diff'\text{-}eq\text{-}\pi diff'$: **shows** $\tau' \circ differ_{diff\text{-}group\text{-}ker\text{-}p'} = \pi \circ differ_{diff\text{-}group\text{-}ker\text{-}p'}$
proof (*rule ext*)
fix *x*
show $(\tau' \circ differ_{diff\text{-}group\text{-}ker\text{-}p'})\ x = (\pi \circ differ_{diff\text{-}group\text{-}ker\text{-}p'})\ x$
proof (*cases* $x \in kernel\ D'\ D'\ p'$)
case *True* **from** *reduction.C-diff-group* [*OF lemma-2-2-15*] **and** *diff-group.diff-hom*
have $differ_{diff\text{-}group\text{-}ker\text{-}p'} \in hom\text{-}completion\ diff\text{-}group\text{-}ker\text{-}p'\ diff\text{-}group\text{-}ker\text{-}p'$ **by** *auto*
then have $image\ (differ_{diff\text{-}group\text{-}ker\text{-}p'})\ (kernel\ D'\ D'\ p') \subseteq kernel\ D'\ D'\ p'$
unfolding *diff-group-ker-p'-def hom-completion-def hom-def Pi-def image-def kernel-def D'-def* **by** *auto*
with *im-π-ker-p* **and** *D'-def* **have** $image\ (differ_{diff\text{-}group\text{-}ker\text{-}p'})\ (kernel\ D'\ D'\ p') \subseteq image\ \pi'\ (carrier\ D)$ **by** *simp*
with *True* **and** τ'-*def* **show** $(\tau' \circ differ_{diff\text{-}group\text{-}ker\text{-}p'})\ x = (\pi \circ differ_{diff\text{-}group\text{-}ker\text{-}p'})\ x$
unfolding *completion-def image-def D'-def* **by** (*auto simp add*: *expand-fun-eq*)
next
case *False* **from** *reduction.C-diff-group* [*OF lemma-2-2-15*] **and** *diff-group.diff-hom*
have $differ_{diff\text{-}group\text{-}ker\text{-}p'} \in hom\text{-}completion\ diff\text{-}group\text{-}ker\text{-}p'\ diff\text{-}group\text{-}ker\text{-}p'$ **by** *auto*
with *completion-closed2* [*of* $differ_{diff\text{-}group\text{-}ker\text{-}p'}$ *diff-group-ker-p' diff-group-ker-p' x*] **and** *False* **and** *D'-def*
have *diff'-x*: $(differ_{diff\text{-}group\text{-}ker\text{-}p'})\ x = \mathbf{1}$ **unfolding** *hom-completion-def diff-group-ker-p'-def* **by** *simp*
with *lemma-2-2-17* **and** *hom-completion-one* [*of diff-group-ker-p' diff-group-ker-p* τ'] **and** *lemma-2-2-14* **and** *lemma-2-2-15*
have *r-h-s*: $\tau'\ ((differ_{diff\text{-}group\text{-}ker\text{-}p'})\ x) = \mathbf{1}_D$
unfolding *reduction-def diff-group-def comm-group-def group-def τ'-def iso-inv-compl-def iso-compl-def diff-group-ker-p-def*
diff-group-ker-p'-def **by** *simp*
from *π-in-R* **and** *ring-R* **and** *hom-completion-one* [*of D D π*] **and** *diff'-x* **and** *D.diff-group-is-group*
have *l-h-s*: $\pi\ ((differ_{diff\text{-}group\text{-}ker\text{-}p'})\ x) = \mathbf{1}$ **by** *simp*
from *r-h-s* **and** *l-h-s* **show** *?thesis* **by** *simp*
qed
qed

lemma (**in** *BPL*) $f\pi'g\text{-}eq\text{-}id$: **shows** $f \circ \pi' \circ g = (\lambda x.\ if\ x \in carrier\ C\ then\ id\ x$

else $\mathbf{1}_C$)
proof −
have $f \circ \pi' \circ g = f \circ (\mathbf{1}_R \ominus_R p') \circ g$ **unfolding** π'-*def* **by** *simp*
also from *l-minus-dist-comp* [*OF C-diff-group - R.one-closed*, *of f p'*] **and** *p'-in-R* **and** *f-hom-diff* **and** *C-diff-group*
have $\ldots = (\lambda x.\ \textit{if}\ x \in \textit{carrier D then}\ (f \circ \mathbf{1}_R)\ x \otimes_C (f \circ (\ominus_R\ p'))\ x\ \textit{else}\ \mathbf{1}_C) \circ g$ **unfolding** *hom-diff-def* **by** *simp*
also from *phi-in-R h-in-R pert-in-R* **and** *l-minus-comp-f* [*OF C-diff-group*, *of f* $p'\ \delta \otimes_R h \otimes_R \Phi$] **and** *fp'-eq-fdeltahphi* **and** *f-hom-diff*
have $\ldots = (\lambda x.\ \textit{if}\ x \in \textit{carrier D then}\ (f \circ \mathbf{1}_R)\ x \otimes_C (f \circ \ominus_R (\delta \otimes_R h \otimes_R \Phi))\ x\ \textit{else}\ \mathbf{1}_C) \circ g$
unfolding *hom-diff-def* **by** (*auto simp add*: *expand-fun-eq*)
also from *sym* [*OF l-minus-dist-comp* [*OF C-diff-group - R.one-closed*, *of f* ($\delta \otimes_R h \otimes_R \Phi$)]] **and** *f-hom-diff* **and** *pert-in-R h-in-R phi-in-R*
have $\ldots = f \circ (\mathbf{1}_R \ominus_R (\delta \otimes_R h \otimes_R \Phi)) \circ g$ **unfolding** *hom-diff-def* **by** *simp*
also from *phi-prop* **have** $\ldots = f \circ (\mathbf{1}_R \ominus_R \Phi \otimes_R \delta \otimes_R h) \circ g$ **by** *simp*
also from *minus-dist-comp* [*OF C-diff-group - R.one-closed*, *of g* $\Phi \otimes_R \delta \otimes_R h$] **and** *g-hom-diff* **and** *phi-in-R pert-in-R h-in-R*
have $\ldots = f \circ (\lambda x.\ \textit{if}\ x \in \textit{carrier C then}\ (\mathbf{1}_R \circ g)\ x \otimes (\ominus_R (\Phi \otimes_R \delta \otimes_R h) \circ g)\ x\ \textit{else}\ \mathbf{1})$
by (*unfold hom-diff-def*, *simp add*: *expand-fun-eq*)
also have $\ldots = f \circ (\lambda x.\ \textit{if}\ x \in \textit{carrier C then}\ (\mathbf{1}_R \circ g)\ x\ \textit{else}\ \mathbf{1})$
proof −
have $(\ominus_R (\Phi \otimes_R \delta \otimes_R h) \circ g) = \ominus_R \mathbf{0}_R \circ g$
proof −
from *hg* **and** *ring-R* **and** *hom-completion-one* [*of D D* δ] *hom-completion-one* [*of D D* Φ] **and** *D.diff-group-is-group* **and** *pert-in-R* **and** *phi-in-R*
have $(\Phi \otimes_R \delta \otimes_R h) \circ g = \mathbf{0}_R \circ g$ **by** (*simp add*: *expand-fun-eq*)
with *minus-comp-g2* [*OF C-diff-group - - R.zero-closed*, *of g* $\Phi \otimes_R \delta \otimes_R h$] **and** *g-hom-diff* **and** *phi-in-R pert-in-R h-in-R*
show $(\ominus_R (\Phi \otimes_R \delta \otimes_R h) \circ g) = \ominus_R \mathbf{0}_R \circ g$ **unfolding** *hom-diff-def* **by** *simp*
qed
also have $\ldots = \mathbf{0}_R \circ g$ **by** *simp*
also from *ring-R* **have** $\ldots = (\lambda x.\ \mathbf{1})$ **by** (*simp add*: *expand-fun-eq*)
finally have $\ominus_R (\Phi \otimes_R \delta \otimes_R h) \circ g = (\lambda x.\ \mathbf{1})$ **by** *simp*
with *D.r-one* **and** *g-hom-diff* **and** *hom-completion-closed* [*of g C D*] **and** *ring-R* **show** *?thesis* **by** (*simp add*: *expand-fun-eq*)
qed
also from *comp-hom-compl* [*OF C-diff-group - R.one-closed*, *of g*] **and** *g-hom-diff* **have** $\ldots = f \circ (\mathbf{1}_R \circ g)$ **unfolding** *hom-diff-def* **by** *simp*
also from *g-hom-diff* **and** *hom-completion-closed* [*of g C D*] **and** *ring-R* **have** $\ldots = f \circ g$
proof −
from *g-hom-diff* **and** *hom-completion-closed* [*of g C D*] **and** *ring-R* **have** $(\mathbf{1}_R \circ g) = g$
unfolding *hom-diff-def hom-completion-def completion-fun2-def completion-def* **by** (*auto simp add*: *expand-fun-eq*)
then show *?thesis* **by** *simp*

qed
also from *fg* **have** $\ldots = (\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ C\ \mathit{then}\ \mathit{id}\ x\ \mathit{else}\ \mathbf{1}_C)$ **by** *simp*
finally show $f \circ \pi' \circ g = (\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ C\ \mathit{then}\ \mathit{id}\ x\ \mathit{else}\ \mathbf{1}_C)$ **by** *simp*
qed

lemma (**in** *BPL*) π'*g-eq-psig*: **shows** $\pi' \circ g = \Psi \circ g$
proof −
from π'*-def* **have** $\pi' \circ g = (\mathbf{1}_R \ominus_R p') \circ g$ **by** *simp*
also from *minus-dist-comp* [*of C g* $\mathbf{1}_R$ p'] **and** p'*-in-R* **and** *R.one-closed* **and** *g-hom-diff* **and** *C-diff-group*
have $\ldots = (\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ C\ \mathit{then}\ (\mathbf{1}_R \circ g)\ x \otimes ((\ominus_R p') \circ g)\ x\ \mathit{else}\ \mathbf{1})$ **by** (*unfold hom-diff-def*, *simp*)
also from *psi-in-R h-in-R pert-in-R* **and** *minus-comp-g2* [*OF C-diff-group*, *of g* p' $\Psi \otimes_R h \otimes_R \delta$] **and** p'*g-eq-psihdeltag* **and** *g-hom-diff*
have $\ldots = (\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ C\ \mathit{then}\ (\mathbf{1}_R \circ g)\ x \otimes (\ominus_R (\Psi \otimes_R h \otimes_R \delta) \circ g)\ x\ \mathit{else}\ \mathbf{1})$
unfolding *hom-diff-def* **by** (*auto simp add*: *expand-fun-eq*)
also from *sym* [*OF minus-dist-comp* [*OF C-diff-group - R.one-closed*, *of g* ($\Psi \otimes_R h \otimes_R \delta$)]] **and** *g-hom-diff* **and** *pert-in-R h-in-R phi-in-R*
have $\ldots = (\mathbf{1}_R \ominus_R (\Psi \otimes_R h \otimes_R \delta)) \circ g$ **unfolding** *hom-diff-def* **by** *simp*
also from *psi-prop* **have** $\ldots = \Psi \circ g$ **by** *simp*
finally show *?thesis* **by** *simp*
qed

11.4 BPL simplification

Now we can prove the simplifications of the terms in the reduction; these simplification processes correspond to the ones in pages 56 and 57 of Aransay's memoir

lemma (**in** *BPL*) *BPL-simplifications*: **shows** *f*: $(f' \circ (\tau' \circ \mathbf{1}_R \ominus_R p')) = f \circ \Phi$ **and** *g*: $(\mathit{inc\text{-}ker\text{-}p}' \circ \tau \circ g) = \Psi \circ g$
and *diff-C*: $f' \circ (\tau' \circ \mathit{differ}_{\mathit{diff\text{-}group\text{-}ker\text{-}p}'} \circ \tau) \circ g = (\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ C\ \mathit{then}\ (\mathit{differ}_C)\ x \otimes_C (f \circ \delta \circ \Psi \circ g)\ x\ \mathit{else}\ \mathbf{1}_C)$
proof −
show $f' \circ (\tau' \circ \mathbf{1}_R \ominus_R p') = f \circ \Phi$

proof −
have $f' \circ (\tau' \circ \mathbf{1}_R \ominus_R p') = f' \circ (\tau' \circ \pi')$ **unfolding** π'*-def* **by** *simp*
also have $\ldots = f' \circ (\pi \circ \pi')$ **unfolding** $\tau'\pi'$*-eq-*$\pi\pi'$ **by** *simp*
also have $\ldots = f \circ \pi \circ \pi'$ **unfolding** *o-assoc* **unfolding** $f'\pi$*-eq-*$f\pi$ **by** *simp*
also have $\ldots = f \circ \pi'$ **unfolding** $f\pi$*-eq-f* **by** *simp*
also have $\ldots = f \circ (\mathbf{1}_R \ominus_R p')$ **unfolding** π'*-def* **by** *simp*
also from *l-minus-dist-comp* [*OF C-diff-group - R.one-closed*, *of f* p'] **and** p'*-in-R* **and** *f-hom-diff* **and** *C-diff-group*
have $\ldots = (\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ D\ \mathit{then}\ (f \circ \mathbf{1}_R)\ x \otimes_C (f \circ (\ominus_R p'))\ x\ \mathit{else}\ \mathbf{1}_C)$ **unfolding** *hom-diff-def* **by** *simp*

also from *phi-in-R h-in-R pert-in-R* **and** *l-minus-comp-f* [*OF C-diff-group*, *of* $f\ p'\ \delta \otimes_R h \otimes_R \Phi$] **and** *fp′-eq-fdeltahphi* **and** *f-hom-diff*
have $\ldots = (\lambda x.$ *if* $x \in$ *carrier D then* $(f \circ \mathbf{1}_R)\ x \otimes_C (f \circ \ominus_R (\delta \otimes_R h \otimes_R \Phi))\ x$ *else* $\mathbf{1}_C)$
unfolding *hom-diff-def* **by** (*auto simp add*: *expand-fun-eq*)
also from *sym* [*OF l-minus-dist-comp* [*OF C-diff-group - R.one-closed*, *of* $f\ (\delta \otimes_R h \otimes_R \Phi)$]] **and** *f-hom-diff* **and** *pert-in-R h-in-R phi-in-R*
have $\ldots = f \circ (\mathbf{1}_R \ominus_R (\delta \otimes_R h \otimes_R \Phi))$ **unfolding** *hom-diff-def* **by** *simp*
also from *phi-prop* **have** $\ldots = f \circ \Phi$ **by** *simp*
finally show *?thesis* **by** *simp*
qed
next
show $(inc\text{-}ker\text{-}p' \circ \tau \circ g) = \Psi \circ g$
proof −
have $inc\text{-}ker\text{-}p' \circ \tau \circ g = \tau \circ g$ **unfolding** *inc-ker-pτ-eq-τ* **by** *simp*
also have $\ldots = \pi' \circ g$ **unfolding** *τg-eq-π′g* **by** *simp*

also have $\pi' \circ g = (\mathbf{1}_R \ominus_R p') \circ g$ **unfolding** *π′-def* **by** *simp*
also from *minus-dist-comp* [*of C g* $\mathbf{1}_R\ p'$] **and** *p′-in-R* **and** *R.one-closed* **and** *g-hom-diff* **and** *C-diff-group*
have $\ldots = (\lambda x.$ *if* $x \in$ *carrier C then* $(\mathbf{1}_R \circ g)\ x \otimes ((\ominus_R p') \circ g)\ x$ *else* $\mathbf{1})$ **unfolding** *hom-diff-def* **by** *simp*
also from *psi-in-R h-in-R pert-in-R* **and** *minus-comp-g2* [*OF C-diff-group*, *of* $g\ p'\ \Psi \otimes_R h \otimes_R \delta$] **and** *p′g-eq-psihdeltag* **and** *g-hom-diff*
have $\ldots = (\lambda x.$ *if* $x \in$ *carrier C then* $(\mathbf{1}_R \circ g)\ x \otimes (\ominus_R (\Psi \otimes_R h \otimes_R \delta) \circ g)\ x$ *else* $\mathbf{1})$
unfolding *hom-diff-def* **by** (*auto simp add*: *expand-fun-eq*)
also from *sym* [*OF minus-dist-comp* [*OF C-diff-group - R.one-closed*, *of* $g\ (\Psi \otimes_R h \otimes_R \delta)$]] **and** *g-hom-diff* **and** *pert-in-R h-in-R phi-in-R*
have $\ldots = (\mathbf{1}_R \ominus_R (\Psi \otimes_R h \otimes_R \delta)) \circ g$ **unfolding** *hom-diff-def* **by** *simp*
also from *psi-prop* **have** $\ldots = \Psi \circ g$ **by** *simp*
finally show *?thesis* **by** *simp*
qed
next
show $f' \circ (\tau' \circ differ_{diff\text{-}group\text{-}ker\text{-}p'} \circ \tau) \circ g = (\lambda x.$ *if* $x \in$ *carrier C then* $(differ_C)\ x \otimes_C (f \circ \delta \circ \Psi \circ g)\ x$ *else* $\mathbf{1}_C)$
(**is** $f' \circ (\tau' \circ\ ?diff\text{-}ker\text{-}p' \circ \tau) \circ g =\ ?diff\text{-}C\text{-}pert$)
proof −
have $f' \circ (\tau' \circ\ ?diff\text{-}ker\text{-}p' \circ \tau) \circ g = (f' \circ \tau') \circ\ ?diff\text{-}ker\text{-}p' \circ (\tau \circ g)$ **by** (*simp add*: *o-assoc*)
also from *τg-eq-π′g* **have** $\ldots = (f' \circ \tau') \circ\ ?diff\text{-}ker\text{-}p' \circ (\pi' \circ g)$ **by** *simp*
also have $\ldots = f' \circ (\tau' \circ\ ?diff\text{-}ker\text{-}p') \circ \pi' \circ g$ **by** (*simp add*: *o-assoc*)
also from *τ′diff′-eq-πdiff′* **have** $\ldots = f' \circ (\pi \circ differ_{diff\text{-}group\text{-}ker\text{-}p'}) \circ \pi' \circ g$ **by** (*simp add*: *expand-fun-eq*)
also have $\ldots = f' \circ \pi \circ (differ_{diff\text{-}group\text{-}ker\text{-}p'} \circ \pi') \circ g$ **by** (*simp add*: *o-assoc*)
also from *diff-ker-p′π′-eq-diff′π′* **have** $\ldots = f' \circ \pi \circ (diff' \circ \pi') \circ g$ **by** (*simp add*: *expand-fun-eq*)
also from *f′π-eq-fπ* **have** $\ldots = f \circ \pi \circ (diff' \circ \pi') \circ g$ **by** *simp*
also from *fπ-eq-f* **have** $\ldots = f \circ (diff' \circ \pi') \circ g$ **by** *simp*

also have ... = $f \circ \mathit{diff}' \circ (\pi' \circ g)$ **by** (*simp add*: *o-assoc*)
also from *diff'-def* **have** ... = $f \circ (\mathit{differ} \oplus_R \delta) \circ (\pi' \circ g)$ **by** *simp*
also have ... = $f \circ ((\mathit{differ} \oplus_R \delta) \circ \pi') \circ g$ **by** (*simp add*: *o-assoc*)
also from *ring-R* **have** ... = $f \circ ((\mathit{differ} \oplus_R \delta) \otimes_R \pi') \circ g$ **by** *simp*
also from *π'-in-R* **and** *diff-in-R* **and** *pert-in-R* **and** *R.one-closed* **have** ... = $f \circ (\mathit{differ} \otimes_R \pi' \oplus_R \delta \otimes_R \pi') \circ g$ **by** *algebra*
also from *l-add-dist-comp* [*OF C-diff-group*, *of f differ* $\otimes_R$ π' δ $\otimes_R$ π'] *f-hom-diff diff-in-R π'-in-R pert-in-R*
have ... = (λx. *if* $x \in$ *carrier D then* $(f \circ (\mathit{differ} \otimes_R \pi'))\ x \otimes_C\ (f \circ (\delta \otimes_R \pi'))\ x$ *else* $\mathbf{1}_C) \circ g$ **by** (*unfold hom-diff-def*, *simp*)
also from *ring-R* **and** *o-assoc* **have** ... = (λx. *if* $x \in$ *carrier D then* $(f \circ \mathit{differ} \circ \pi')\ x \otimes_C\ (f \circ (\delta \otimes_R \pi'))\ x$ *else* $\mathbf{1}_C) \circ g$
by (*simp add*: *expand-fun-eq*)
also from *hom-diff-coherent* [*OF f-hom-diff*]
have ... = (λx. *if* $x \in$ *carrier D then* $(\mathit{differ}_C \circ f \circ \pi')\ x \otimes_C\ (f \circ (\delta \otimes_R \pi'))\ x$ *else* $\mathbf{1}_C) \circ g$ **by** (*simp add*: *expand-fun-eq*)
also have ... = (λx. *if* $x \in$ *carrier C then* $((\mathit{differ}_C \circ f \circ \pi') \circ g)\ x \otimes_C\ ((f \circ (\delta \otimes_R \pi')) \circ g)\ x$ *else* $\mathbf{1}_C$)
proof (*auto simp add*: *expand-fun-eq*)
fix x
assume x: $x \in$ *carrier C* **and** $g\ x \notin$ *carrier D* **with** *g-hom-diff* **and** *hom-completion-closed* [*of g C D*]
show $\mathbf{1}_C = (\mathit{differ}_C)\ (f\ (\pi'\ (g\ x))) \otimes_C f\ ((\delta \otimes_R \pi')\ (g\ x))$ **by** (*unfold hom-diff-def*, *simp*)
next
fix x
assume $x \notin$ *carrier C* **with** *completion-closed2* [*of g C D x*] **and** *g-hom-diff* **have** *g-x*: $g\ x = \mathbf{1}$
by (*unfold hom-diff-def hom-completion-def*, *simp*)
with *hom-completion-one* [*of D D π'*] **and** *π'-in-R* **and** *ring-R* **and** *D.diff-group-is-group*
and *hom-completion-one* [*of D C f*] **and** *C.diff-group-is-group* **and** *f-hom-diff*
and *hom-completion-one* [*of C C differ*$_C$] **and** *diff-group.diff-hom* [*OF C-diff-group*]
and *hom-completion-one* [*of D D* $\delta \otimes_R \pi'$] **and** *pert-in-R* **and** *R.m-closed* [*of* δ π']
show $(\mathit{differ}_C)\ (f\ (\pi'\ (g\ x))) \otimes_C f\ ((\delta \otimes_R \pi')\ (g\ x)) = \mathbf{1}_C$ **by** (*unfold hom-diff-def*, *auto*)
qed
also from *ring-R* **have** ... = (λx. *if* $x \in$ *carrier C then* $(\mathit{differ}_C \circ (f \circ \pi' \circ g))\ x \otimes_C (f \circ \delta \circ \pi' \circ g)\ x$ *else* $\mathbf{1}_C$)
by (*simp add*: *o-assoc expand-fun-eq*)
also from *fπ'g-eq-id* **and** *diff-group.diff-hom* [*OF C-diff-group*]
have ... = (λx. *if* $x \in$ *carrier C then* $(\mathit{differ}_C)\ x \otimes_C (f \circ \delta \circ \pi' \circ g)\ x$ *else* $\mathbf{1}_C$)
by (*unfold hom-completion-def completion-fun2-def completion-def*, *auto simp add*: *expand-fun-eq*)
also from *π'g-eq-psig* **have** ... = (λx. *if* $x \in$ *carrier C then* $(\mathit{differ}_C)\ x \otimes_C (f \circ \delta \circ \Psi \circ g)\ x$ *else* $\mathbf{1}_C$) **by** (*simp add*: *expand-fun-eq*)
finally show $f' \circ (\tau' \circ \mathit{differ}_{\mathit{diff\text{-}group\text{-}ker\text{-}p}'} \circ \tau) \circ g$ = (λx. *if* $x \in$ *carrier C*

then $(differ_C)$ $x \otimes_C (f \circ \delta \circ \Psi \circ g)$ x *else* $\mathbf{1}_C)$
    by *simp*
  qed
qed

By joining *reduction* D' $(\!|$*carrier* = *carrier* C, *mult* = *op* $\otimes_C$, *one* = $\mathbf{1}_C$, *diff* = $f' \circ (\tau' \circ differ_{diff\text{-}group\text{-}ker\text{-}p'} \circ \tau) \circ g|\!)$ $(f' \circ (\tau' \circ \mathbf{1}_R \ominus_R p'))$ (*inc-ker-p'* $\circ\ \tau \circ g$) h' and $f' \circ (\tau' \circ \mathbf{1}_R \ominus_R p') = f \circ \Phi$

inc-ker-p' $\circ\ \tau \circ g = \Psi \circ g$

$f' \circ (\tau' \circ differ_{diff\text{-}group\text{-}ker\text{-}p'} \circ \tau) \circ g = (\lambda x.$ *if* $x \in$ *carrier* C *then* $(differ_C)$ $x \otimes_C (f \circ \delta \circ \Psi \circ g)$ x *else* $\mathbf{1}_C)$ we get the proof of the BPL, stated as in *Lemma 2.2.20* in Aransay's memoir

lemma (**in** *BPL*) *BPL*: **shows** *reduction* D'
  $(\!|$ *carrier* = *carrier* C, *mult* = *mult* C, *one* = *one* C, *diff* = $(\lambda x.$ *if* $x \in$ *carrier* C *then* $(differ_C)$ $x \otimes_C (f \circ \delta \circ \Psi \circ g)$ x *else* $\mathbf{1}_C)|\!)$
  $(f \circ \Phi)$ $(\Psi \circ g)$ h'
using *BPL-reduction* **and** *BPL-simplifications* **by** *simp*

end
theory *Acc-tools* **imports** *Main*
begin

12 Definition of some results about the accesible part of a relation.

term *acc*
thm *accp.intros*

lemma *wf-imp-subset-accP*[*rule-format*]: *wf* $\{(y,x)\ .\ Q\ y \wedge Q\ x \wedge r\ y\ x\} \Longrightarrow (\forall\ y\ x.\ Q\ x \longrightarrow r\ y\ x \longrightarrow Q\ y) \Longrightarrow Q\ x \longrightarrow accp\ r\ x$
  apply (*erule wf-induct*)
  apply *clarify*
  apply (*rule accp.intros*)
  apply (*simp* (*no-asm-use*))
  apply *blast*
  done

lemma *subset-accP-imp-wf*:
  assumes *Q-subset*: $\forall\ x.\ Q\ x \longrightarrow accp\ r\ x$
  shows *wf* $(\{(a,b).\ Q\ b \wedge r\ a\ b\})$
proof −
  let *?s* = $\lambda\ y\ x.\ Q\ x \wedge r\ y\ x$
  have !! $y\ x.$ *?s* $y\ x \Longrightarrow r\ y\ x$ **by** *simp*
  then have *Q-sub-accP-s*: !! $x.\ Q\ x \Longrightarrow accp$ *?s* x
    apply (*rule accp-subset*[*simplified le-fun-def le-bool-def*, *rule-format*])
    apply (*auto simp add*: *Q-subset*)
    done

```
  {
    fix x
    have ¬ (Q x) ⟹ accp ?s x
      apply (rule accp.intros)
      apply auto
      done
    with Q-sub-accP-s have accp ?s x by auto
  }
  note accP-univ = this
  show ?thesis
    apply (rule accp-wfPI[simplified wfP-def])
    apply (simp add: accP-univ)
    done
qed

lemma downchain-contra-imp-subset-accP:
  assumes downchain:⋀ f. (⋀ i. Q (f i)) ⟹ (⋀ i. r (f (Suc i)) (f i)) ⟹ False
  and downclosed: ⋀ y x. ⟦Q x; r y x⟧ ⟹ Q y
  shows Q x ⟹ accp r x
  apply (rule wf-imp-subset-accP[where Q=Q])
  prefer 3
  apply simp
  prefer 2
  apply (rule-tac y=y and x=xa in downclosed)
  apply simp-all
  apply (simp add: wf-iff-no-infinite-down-chain)
  apply (insert downchain)
  apply blast
  done

lemma accP-subset-induct:
  assumes Q-subset: ∀ x. Q x ⟶ accp r x
  assumes Q-downward: ∀ x y. Q x ⟶ r y x ⟶ Q y
  assumes Q-a: Q a
  assumes Q-induct: !! x. Q x ⟹ ∀ y. r y x ⟶ P y ⟹ P x
  shows P a
proof −
  show ?thesis
    apply (rule-tac accp-subset-induct[where x=a and D=Q and R=r])
    apply (simp add: le-fun-def Q-subset le-bool-def)
    apply (auto simp add: Q-a Q-subset)
    apply (erule Q-downward[rule-format])
    apply simp
    apply (erule Q-induct[rule-format])
    apply simp
    done
qed

end
```

theory *Orbit*
imports *Main*
begin

13 Definition of orbits of functions and termination conditions.

lemma *funpow-1*: $(f::'a \Rightarrow 'a)\,\hat{}\,(1::nat) = f$
proof –
  have *1*: $1 = Suc\ 0$ **by** *simp*
  show *?thesis* **by** (*simp add*: *1*)
qed

lemma *funpow-2*: $f\hat{}2 = f\ o\ f$
proof –
  have *wow*: $(2::nat) = (Suc\ (Suc\ 0))$ **by** *auto*
  show *?thesis* **by** (*simp add*: *wow*)
qed

lemma *funpow-zip*: $(f\hat{}n)\ (f\ x) = (f\hat{}(n+1))\ x$
  apply (*induct n*)
  apply *auto*
  done

lemma *funpow-mult*: $(f::'a \Rightarrow 'a)\,\hat{}\,(m*n) = (f\hat{}m)\,\hat{}\,n$
  apply (*induct n*)
  apply *simp*
  apply (*simp add*: *funpow-add*)
  done

lemma *funpow-swap*: $(f\hat{}n)((f\hat{}m)\ x) = (f\hat{}m)((f\hat{}n)\ x)$
  apply (*induct n arbitrary*: *m*)
  apply (*auto simp add*: *funpow-zip*)
  done

lemma *nat-remainder-div*: $0 < (n::nat) \Longrightarrow \exists\ q\ r.\ r < n \wedge m = q * n + r$
  apply (*rule exI*[**where** $x = m\ div\ n$])
  apply (*rule exI*[**where** $x = m\ mod\ n$])
  apply *simp*
  done

lemma *cyclic-fun-range*: **assumes** *n*: $0 < n$ **and** *cycle*: $(f\hat{}n)\ v = v$ **shows** $\exists\ r.$ $r < n \wedge (f\hat{}m)\ v = (f\hat{}r)\ v$
proof –
  from *nat-remainder-div*[**where** $m=m$ **and** $n=n$, *OF n*]
  obtain *q r* **where** *qr*: $r < n \wedge m = q * n + r$ **by** *auto*
  have *q-pow*[*rule-format*]: $((f\hat{}n)\,\hat{}\,q)\ v = v$
    apply (*induct q*)

apply (*auto simp add*: *cycle*)
done
have $(f\hat{\ }m)\ v = (f\hat{\ }r)\ v$
apply (*simp add*: *qr*)
apply (*subst add-commute*)
apply (*simp add*: *funpow-add*)
apply (*subst mult-commute*)
apply (*simp add*: *funpow-mult q-pow*)
done
with *qr* **show** *?thesis* **by** *auto*
qed

13.1 Definition of the orbit of a function over a given point.

constdefs
Orbit :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'a\ set$
Orbit f d $\equiv \{\ (f\hat{\ }n)\ d\ |\ n.\ True\}$

lemma *Orbit-refl*[*simp*]: $x \in$ *Orbit f x*
apply (*simp add*: *Orbit-def*)
apply (*rule-tac x=0* **in** *exI*)
by *simp*

lemma *Orbit-next*[*dest*]: $y \in$ *Orbit f* $(f\ x) \Longrightarrow y \in$ *Orbit f x*
apply (*auto simp add*: *Orbit-def*)
apply (*rule-tac x=Suc n* **in** *exI*)
apply (*simp add*: *funpow-zip*)
done

lemma *Orbit-reduce*:
shows *Orbit f x* $= \{\ (f\hat{\ }m)(x)\ |\ m.\ \forall\ i \le m.\ i > 0 \longrightarrow (f\hat{\ }i)\ x \neq x\ \}$ (**is** *?L* =
?R)
proof −
have *case1*: $?R \subseteq ?L$
by (*auto simp add*: *Orbit-def*)
{
assume *terminates*: $\exists\ n.\ n > 0 \wedge (f\hat{\ }n)\ x = x$
let $?M = LEAST\ m.\ m > 0 \wedge (f\hat{\ }m)\ x = x$
have *M*: $?M > 0 \wedge (f\hat{\ }?M)\ x = x$
apply (*rule LeastI-ex*)
apply (*rule terminates*)
done
{
fix *N*::*nat*
assume *N1*: $N > 0$
assume *N2*: $(f\hat{\ }N)\ x = x$
with *N1 N2* **have** $?M \le N$ **by** (*simp add*: *Least-le*)
}
note *M-least* = *this*

```
    have ?L ⊆ ?R
      apply (auto simp add: Orbit-def)
      proof −
        fix n
        have ? r. r < ?M ∧ (f^n) x = (f^r) x
          apply (rule cyclic-fun-range)
          apply (simp-all add: M)
          done
        then obtain r where r: r < ?M ∧ (f^n) x = (f^r) x by blast
        show ∃ m. (f ^ n) x = (f ^ m) x ∧ (∀ i≤m. 0 < i ⟶ (f ^ i) x ≠ x)
          apply (rule-tac x=r in exI)
          apply (auto simp add: r)
          apply (drule M-least)
          apply auto
          apply (subgoal-tac r < ?M)
          apply simp
          apply (simp add: r)
          done
      qed
  }
  note case2a = this
  {
    assume not-terminates: ¬ (∃ n. n > 0 ∧ (f^n) x = x)
    then have ?L ⊆ ?R
      apply (auto simp add: Orbit-def)
      apply (rule-tac x=n in exI)
      apply auto
      apply (drule-tac x=i in spec)
      apply simp
      done
  }
  note case2b = this
  from case1 case2a case2b show ?thesis by blast
qed

lemma Orbit-unfold1: Orbit f x = {x} ∪ Orbit f (f x)
  apply (auto simp add: Orbit-def)
  apply (rule-tac x=n − 1 in exI)
  apply (case-tac n=0)
  apply (auto simp add: funpow-zip)
  apply (rule-tac x=0 in exI)
  apply simp
  apply (rule-tac x=n+1 in exI)
  apply (simp add: funpow-zip)
  done
```

13.2 Definition of the section of a function over a given point.

constdefs

Section continue (*f*::′*a*⇒′*a*) *x0* ≡ { (*f*ˆ*n*) *x0* | *n*. (∀ *m*. *m* ≤ *n* ⟶ *continue* ((*f*ˆ*m*)(*x0*))) }

lemma *Section-x-x*: ¬ (*continue x*) ⟹ *Section continue f x* = {}
  apply (*simp add*: *Section-def*)
  apply *auto*
  done

lemma *Section-unfold*: *continue i* ⟹ *Section continue f i* = *insert i* (*Section continue f* (*f i*))
  apply (*auto simp add*: *Section-def*)
  apply (*case-tac n*)
  apply *simp*
  apply (*rule-tac x*=*n* − *1* **in** *exI*)
  apply (*simp add*: *funpow-zip*)
  apply *clarsimp*
  apply (*drule-tac x*=*m*+*1* **in** *spec*)
  apply *simp*
  apply (*rule-tac x*=*0* **in** *exI*)
  apply *simp*
  apply (*rule-tac x*=*n*+*1* **in** *exI*)
  apply (*auto simp add*: *funpow-zip*)
  apply (*case-tac m*)
  apply *auto*
  done

lemma *Section-rightopen*[*dest*]: *y* ∈ *Section continue f x* ⟹ *continue y*
  by (*auto simp add*: *Section-def*)

lemma *Section-is-Orbit*:
  assumes *x0-elem-S*: *x0* ∈ *Section continue f* (*f x0*) (**is** *x0* ∈ *?S*)
  shows *?S* = *Orbit f x0*
proof −
  have *x0-noteq-x1*: *continue x0*
    apply (*insert x0-elem-S*)
    apply *blast*
    done
  have ∃ *n*. *n* > *0* ∧ (*f*ˆ*n*) *x0* = *x0* ∧ (∀ *m* ≤ *n*. *continue* ((*f*ˆ*m*) *x0*))
    apply (*insert x0-elem-S*)
    apply (*auto simp add*: *Section-def*)
    apply (*rule-tac x*=*n*+*1* **in** *exI*)
    apply (*auto simp add*: *funpow-zip*)
    apply (*case-tac m*)
    apply *auto*
    done
  then obtain *N* **where** *N*:*N* > *0* ∧ (*f*ˆ*N*) *x0* = *x0* ∧ (∀ *m* ≤ *N*. *continue* ((*f*ˆ*m*) *x0*)) **by** *auto*
  {
    fix *n*::*nat*

assume *n*: *n* > *0*
have *? r*. *r* < *N* ∧ (*f*^*n*) *x0* = (*f*^*r*) *x0*
apply (*rule cyclic-fun-range*)
apply (*simp-all add*: *N*)
done
then obtain *r* **where** *r*: *r* < *N* ∧ (*f*^*n*) *x0* = (*f*^*r*) *x0* **by** *blast*
then have *continue* ((*f*^*n*) *x0*)
apply (*case-tac n=0*)
apply (*insert n*)
apply (*auto simp add*: *N*)
done
}
note *hammer* = *this*
{
fix *n* :: *nat*
note *h* = *hammer*[*of n+1*, *simplified*]
}
note *hammer* = *this*
show *?thesis*
apply (*auto simp add*: *Orbit-def Section-def*)
apply (*rule-tac x=n+1* **in** *exI*)
apply (*simp add*: *funpow-zip*)
apply (*case-tac n=0*)
apply *simp*
apply (*rule-tac x=N* − *1* **in** *exI*)
apply (*auto simp add*: *N funpow-zip hammer*)
apply (*subgoal-tac ? m*. *n* = *Suc m*)
apply *auto*
apply *arith*
done
qed

thm *Section-is-Orbit*

thm *Section-def*

term *continue y* = (∀ *m*. *y* = (*f*^*m*) *x* ⟶ (∀ *n*. *n* > *0* ∧ *n* ≤ *m* ⟶ (*f*^*m*) *x* ≠ *x*))

lemma *Section-is-Orbit′*: *insert x* (*Section* (λ*y*. *y* ≠ *x*) *f* (*f x*)) = *Orbit f x*
apply *auto*
apply (*simp add*: *Section-def Orbit-def*)
apply *clarsimp*
apply (*rule-tac x=n+1* **in** *exI*)
apply (*simp add*: *funpow-zip*)
apply (*auto simp add*: *Section-def Orbit-reduce*)
apply (*case-tac m*)
apply (*simp-all add*: *funpow-zip*)

```
  apply (drule-tac x=nat in spec)
  apply clarsimp
  apply (case-tac ma = nat)
  apply auto
  done
```

13.3 Definition of a termination condition in terms of orbits.

```
fun terminates :: ('a ⇒ bool) × ('a ⇒ 'a) × 'a ⇒ bool
where
  terminates (continue, f, x) = (∃ y. (y ∈ Orbit f x) ∧ ¬ (continue y))

declare terminates.simps[simp del]

lemmas terminates-simp = terminates.simps

lemma finite-Section:
  assumes terminates: terminates (continue, f, x)
  shows finite (Section continue f x)
proof −
  have ? N. ¬ continue ((f^N) x)
    apply (insert terminates)
    apply (auto simp add: terminates-simp Orbit-def)
    done
  then obtain N where N: ¬ continue ((f^N) x) by auto
  have Section continue f x ⊆ image (λ n. (f^n) x) {..N}
    apply (auto simp add: Section-def image-def Bex-def)
    apply (rule-tac x=n in exI)
    apply simp
    apply (drule-tac x=N in spec)
    apply (auto simp add: N)
    done
  note finite-sub = finite-subset[OF this, simplified]
  then show ?thesis by blast
qed

lemma terminates-imp-notin-Section:
  assumes terminates: terminates (continue, f, x)
  shows x ∉ Section continue f (f x)
proof −
  {
    assume x ∈ Section continue f (f x)
    then have SO: Section continue f (f x) = Orbit f x
      by (rule Section-is-Orbit)
    from terminates[simplified terminates-simp]
    obtain y where y: y ∈ Orbit f x ∧ ¬ continue y by blast
    have y ∉ Orbit f x
      by (auto simp add: SO[symmetric] y)
    with y have False
```

```
    by auto
  }
  then show ?thesis by auto
qed

lemma orbit-stepback: i ≠ x ⟹ (x ∈ Orbit f (f i)) = (x ∈ Orbit f i)
  apply (auto simp add: Orbit-def)
  apply (rule-tac x = n+1 in exI)
  apply (simp add: funpow-zip)
  apply (case-tac n)
  apply (auto simp add: funpow-zip)
  done

lemma terminates-rec: terminates (continue, f, x) = (if continue x then terminates
(continue, f, f x) else True)
  apply (auto simp add: terminates-simp)
  apply (rule-tac x=y in exI)
  apply (simp add: Orbit-unfold1[of f x])
  apply clarsimp
  apply (rule-tac x=x in exI)
  apply simp
  done

lemma Suc-card-Section-eq:
  assumes x0-neq-x1: continue x0
  and terminates: terminates (continue, f, f x0)
  and finite-A: finite A
  and x0-elem-A: x0 ∈ A
  shows Suc (card (Section continue f (f x0) ∩ A)) = card (Section continue f x0
∩ A)
proof −
  have insert: insert x0 (Section continue f (f x0) ∩ A) = Section continue f x0
∩ A (is ?L=?R)
    by (auto simp add: Section-unfold[of continue, OF x0-neq-x1] x0-elem-A)
  have x0-notin-Section: x0 ∉ Section continue f (f x0)
    apply (rule terminates-imp-notin-Section)
   apply (simp add: terminates-rec[where x=x0 and continue=continue] x0-neq-x1
terminates)
    done
  from x0-notin-Section finite-A  have card-L: card ?L = Suc (card (Section con-
tinue f (f x0) ∩ A))
    by simp
  then show ?thesis
    by (simp add: card-L[symmetric] insert)
qed

end
theory Cycle imports Main
begin
```

constdefs
  closed :: *'a set* ⇒ (*'a* ⇒ *'a*) ⇒ *bool*
  closed A f ≡ ∀ *x* ∈ *A*. *f x* ∈ *A*

constdefs
  cyclic :: *'a set* ⇒ (*'a* ⇒ *'a*) ⇒ *bool*
  cyclic A f ≡ ∀ *d* ∈ *A*. ∃ *n*. *n* > *0* ∧ (*f^n*) *d* = *d*

constdefs
  cyclic-equiv :: *'a set* ⇒ (*'a* ⇒ *'a*) ⇒ (*'a* × *'a*) *set*
  cyclic-equiv A f ≡ { (*a*,*b*) . *a* ∈ *A* ∧ *b* ∈ *A* ∧ (*? n*. (*f^n*) *a* = *b*) }

lemma *refl-cyclic-equiv*: *refl A* (*cyclic-equiv A f*)
  apply (*auto simp add*: *cyclic-equiv-def refl-def*)
  apply (*rule exI*[**where** *x*=*0*])
  apply *simp*
  done

lemma *archimedian-law*: (*m*::*nat*) > *0* ⟹ *? q*. *n* < *q*∗*m*
  apply (*induct n*)
  apply *auto*
  apply (*rule-tac x*=*q*+*1* **in** *exI*)
  apply *simp*
  done

lemma *cyclic-wrap*:
    assumes *c*: *cyclic A f*
    assumes *x*: *x* ∈ *A*
    shows *? n'*. (*f^n'*) ((*f^n*) *x*) = *x*
proof −
  from *c x* **have** *? m*. *m* > *0* ∧ (*f^m*)(*x*) = *x* **by** (*auto simp add*: *Ball-def cyclic-def*)
  then obtain *m* **where** *m*: *m* > *0* ∧ (*f^m*)(*x*) = *x* **..**
  with *archimedian-law* **have** *? q*. *n* < *q*∗*m* **by** *auto*
  then obtain *q* **where** *n* < *q*∗*m* **..**
  then have *q*: *q*∗*m* − *n* + *n* = *q* ∗ *m* **by** *simp*
  show *? n'*. (*f^n'*)((*f^n*) *x*) = *x*
    apply (*rule exI*[**where** *x*=*q*∗*m*−*n*])
    apply (*simp only*: *o-apply*[**where** *f*=*f* ^ (*q* ∗ *m* − *n*) **and** *g*=(*f* ^ *n*), *symmetric*] *funpow-add*[*symmetric*] *q*)
    apply (*induct q*)
    apply (*simp-all add*: *funpow-add m*)
    done
qed

lemma *sym-cyclic-equiv*: *cyclic A f* ⟹ *sym* (*cyclic-equiv A f*)
  by (*auto simp add*: *sym-def cyclic-equiv-def cyclic-wrap*)

lemma *trans-cyclic-equiv*: *trans* (*cyclic-equiv A f*)
  apply (*auto simp add*: *cyclic-equiv-def trans-def*)
  apply (*rule-tac x=na+n* **in** *exI*)
  apply (*simp add*: *funpow-add*)
  done

lemma *cyclic A f* $\Longrightarrow$ *equiv A* (*cyclic-equiv A f*)
  by (*simp add*: *equiv-def refl-cyclic-equiv sym-cyclic-equiv trans-cyclic-equiv*)

constdefs
  cyclic-on A f $\equiv$ *closed A f* $\wedge$ *cyclic A f*

constdefs
  representing A f R $\equiv$ *closed A f* $\wedge$ *cyclic A f* $\wedge$ $R \subseteq A \wedge (\forall\ x\ y.\ (x \in R \wedge y \in R \wedge (\exists\ n.\ (f\hat{}n)\ (x) = y)) \longrightarrow x = y) \wedge A = \{\ (f\hat{}n)\ (x) \mid n\ x.\ x \in R\ \}$

end
theory *While* **imports** *Acc-tools Orbit Cycle*
begin

14 Definition of *while* loops as tail recursive functions.

function (*tailrec*) *While* :: ($'a \Rightarrow$ *bool*) $\Rightarrow$ ($'a \Rightarrow 'a$) $\Rightarrow 'a \Rightarrow 'a$
where
  While continue f s = (*if continue s then While continue f* (*f s*) *else s*)
by *auto*

We delete the definition from the simplifier to avoid infinite loops.

declare *While.simps*[*simp del*]
lemmas *While-simp* = *While.simps*

An alternative definition for termination sets.

fun *terminates-slice* :: ($'a \Rightarrow$ *bool*) $\Rightarrow$ ($'a \Rightarrow 'a$) $\Rightarrow$ ($'a \Rightarrow$ *bool*) $\times$ ($'a \Rightarrow 'a$) $\times$ $'a \Rightarrow$ *bool*
where
  terminates-slice continue0 f0 (*continue*, *f*, *x*) = (*continue* = *continue0* $\wedge$ *f* = *f0* $\wedge$ *terminates* (*continue*, *f*, *x*))

lemma *lfp-const*: *lfp* (λ *p*. *P*) = *P*
proof –
  have *mono*: *mono* (λ *p*. *P*) **by** (*simp add*: *mono-def*)
  then show *?thesis*
    by (*simp add*: *lfp-unfold*[*OF mono*])
qed

Definition of the relation condition of *While* loops.

lemmas *While-rel-def'* = *While-rel-def*[*simplified lfp-const*]

lemma *While-rel-continue*: *While-rel q* (*continue*, *f*, *s*) $\Longrightarrow$ *continue s*
apply (*erule While-rel.cases*)
apply *simp*
done

lemma assumes *n-g-0*: *0* < (*n*::*nat*) **shows** $\exists$ *m*. *n* = *Suc m*
using *n-g-0* **by** *arith*

lemma *helper*: **shows** $\bigwedge$*n*::*nat*. *0* < *n* ==> $\exists$ *m*. *n* = *Suc m* **by** *arith*

lemma *terminates-downward*: *terminates x* $\Longrightarrow$ *While-rel y x* $\Longrightarrow$ *terminates y*
apply (*auto simp add*: *While-rel-def′ terminates-simp*)
apply (*auto simp add*: *Orbit-def*)
apply (*rule-tac x*=(*f*^*n*) *s* **in** *exI*)
apply (*case-tac n*=*0*)
apply *simp*
apply *simp*
apply (*subgoal-tac ? m*. *n* = *Suc m*)
apply *auto*
apply (*rule-tac x*=*m* **in** *exI*)
apply (*simp add*: *funpow-zip*)
apply (*simp add*: *helper*)
done

lemma *terminates-slice-downward*: *terminates-slice continue f x* $\Longrightarrow$ *While-rel y x* $\Longrightarrow$ *terminates-slice continue f y*
apply (*cases x*, *cases y*) **apply** *auto*
unfolding *While-rel-def′* **apply** *auto* **using** *terminates-rec* [*of continue f* -] **by** *auto*

lemma *terminates-subset-dom*[*rule-format*]: *terminates* (*continue*, *f*, *s*) $\longrightarrow$ *While-dom* (*continue*, *f*, *s*)
proof −
let *?Q* = *terminates*
let *?r* = *While-rel*
note *Q* = *terminates-downward*
{
fix *F*
assume *F-Q*: !! *i*::*nat*. *?Q* (*F i*)
assume *F-r*: !! *i*::*nat*. *?r* (*F* (*Suc i*)) (*F i*)
{ fix *i*
have *? continue f s*. *F i* = (*continue*, *f*, *s*)
apply (*cases F i*)
apply *simp*
done
}
note *F-split* = *this*

```
{
  fix continue0 f0 s0
  assume F0:F 0 = (continue0, f0, s0)
  {
    fix i
    fix continue f s
    have F i = (continue, f, s) ⟹ continue s
      apply (subgoal-tac ?r (F (Suc i)) (F i))
      apply simp
      apply (rule While-rel-continue[where q=(F (Suc i)) and f=f])
      apply simp
      apply (rule F-r)
      done
  }
  note continue = this
  {
    fix i
    have ∀ continue f s. F i = (continue, f, s) ⟶ s = (f0^i) s0 ∧ f = f0 ∧
continue = continue0
      apply (induct i)
      apply (simp add: continue)
      apply (simp add: F0)
      apply clarsimp
      apply (subgoal-tac ? continue f s. F i = (continue, f, s))
      prefer 2
      apply (simp add: F-split)
      apply clarsimp
      apply (subgoal-tac ?r (F (Suc i)) (F i))
      prefer 2
      apply (rule F-r)
      apply (simp add: While-rel-def')
      done
  }
  note calc-F = this
  {
    fix n
    have continue0 ((f0^n) s0)
      apply (insert calc-F[of n])
      apply (cases F n)
      apply simp
      apply (simp add: continue)
      done
  }
  note no-Orbit = this
  have Orbit: ∃ y0 ∈ Orbit f0 s0. ¬ (continue0 y0)
    apply (insert F0)
    apply (insert F-Q[of 0])
    apply (simp add: terminates-simp Bex-def)
    done
```

then have ∃ *n*. ¬ (*continue0* ((*f0*^*n*) *s0*))
by (*auto simp add*: *Orbit-def*)
with *no-Orbit* **have** *False* **by** *auto*
}
then have *False*
apply (*cases F 0*)
apply *simp*
apply *blast*
done
}
note *r* = *this*
show *?thesis*
apply (*rule-tac impI*)
apply (*rule downchain-contra-imp-subset-accP*[**where** *Q*=*?Q*])
prefer *3*
apply *simp*
apply (*drule r*)
apply *simp*
apply *simp*
apply (*rule Q*)
apply *simp-all*
done
qed

lemma *terminates-slice-subset-dom*: *terminates-slice continue f x* ⟹ *While-dom x*
apply (*cases x*)
apply *simp*
apply (*rule terminates-subset-dom*)
apply (*auto*)
done

Some additional induction rules for the previous *While* definition.

lemma *While-pinduct*:
assumes *terminates*: *terminates* (*continue*, *f*, *s*)
and *I*: !! *s*. ⟦ *terminates* (*continue*, *f*, *s*); *continue s* ⟹ *P* (*f s*)⟧ ⟹ *P s*
shows *P s*
proof −
show *?thesis*
apply (*subgoal-tac P s* = (λ (*continue*, *f*, *s*). *P s*) (*continue*, *f*, *s*))
prefer *2*
apply *clarify*
apply (*simp only*:)
apply (*rule accP-subset-induct*[**where** *r*=*While-rel* **and** *Q*=*terminates-slice continue f*])
apply (*simp add*: *terminates-slice-subset-dom*)
apply (*simp add*: *terminates-slice-downward*)
apply (*simp add*: *terminates*)
apply *clarsimp*

```
    apply (simp add: While-rel-def' I)
    done
qed

lemma While-pinduct-weak:
  assumes terminates: terminates (continue, f, s)
  and I: !! s. ⟦ continue s ⟹ P (f s)⟧ ⟹ P s
  shows P s
  apply (rule While-pinduct[where P=P])
  apply (rule terminates)
  apply (rule I)
  apply simp
  done

lemma While-hoare-total:
  assumes wf-R: wf R
  and R-down: !! x. P x ⟹ continue x ⟹ (f x, x) ∈ R
  and P-cont: !! x. P x ⟹ continue x ⟹ P (f x)
  and P-not-cont: !! x. P x ⟹ ¬ (continue x) ⟹ Q x
  and P-start: P s
  shows Q (While continue f s)
proof −
  {
    fix x
    assume Px: P x
    assume ¬ (terminates (continue, f, x))
   then have continue: !! n. continue ((f^n) x) by (auto simp add: terminates-simp
Orbit-def)
    {
      fix n
      have P: P ((f^n) x)
        apply (induct n)
        apply (simp add: Px)
        apply (simp)
        apply (rule P-cont)
        apply (simp-all add: continue)
        done
      note R = R-down[OF P, OF continue]
    }
    then have ¬ (wf R)
      apply (simp add: wf-iff-no-infinite-down-chain)
      apply (rule-tac x=λ n. (f^n) x in exI)
      apply (auto)
      done
    with wf-R have False by auto
  }
  then have !! x. P x ⟹ terminates (continue, f, x) by auto
  with P-start have terminates: terminates (continue, f, s) by auto
  have P s ⟶ Q (While continue f s)
```

proof (*induct rule*: *While-pinduct-weak*[*OF terminates*])
case (*1 x*)
show *?case*
apply (*subst While-simp*)
apply (*case-tac continue x*)
apply (*auto simp add*: *P-not-cont*)
apply (*rule 1*[*rule-format*])
apply (*simp-all add*: *P-cont*)
done
qed
then show *?thesis* **by** (*simp add*: *P-start*)
qed

15 Definition of *For* loops.

constdefs
For′ :: (′*a* ⇒ *bool*) ⇒ (′*a* ⇒ ′*a*) ⇒ (′*a* ⇒ ′*b* ⇒ ′*b*) ⇒ ′*a* ⇒ ′*b* ⇒ (′*b* × ′*a*)
For′ continue f Ac x ac ≡ *While* (λ (*ac*, *x*). *continue x*) (λ (*ac*, *x*). (*Ac x ac*, *f x*)) (*ac*, *x*)
For continue f Ac x ac ≡ *fst* (*For′ continue f Ac x ac*)

term *For*

lemma *For′-simp*: *For′ continue f Ac x ac* = (*if continue x then For′ continue f Ac* (*f x*) (*Ac x ac*) *else* (*ac*, *x*))
proof −
have *trivial1*: *split* (λ*ac*. *continue*) (*ac*, *x*) = *continue x*
by *simp*
have *trivial2*: ((λ(*ac*, *x*). (*Ac x ac*, *f x*)) (*ac*, *x*)) = (*Ac x ac*, *f x*)
by *simp*
show *?thesis*
apply (*subst For′-def*)
thm *While-simp*
apply (*subst For′-def*)
apply (*subst While-simp*[*of* (λ (*ac*, *x*). *continue x*) (λ (*ac*, *x*). (*Ac x ac*, *f x*)) (*ac*, *x*)])
apply (*simp only*: *trivial1 trivial2*)
done
qed

lemma *For-simp*: *For continue f Ac x ac* = (*if continue x then For continue f Ac* (*f x*) (*Ac x ac*) *else ac*)
proof −
note *unfold-For′* = *For′-simp*[*of continue f Ac x ac*]
show *?thesis* **by** (*auto simp add*: *For-def unfold-For′*)
qed

lemma *split-power-of-for-state*: *? ac′*. (((λ(*ac*, *x*). ((*Ac* :: ′*a* ⇒ ′*b* ⇒ ′*b*) *x ac*, (*f* :: ′*a* ⇒ ′*a*) *x*)) ˆ *n*) (*a*, *b*) = (*ac′*, (*f*ˆ*n*) *b*))

apply (*induct n*)
apply *auto*
done

lemma *swap-Ex*: (∃ *a b*. *P a b*) = (∃ *b a*. *P a b*) **by** *blast*

lemma *terminates-For*: *terminates* (λ(*ac*, *x*). *continue x*, λ(*ac*, *x*). (*Ac x ac*, *f x*), *s*) = *terminates* (*continue*, *f*, *snd s*)
apply (*cases s*)
apply *simp*
apply (*auto simp add*: *terminates-simp Bex-def Orbit-def*)
apply (*rule-tac x=ba* **in** *exI*)
apply *clarsimp*
apply (*rule-tac x=n* **in** *exI*)
apply (*subgoal-tac ? ac′*. (((λ(*ac*, *x*). (*Ac x ac*, *f x*)) ^ *n*) (*a*, *b*) = (*ac′*, (*f*^*n*) *b*)))
apply *clarsimp*
apply (*rule split-power-of-for-state*)
apply (*subst swap-Ex*)
apply (*rule-tac x*=(*f*^*n*) *b* **in** *exI*)
apply *clarsimp*
apply (*subst swap-Ex*)
apply (*rule-tac x=n* **in** *exI*)
apply (*subgoal-tac* ∃ *aa*. ((λ(*ac*, *x*). (*Ac x ac*, *f x*)) ^ *n*) (*a*, *b*) = (*aa*, (*f* ^ *n*) *b*))
prefer *2*
apply (*rule split-power-of-for-state*)
apply *clarsimp*
done

Additional induction rules for *For* loops.

lemma *For-pinduct*:
assumes *terminates*: *terminates* (*continue*, *f*, *i*)
and *I*: ⋀*i s*. ⟦*terminates* (*continue*, *f*, *i*); *continue i* ⟹ *P* (*f i*) (*Ac* (*i*::′*a*) (*s*::′*b*::*type*))⟧ ⟹ *P i s*
shows *P i s*
apply (*rule While-pinduct*[*of* (λ (*ac*, *x*). *continue x*) (λ (*ac*, *x*). (*Ac x ac*, *f x*)), *simplified terminates-For*, **where** *s*=(*s*,*i*) **and** *P* = λ (*s*,*i*). *P i s*, *simplified*])
apply (*simp add*: *terminates*)
apply (*subgoal-tac ? s′ i′*. *s* = (*s′*, *i′*))
prefer *2*
apply *simp*
apply (*auto simp add*: *I*)
done

lemma *For-pinduct-weak*:
assumes *terminates*: *terminates* (*continue*, *f*, *i*)
and *I*: ⋀*i s*. ⟦*continue i* ⟹ *P* (*f i*) (*Ac* (*i*::′*a*) (*s*::′*b*::*type*))⟧ ⟹ *P i s*
shows *P i s*
apply (*rule For-pinduct*[**where** *P*=*P*])

apply (*rule terminates*)
apply (*rule I*)
apply *simp*
done

constdefs
find f x ≡ *While* (λ *x*. *f x* ≠ *x*) *f x*
step1 f ≡ {(*y*,*x*). *y* = *f x* ∧ *y* ≠ *x*}

thm *find-def*[*symmetric*]
thm *While-simp*

lemma *find-simp*: *find f x* = (*if f x* ≠ *x then find f* (*f x*) *else x*)
apply (*simp only*: *find-def*)
apply (*rule While-simp*)
done

lemma *find-wf-step1-terminates*: *wf* (*step1 f*) ⟹ *terminates* (λ *x*. *f x* ≠ *x*, *f*, *i*)
apply (*erule wf-induct*)
apply (*auto simp add*: *step1-def terminates-simp*)
done

lemmas *find-pinduct* = *While-pinduct-weak*[*of* (λ *x*. *f x* ≠ *x*) *f x*]

lemma *find*:
assumes *terminates*: *terminates* (λ *x*. *f x* ≠ *x*, *f*, *x*)
shows *f*(*find f x*) = *find f x*
proof (*induct rule*: *find-pinduct*)
case *1*
show *?case* **by** (*simp add*: *terminates*)
case (*2 x*)
show *?case*
by (*auto simp add*: *find-simp*[*of f x*] *2*)
qed

constdefs
section continue f x0 ≡ *For continue f insert x0*
card-section continue f x0 ≡ *For continue f* (λ *x y*. *y* + (*1*::*nat*)) *x0*

lemmas *section-pinduct*=*For-pinduct*[**where** *Ac*=*insert*]
lemmas *section-pinduct-weak*=*For-pinduct-weak*[**where** *Ac*=*insert*]
lemmas *card-section-pinduct-weak*=*For-pinduct-weak*[**where** *Ac*=λ *x y*. *y* + (*1*::*nat*)]

lemma *section-simp*: *section continue f x A* = (*if continue x then section continue f* (*f x*) (*insert x A*) *else A*)
apply (*simp only*: *section-def*)
apply (*rule For-simp*)

done

lemma *card-section-simp*:*card-section continue f x A* = (*if continue x then card-section continue f* (*f x*) (*A* + *1*) *else A*)
apply (*simp only*: *card-section-def*)
apply (*rule For-simp*)
done

lemma *section-is-Section*:
assumes *terminates*: *terminates* (*continue*, *f*, *x0*)
shows *section continue f x0 A* = *A* ∪ (*Section continue f x0*)
proof (*induct rule*: *section-pinduct-weak*[**where** *f*=*f* **and** *continue*=*continue* **and** *i*=*x0* **and** *P*=λ *x0 A*. *section continue f x0 A* = *A* ∪ (*Section continue f x0*)])
case *1*
show *?case* **by** (*simp add*: *terminates*)
case (*2 i A*)
{
assume *i-eq-x1*: ¬ *continue i*
have *e*: *Section continue f i* = {} **by** (*simp add*: *i-eq-x1 Section-x-x*)
have *?case*
apply (*simp add*: *e*)
apply (*subst section-simp*)
apply (*simp add*: *i-eq-x1*)
done
}
note *eq* = *this*
{
assume *i-neq-x1*: *continue i*
have *?case*
apply (*subst section-simp*)
apply (*simp add*: *i-neq-x1 2*[*OF i-neq-x1*])
apply (*simp only*: *Un-insert-right*[*symmetric*])
apply (*simp add*: *Section-unfold*[**where** *continue*=*continue*, *OF i-neq-x1*])
done
}
note *neq* = *this*
from *eq neq* **show** *?case* **by** *auto*
qed

constdefs
orbit f x ≡ *section* (λ *y*. *y* ≠ *x*) *f* (*f x*) {*x*}
card-orbit f x ≡ *card-section* (λ *y*. *y* ≠ *x*) *f* (*f x*) *1*

lemma *terminates-implies*: *terminates* (*cond*, *f*, *x*) ⟹ ∃ *n*. ¬ (*cond* ((*f*ˆ*n*) *x*))
by (*auto simp add*: *terminates-simp Orbit-def*)

lemma *orbit-is-Orbit*:
assumes *terminates*: *terminates* (λ *y*. *y* ≠ *x*, *f*, *f x*)
shows *orbit f x* = *Orbit f x*

by (*simp add*: *orbit-def section-is-Section*[*OF terminates*] *Section-is-Orbit′*)

lemma *card-section-add*:
assumes *terminates*: *terminates* (*continue*, *f*, *x0*)
shows *card-section continue f x0* (*a* + *b*) = *a* + (*card-section continue f x0* (*b*::*nat*))
proof (*induct x0 b arbitrary*: *a rule*: *card-section-pinduct-weak*[**where** *continue*=*continue* **and** *f*=*f*])
case *1*
show *?case* **by** (*rule terminates*)
case (*2 i y x*)
show *?case*
by (*auto simp add*: *card-section-simp*[**where** *x0*=*i*] *2*[*of x*, *simplified*])
qed

lemma *card-section-suc*:
assumes *terminates*: *terminates* (*continue*, *f*, *x0*)
shows *card-section continue f x0* (*Suc a*) = *Suc* (*card-section continue f x0 a*)
by (*rule card-section-add*[*OF terminates*, **where** *a*=*1* **and** *b*=*a*, *simplified*])

lemma *terminates-imp*: *terminates* (*continue*, *f*, *i*) ⟹ *continue i* ⟹ *terminates* (*continue*, *f*, *f i*)
by (*simp add*: *terminates-rec*[**where** *x*=*i*])

lemma *Suc-first*: *Suc* (*a* + *b*) = *Suc a* + *b* **by** *simp*

lemma *card-section-eq*[*rule-format*]:
terminates (*continue*, *f*, *x0*) ⟹ *finite A* ⟶ *card-section continue f x0* (*card A*) = *card* (*section continue f x0 A*) + *card* (*Section continue f x0* ∩ *A*)
apply (*rule section-pinduct*[**where** *P*=λ *x0 A*. *finite A* ⟶ *card-section continue f x0* (*card A*) = *card* (*section continue f x0 A*) + *card* (*Section continue f x0* ∩ *A*)])
apply (*assumption*)
apply (*subst section-simp*)
apply (*subst card-section-simp*)
apply (*case-tac* ¬ (*continue i*))
apply *clarsimp*
apply (*simp add*: *Section-x-x*)
apply *clarsimp*
apply (*case-tac i* ∈ *s*)
apply (*simp add*: *insert-absorb*)
apply (*subst card-section-suc*)
apply (*subgoal-tac terminates* (*continue*, *f*, *i*) = (*if continue i then terminates* (*continue*, *f*, *f i*) *else True*))
apply *simp*
apply (*rule terminates-rec*)
apply *simp*
apply (*subst Suc-card-Section-eq*)

apply *simp-all*
apply (*subgoal-tac terminates* (*continue*, *f*, *i*) = (*if continue i then terminates* (*continue*, *f*, *f i*) *else True*))
apply *simp*
apply (*rule terminates-rec*)
apply (*case-tac i* ∈ *Section continue f* (*f i*))
apply (*simp add*: *terminates-imp-notin-Section*)
apply (*drule Section-unfold*[**where** *f*=*f* **and** *continue*=*continue*])
apply (*subgoal-tac insert i* (*Section continue f* (*f i*)) ∩ *s* = *Section continue f* (*f i*) ∩ *insert i s*)
apply *simp*
apply *auto*
done

lemma
assumes *terminates*: *terminates* (*continue*, *f*, *x*)
shows *card-section continue f x 0* = *card* (*Section continue f x*)
by (*simp add*: *card-section-eq*[*OF terminates*, *of* {}, *simplified*] *section-is-Section*[*OF terminates*])

constdefs
orbit-terminates f x x′ ≡ *terminates* (λ *y*. *y* ≠ *x*, *f*, *f x′*)

lemma *card-orbit-is-card-Orbit*:
assumes *terminates*: *orbit-terminates f x x*
shows *card-orbit f x* = *card* (*Orbit f x*)
apply (*simp add*: *card-orbit-def*)
apply (*simp add*: *orbit-is-Orbit*[*OF terminates*[*simplified orbit-terminates-def*], *symmetric*])
apply (*auto simp add*: *card-section-eq*[*OF terminates*[*simplified orbit-terminates-def*], **where** *A*={*x*}, *simplified*] *orbit-def*)
done

lemma *orbit-terminates-rec*: *orbit-terminates f x x′* = (*if f x′* ≠ *x then orbit-terminates f x* (*f x′*) *else True*)
apply (*simp only*: *orbit-terminates-def*)
apply (*rule terminates-rec*)
done

constdefs
fold-section-def: *fold-section continue h f g z a* == *For continue h* (λ *z a*. *f* (*g z*) *a*) *z a*

lemma *finite-Orbit*:
  assumes *terminates*: *orbit-terminates f a a*
  shows *finite* (*Orbit f a*)
proof −
  from *terminates* **show** *?thesis*
    apply (*simp add*: *Section-is-Orbit'*[*symmetric*])
    apply (*rule finite-Section*)
    apply (*simp add*: *orbit-terminates-def*)
    done
qed

lemma
  fold-section-simp:
    fold-section continue h f g x ac =
    (*if continue x*
     then fold-section continue h f g (*h x*) (*f* (*g x*) *ac*) *else ac*)
  apply (*subst fold-section-def*)+
  apply (*rule For-simp*)
  done

The following locale is simply a rewriting, or an interpretation, of *ab-semigroup-mult*, and is only used to use f as a a binary operation instead of *op* $*$

locale *ACf* =
  fixes *f* :: *'a => 'a => 'a*   (**infixl** · *70*)
  assumes *commute*: $x \cdot y = y \cdot x$
    and *assoc*: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
begin

lemma *left-commute*: $x \cdot (y \cdot z) = y \cdot (x \cdot z)$
proof −
  have $x \cdot (y \cdot z) = (y \cdot z) \cdot x$ **by** (*simp only*: *commute*)
  also have ... $= y \cdot (z \cdot x)$ **by** (*simp only*: *assoc*)
  also have $z \cdot x = x \cdot z$ **by** (*simp only*: *commute*)
  finally show *?thesis* **.**
qed

lemmas *AC* = *assoc commute left-commute*

end

interpretation *ACf* ⊆ *ab-semigroup-mult f* (**infixl** · *70*)
by *unfold-locales* (*simp-all add*: *AC*)

lemma (**in** *ACf*) *fold-Section-eq*:
  assumes *terminates*: *terminates* (*continue*, *h*, *z*)
  shows *fold f g a* (*Section continue h z*) = *fold-section continue h f g z a*
proof −
  let *?Ac* = λ *z a*. *f* (*g z*) *a*

show *?thesis*
proof (*rule For-pinduct* [**where** *P*=λ *z a*. *fold f g a* (*Section continue h z*) = *fold-section continue h f g z a* **and** *Ac*=*?Ac*])
show *terminates* (*continue*, *h*, *z*) **using** *terminates* **by** *simp*
fix *i s*
assume *termin*: *terminates* (*continue*, *h*, *i*)
and *continue*: *continue i* ⟹ *fold op* · *g* (*g i* · *s*) (*Section continue h* (*h i*)) = *fold-section continue h op* · *g* (*h i*) (*g i* · *s*)
show *fold op* · *g s* (*Section continue h i*) = *fold-section continue h op* · *g i s*
proof (*cases continue i*)
case *False* **with** *Section-x-x* [*of continue i h*] **show** *?thesis* **using** *fold-section-simp* [*of continue h f g i s*] **by** *auto*
next
case *True* **show** *?thesis*
unfolding *fold-section-simp* [*of continue h f g i s*]
using *True* **apply** *simp*
unfolding *Section-unfold* [*of continue i h*]
unfolding *sym* [*OF continue*]
using *finite-Section* [*OF terminates-imp* [*OF termin*]] **using** *True* **apply** *simp*
using *fold-commute* [*of* (*Section continue h* (*h i*)) *g i g s*] **using** *terminates-imp-notin-Section* [*OF termin*]
using *fold-insert* [*of* (*Section continue h* (*h i*)) *i g s*] **by** *simp*
qed
qed
qed

lemma (**in** *ACf*) *fold-Orbit-eq*:
assumes *terminates*: *orbit-terminates h z z*
shows *fold f g a* (*Orbit h z*) = *fold-section* (λ *y*. *y* ≠ *z*) *h f g* (*h z*) (*f* (*g z*) *a*)
proof −
note *t* = *terminates*[*simplified orbit-terminates-def*]
show *?thesis*
apply (*subst fold-Section-eq*[*symmetric*])
prefer *2*
apply (*subst Section-is-Orbit'*[*symmetric*])
apply (*subst fold-commute*[*symmetric*])
prefer *2*
apply (*subst fold-insert*)
prefer *3*
apply (*simp-all add*: *t finite-Section*)
apply (*rule terminates-imp-notin-Section*)
apply (*subst terminates-rec*)
apply *simp*
done
qed

lemma *While-postcondition*:
assumes *terminates*: *terminates* (*continue*, *f*, *x*)

shows ¬ (*continue* (*While continue f x*))
apply (*rule While-pinduct-weak*[*OF terminates*])
apply (*subst While-simp*)
apply (*case-tac continue s*)
apply *simp-all*
done

end

theory *BPL-classes-2008*
imports
Basic-Perturbation-Lemma-local-nilpot
While
begin

16 Additional type classes

In this section we introduce some additional type classes to those provided by the Isabelle standard distribution

For instance, we need a class *diff-group-add* that can be defined from the *ab-group-add* type class from the Isabelle library:

class *diff-group-add* = *ab-group-add* +
fixes *diff* :: *'a* => *'a* (*d* - [*81*] *80*)
assumes *diff-hom*: *d* (*x* + *y*) = (*d x*) + (*d y*)
and *diff-nilpot*: *diff* ∘ *diff* = (λ*x*. *0*)

lemma (**in** *diff-group-add*) [*simp*]: *d* (*d x*) = *0*
using *diff-nilpot*
unfolding *expand-fun-eq* **by** *simp*

According to the previous syntax definitions, *diff-group-add-class.diff* is to be used with the parameter over which it is applied, and *diff-group-add-class.diff* remains to be used as a function

We can indeed prove instances of the specified type classes. An instance of a type class makes the type class sound.

instantiation *int* :: *diff-group-add*
begin

definition *diff-int-def*: *diff* ≡ (λ*x*. *0*::*int*)

instance
proof
show *diff* ∘ *diff* = (λ*x*::*int*. *0*)
unfolding *diff-int-def*
unfolding *expand-fun-eq* **by** *simp*

```
  fix x y ::int
  show d (x + y) = (d x) + (d y)
    unfolding diff-int-def by arith
qed

end
```

A limitation of type classes can be observed in the following definition; using the *op* + symbol for *fun* is not possible. In a type class definition, symbols only refer to the type class being defined.

The following type class definition is not valid; the + operation can only be used for the type class being defined

The following type class represents a differential group and a perturbation over it.

```
class diff-group-add-pert = diff-group-add +
  fixes pert :: 'a ⇒ 'a (δ - [81] 80)
  assumes pert-hom-ab: δ (a + b) = δ a + δ b
  and pert-preserv-diff-group-add:
  diff-group-add (op −) (λx. − x) 0 (op +) (λx. d x +  δ x)

instantiation int :: diff-group-add-pert
begin

definition pert-int-def: pert ≡ (λx. 0::int)

instance proof
  fix a b :: int
  show δ (a + b) = δ a + δ b
    unfolding pert-int-def by simp
next
  show diff-group-add op − uminus (0::int) op + (λx. d x + δ x)
    unfolding diff-group-add-def
    unfolding diff-group-add-axioms-def
  proof (intro conjI)
    show  ab-group-add op − uminus (0::int) op +
      by intro-locales
  next
    show ∀ x (y::int). d (x + y) + δ (x + y) = d x + δ x + (d y + δ y)
    proof (rule allI)+
      fix a b :: int
      show d (a + b) + δ (a + b) = d a + δ a + (d b + δ b)
        unfolding diff-int-def
        unfolding pert-int-def by arith
    qed
  next
    show (λx::int. d x + δ x) ∘ (λx. d x + δ x) = (λx. 0)
      unfolding diff-int-def
```

unfolding *pert-int-def* **by** (*simp add*: *expand-fun-eq*)
qed
qed

end

We now prove some facts about generic functions. With appropriate restrictions over the type classes over which they are defined, functions can be proved to be also instances of some type classes.

instantiation *fun* :: (*ab-semigroup-add*, *ab-semigroup-add*) *ab-semigroup-add*
begin

definition *plus-fun-def*: $f + g == (\%x.\ f\ x + g\ x)$

instance proof
fix $x\ y\ z :: 'a => 'b$
show $x + y + z = x + (y + z)$
unfolding *plus-fun-def* **by** (*auto simp add*: *add-assoc*)
next
fix $x\ y :: 'a => 'b$
show $x + y = y + x$
unfolding *plus-fun-def* **by** (*auto simp add*: *add-commute*)
qed

end

instantiation *fun* :: (*comm-monoid-add*, *comm-monoid-add*) *comm-monoid-add*
begin

definition *zero-fun-def*: $0 == (\lambda x.\ 0)$

instance proof
fix $a :: 'a => 'b$
show $0 + a = a$
unfolding *zero-fun-def plus-fun-def* **by** *simp*
qed

end

The Isabelle release 2008 already contains the definition of the difference of functions and also the unary minus

instantiation *fun* :: (*ab-group-add*, *ab-group-add*) *ab-group-add*
begin

instance proof
fix $a :: 'a => 'b$
show $-\ a + a = 0$
unfolding *fun-Compl-def*
unfolding *zero-fun-def*

unfolding *plus-fun-def* **by** *simp*
next
fix *a b* :: *'a* => *'b*
show $a - b = a + - b$
unfolding *plus-fun-def*
unfolding *fun-diff-def*
unfolding *fun-Compl-def* **by** (*simp add*: *expand-fun-eq*)
qed

end

The following type class specifies a differential group with a perturbation and also a homotopy operator.

The previous fact about the *fun* datatype contructor allows us now to use *op* − to define α in a more readable way

class *diff-group-add-pert-hom* = *diff-group-add-pert* +
fixes *hom-oper*:: *'a* => *'a* (*h* - [*81*] *80*)
assumes *h-hom-ab*: $h\ (a + b) = h\ a + h\ b$
and *h-nilpot*: $(\lambda x.\ h\ x) \circ (\lambda x.\ h\ x) = (\lambda x.\ 0)$
begin

definition α :: *'a* => *'a*
where $\alpha = (\lambda x.\ - (pert\ (hom\text{-}oper\ x)))$

end

instantiation *int* :: *diff-group-add-pert-hom*
begin

definition *hom-oper-int-def*: *hom-oper* ≡ ($\lambda x.$ *0*::*int*)

instance proof
fix *a b* :: *int*
show $h\ (a + b) = h\ a + h\ b$
unfolding *hom-oper-int-def* **by** *arith*
next
show $(\lambda x.\ h\ x) \circ (\lambda x.\ h\ x) = (\lambda x::int.\ 0)$
unfolding *hom-oper-int-def*
unfolding *expand-fun-eq* **by** *auto*
qed

end

lemma [*code*]: **shows** $\alpha = (- ((\lambda\ x.\ \delta\ x) \circ (\lambda x.\ h\ x)))$
unfolding *α-def*
unfolding *fun-Compl-def* **by** *simp*

17 Local nilpotency

We add now the notion of *local-bounded-func*in a purely *existential* way; from the existential definition we will later define the function providing this local bound for every x.

The reason to introduce now this notion is that α is the function verifying the local nilpotency condition

context *ab-group-add*
begin

definition *local-bounded-func* :: $('a => 'a) => bool$
  where *local-bounded-func* $f = (\forall\ x.\ \exists\ n.\ (f\hat{\ }n)\ x = 0)$

Here is a relevant difference with the previous proof of the BPL; there, the local bound was defined as the Least natural number n satisfying the property $(\alpha\ \hat{}\ n)\ x = (0::'a)$. Now, in our attempt to make this definition computable, or executable, we define it as an iterating structure (a *For* loop), where the boolean condition in the loop is expressed as $\lambda y.\ y \neq (0::'a)$

Later we will try to apply the code generator over these definitions

definition *local-bound-gen* :: $('a => 'a) => 'a => nat => nat$
  where *local-bound-gen* $f\ x\ n == For\ (\lambda\ y.\ y \neq 0)\ f\ (\lambda\ y\ n.\ n + (1::nat))\ x\ n$

definition *local-bound*:: $('a => 'a) => 'a => nat$
  where *local-bound* $f\ x =$ *local-bound-gen* $f\ x\ 0$

end

We now define the simplification rule for *local-bound-gen*:

lemmas *local-bound-gen-simp* =
  For-simp[*of* $(\lambda\ y.\ y \neq (0::'a::ab\text{-}group\text{-}add))$ - $\lambda\ y\ n.\ n+(1::nat)$,
  simplified local-bound-gen-def[*symmetric*]]

Two simple "calculations" with *local-bound*:

lemma *local-bound* $f\ 0 = 0$
  unfolding *local-bound-def*
  unfolding *local-bound-gen-simp* [*of f 0 0*] **by** *simp*

lemma $x \neq 0 \Longrightarrow$ *local-bound* $(\lambda\ x.\ 0)\ x = 1$
  unfolding *local-bound-def*
  using *local-bound-gen-simp* **by** *simp*

Now, we connect the neccesary termination of For with our termination condition, *local-bounded-func*.

Then, under the *local-bounded-func* premise the loop will be terminating.

lemma *local-bounded-func-impl-terminates-loop*:
local-bounded-func f = (∀ *x*. *terminates* (λ *y*. *y* ≠ *0*, *f*, *x*))
unfolding *local-bounded-func-def*
unfolding *terminates-simp*
unfolding *Orbit-def* **by** *simp*

lemma *LEAST-local-bound-0*:
(*LEAST n*::*nat*. (*f* ^ *n*) (*0*::'*a*::*ab-group-add*) = (*0*::'*a*)) = *0*
using *Least-le* [*of* λ*n*. (*f* ^*n*) *0* = *0 0*] **by** *simp*

lemma *local-bound-gen-correct*:
terminates (λ *y*. *y* ≠ (*0*::'*a*::*ab-group-add*), *f*, *x*)
⟹ *local-bound-gen f x m* = *m* + (*LEAST n*::*nat*. (*f*^*n*) *x* = *0*)
apply (*rule For-pinduct*[**where** *i*=*x* **and** *s*=*m* **and** *Ac*=λ *y n*. *n*+*1*])
apply *simp*
apply (*subst local-bound-gen-simp*)
apply (*case-tac i* = *0*)
apply (*simp add*: *LEAST-local-bound-0*)
apply *simp*
apply (*frule-tac x*=*i* **in** *terminates-implies*)
apply (*frule-tac i*=*i* **in** *terminates-imp*)
apply *simp*
apply (*frule-tac x*=*f i* **in** *terminates-implies*)
apply *auto*
apply (*rule Least-Suc2*[*symmetric*])
apply (*auto simp add*: *funpow-zip*)
done

The following lemma exactly represents the difference between our old definitions, with which we proved the BPL, and the new ones, from which we are trying to generate code; under the termination premise, both *LEAST n*. (*f* ^ *n*) *x* = (*0*::'*b*), the old definition of local nilpotency, and *local-bound f x*, the loop computing the lower bound, are equivalent

Whereas *LEAST n*. (*f* ^ *n*) *x* = (*0*::'*b*) does not have a computable interpretation, *local-bound f x* does have it, and code can be generated from it.

lemma *local-bound-correct*:
terminates (λ *y*. *y* ≠ (*0*::'*a*::*ab-group-add*), *f*, *x*)
⟹ *local-bound f x* = (*LEAST n*::*nat*. (*f*^*n*) *x* = *0*)
unfolding *local-bound-def*
unfolding *local-bound-gen-correct* [*of f x 0*] **by** *arith*

lemma *local-bounded-func-impl-local-bound-is-Least*:
assumes *lbf-f*:*local-bounded-func f*
shows *local-bound f x* = (*LEAST n*::*nat*. (*f*^*n*) *x* = *0*)
using *lbf-f*
unfolding *local-bounded-func-impl-terminates-loop* [*OF*]
using *local-bound-correct* [*of f x*] **..**

Both *local-bound* and *terminates* are executable.

This is another possible definition of our iterative process as a tail recursion, instead of using *While*, suggested by Alexander Krauss; code generation is also possible from this definition.

A good motivation to use the *While* operator instead of this one, is that some additional induction principles have been provided for the *For* and *While* operators.

function (*tailrec*) *local-bound′* :: (′*a*::*zero* ⇒ ′*a*) ⇒ ′*a* ⇒ *nat* ⇒ *nat*
  where
  local-bound′ f x n
  = (*if* (*f*ˆ*n*) *x* = *0* *then n else local-bound′ f x* (*Suc n*))
  by *pat-completeness auto*

lemma [*code func*]:
  local-bound′ f (*x*::′*a*::*zero*) *n*
  = (*if* (*f*ˆ*n*) *x* = *0* *then n else local-bound′ f x* (*Suc n*))
  by *simp*

export-code *local-bound′*
  in *SML* **file** *local-bound.ML*

lemma [*code func*]:
  While continue f s
  = (*if continue s then While continue f* (*f s*) *else s*)
  unfolding *While-simp* [*of continue f s*] **..**

export-code *While*
  in *SML* **file** *Loop.ML*

export-code *local-bound*
  in *SML* **file** *local-bound2.ML*

18 Finite sums

The following definition of *fin-sum* will replace the definitions for sums of series used in the formal proof of the BPL.

That definitions were based on the fold operator over sets, from which direct code generation cannot be obtained.

The finite sum of a series is defined as a primitive recursive function over the natural numbers.

This definition will have to be proved later equivalent, in our setting, to the sums appearing in the BPL proof.

primrec *fin-sum* :: ((′*a*::*ab-group-add*) => ′*a*) => *nat* => (′*a* => ′*a*)

where
fin-sum f 0 $=$*id*
| *fin-sum f* (*Suc n*) $=$ *f*^(*Suc n*) $+$ (*fin-sum f n*)

The following definition of *diff-group-add-pert-hom-bound-exist* contains the local nilpotency condition. It is based on an existential statement.

For all x belonging to our differential group, we state the existence of a natural number n which is a bound for α

We then prove that it is equivalent to the previously given definition of *locale-bounded-func*.

Finally, we link this fact to the previous results about computation of the bounds as a *For* operator.

class *diff-group-add-pert-hom-bound-exist* $=$
diff-group-add-pert-hom $+$
assumes *local-nilp-cond*: $\forall\, x.\exists\, n::nat.\ (\alpha\,\hat{}\,n)\ x = 0$

lemma *diff-group-add-pert-hom-bound-exist-impl-local-bounded-func-alpha*:
assumes *diff-group-add-pert-hom-bound-exist*:
diff-group-add-pert-hom-bound-exist
op $-$ $(\lambda x.\ -\ x)$ *0 op* $+$ $(\lambda x.\ d\ x)$ $(\lambda x.\ \delta\ x)$ $(\lambda x.\ h\ x)$
shows *local-bounded-func* (α::'*a*::*diff-group-add-pert-hom-bound-exist* => '*a*)
unfolding *local-bounded-func-def*
using *local-nilp-cond* **.**

lemma *diff-group-add-pert-hom-bound-exist-impl-local-bound-is-Least*:
assumes *diff*:
diff-group-add-pert-hom-bound-exist
op $-$ $(\lambda x.\ -\ x)$ *0 op* $+$ $(\lambda x.\ d\ x)$ $(\lambda x.\ \delta\ x)$ $(\lambda x.\ h\ x)$
shows *local-bound* α (x::'*a*::*diff-group-add-pert-hom-bound-exist*)
$=$ (*LEAST* $n::nat.\ (\alpha\,\hat{}\,n)\ x = 0$)
unfolding *local-bounded-func-impl-local-bound-is-Least*
[*OF diff-group-add-pert-hom-bound-exist-impl-local-bounded-func-alpha*
[*OF diff*]] **..**

Apparently, 'a does not belong to the appropriate type class.

It does not seem either a good option to use long qualifiers with the locale name

Instead, we have to use the following explicit restriction of the type parameter

The following definitions will have to be later compared with the ones Φ, ...

Additionally, code generation from them must be possible.

definition Φ ::
  $('a::\textit{diff-group-add-pert-hom-bound-exist} => 'a)$
  where $\Phi = (\lambda x.\ \textit{fin-sum}\ \alpha\ (\textit{local-bound}\ \alpha\ x)\ x)$

definition β :: $('a::\textit{diff-group-add-pert-hom-bound-exist} => 'a)$
  where $\beta = (-\ ((\lambda x.\ h\ x) \circ (\lambda\ x.\ \delta\ x)))$

definition Ψ :: $('a::\textit{diff-group-add-pert-hom-bound-exist} \Rightarrow 'a)$
  where $\Psi = (\lambda x.\ \textit{fin-sum}\ \beta\ (\textit{local-bound}\ \beta\ x)\ x)$

The following definitions are also to be compared with the ones appearing in the output of the *BPL ?D ?R ?h ?C ?f ?g ?δ ?bound-phi* $\Longrightarrow$ *reduction* (*lemma-2-2-15.D′ ?D ?R ?δ*) (|*carrier* = *carrier ?C*, *mult* = *op* $\otimes_{?C}$, *one* = $\mathbf{1}_{?C}$, *diff-group.diff* = $\lambda x.$ *if* $x \in$ *carrier ?C then diff-group.diff ?C x* $\otimes_{?C}$ (*?f* $\circ$ *?δ* $\circ$ *local-nilpotent-alpha.Ψ ?D ?R ?h ?δ* $\circ$ *?g*) *x else* $\mathbf{1}_{?C}$|) (*?f* $\circ$ *local-nilpotent-alpha.Φ ?D ?R ?h ?δ ?bound-phi*) (*local-nilpotent-alpha.Ψ ?D ?R ?h ?δ* $\circ$ *?g*) (*lemma-2-2-15.h′ ?D ?R ?h ?δ ?bound-phi*)

definition dC' :: $('a::\textit{diff-group-add-pert-hom-bound-exist} => 'b::\textit{diff-group-add})$
  $=> ('b => 'a) => ('b => 'b)$
  where $dC'\ f\ g = \textit{diff} + (f \circ (\lambda x.\ \delta\ x) \circ \Psi \circ g)$

definition f' :: $('a::\textit{diff-group-add-pert-hom-bound-exist} => 'b::\textit{diff-group-add})$
  $=> ('a => 'b)$
  where $f'\ f = f \circ \Phi$

definition g' :: $('b::\textit{diff-group-add} => 'a::\textit{diff-group-add-pert-hom-bound-exist})$
  $=> ('b => 'a)$
  where g'*-def*: $g'\ g == \Psi \circ g$

definition h' :: $('a::\textit{diff-group-add-pert-hom-bound-exist} \Rightarrow 'a)$
  where $h' == (\lambda x.\ h\ x) \circ \Phi$

export-code $\Phi\ \Psi\ f'\ g'\ h'\ dC'$ **in** *SML* **file** *output-reduction.ML*

Some facts about the product of types:

instantiation $*$ :: (*ab-semigroup-add*, *ab-semigroup-add*) *ab-semigroup-add*
begin

definition *mult-plus-def*:
  $x + y \equiv (\textit{let}\ (x1,\ x2) = x;\ (y1,\ y2) = y\ \textit{in}\ (x1 + y1,\ x2 + y2))$

instance proof
  fix $a::'a::\textit{ab-semigroup-add} \times 'b::\textit{ab-semigroup-add}$
  obtain $i\ j$ **where** *a-split*: $a = (i,\ j)$ **by** *force*
  fix $b::'a \times 'b$
  obtain $k\ l$ **where** *b-split*: $b = (k,\ l)$ **by** *force*
  fix $c::'a \times 'b$
  obtain $m\ n$ **where** *c-split*: $c = (m,\ n)$ **by** *force*

show $a + b + c = a + (b + c)$
unfolding *a-split b-split c-split*
unfolding *mult-plus-def*
by (*auto simp add*: *add-assoc*)
show $a + b = b + a$
unfolding *a-split b-split*
unfolding *mult-plus-def*
by (*auto simp add*: *add-commute*)
qed

end

definition *x5*:: (*int* × *int*)
where *x5* = ((*3*::*int*), (*5*::*int*)) + (*5*, *7*)

definition *x6* :: (*int* × *int*) × (*int* × *int*)
where *x6* = (((*3*::*int*), (*5*::*int*)), ((*3*::*int*), (*5*::*int*))) + ((*5*, *7*), (*5*, *7*))

export-code *x5 x6*
in *SML* **file** *x6.ML*

instantiation ∗ :: (*comm-monoid-add*, *comm-monoid-add*) *comm-monoid-add*
begin

definition *mult-zero-def*: *0* ≡ (*0*, *0*)

instance by *default* (*simp add*: *split-paired-all mult-plus-def mult-zero-def*)

end

19 Equivalence of both approaches

19.1 Algebraic structures

In the following section we prove that the results already proved using the definitions provided by the Algebra Isabelle Library (leading to the *reduction* D' (|*carrier* = *carrier C*, *mult* = *op* $\otimes_C$, *one* = $\mathbf{1}_C$, *diff-group.diff* = λx. *if x* ∈ *carrier C then diff-group.diff C x* $\otimes_C$ (*f* ∘ δ ∘ *D-R-C-f-g-h-δ-α-bound-phi*.Ψ ∘ *g*) *x else* $\mathbf{1}_C$|) (*f* ∘ *D-R-C-f-g-h-δ-α-bound-phi*.Φ) (*D-R-C-f-g-h-δ-α-bound-phi*.Ψ ∘ *g*) *D-R-h-C-f-g-δ-α-bound-phi.h*′) also hold for the new definitions, where algebraic structures are implemented by means of type classes.

This does not mean that the proof of the BPL can be developed only by using type classes; the degree of expressivity needed in its proofs should be quite hard to achieve using type classes; specially, the parts where restrictions of domains of functions have to be used.

We only pretend to prove that the new definitions, to some extent simplified (see for instance the new proposed series in relation to the previous implementation of series), are equivalent to the old ones, and thus, also satisfy the BPL.

Functions (or functors) translating type classes into algebraic structures implemented as recods are used; these translations are the natural ones

definition *monoid-functor* :: ('*a* => '*a* => '*a*) => ('*a*) => '*a monoid*
  where *monoid-functor A B* == (| *carrier* = *UNIV*, *mult* = *A*, *one* = *B*|)

lemma *monoid-add-impl-monoid*:
  assumes *mon-add*: *monoid-add zero' plus'*
  shows *monoid* (| *carrier* = *UNIV*, *mult* = *plus'*, *one* = *zero'*|)
  using *mon-add*
  unfolding *monoid-def*
  unfolding *monoid-add-def monoid-add-axioms-def*
  unfolding *ab-semigroup-add-def*
  unfolding *ab-semigroup-add-axioms-def*
  unfolding *semigroup-add-def* **by** *simp*

lemma *monoid-functor-preserv*:
  assumes *monoid-add*:*monoid-add zero' plus'*
  shows *monoid* (*monoid-functor plus' zero'*)
  using *monoid-add-impl-monoid* [*OF monoid-add*]
  unfolding *monoid-functor-def* [*of plus' zero'*] **.**

lemma *comm-monoid-add-impl-monoid-add*:
  assumes *comm-monoid-add*: *comm-monoid-add zero' plus'*
  shows *monoid-add zero' plus'*
  using *comm-monoid-add*
  unfolding *comm-monoid-add-def*
  unfolding *monoid-add-def*
  unfolding *ab-semigroup-add-def*
  unfolding *comm-monoid-add-axioms-def*
  unfolding *monoid-add-axioms-def*
  unfolding *ab-semigroup-add-axioms-def* **by** *auto*

lemma *comm-monoid-add-impl-monoid*:
  assumes *c-m*: *comm-monoid-add zero' plus'*
  shows *monoid* (| *carrier* = *UNIV*, *mult* = *plus'*, *one* = *zero'*|)
  using *monoid-add-impl-monoid* [*OF comm-monoid-add-impl-monoid-add*
    [*OF c-m*]] **.**

lemma *ab-group-add-impl-comm-monoid-add*:
  assumes *ab-gr-add*: *ab-group-add uminus' minus' zero' plus'*
  shows *comm-monoid-add zero' plus'*
  using *ab-gr-add*
  unfolding *ab-group-add-def* **..**

lemma *ab-group-class-impl-group*:

assumes *ab-gr-class*: *ab-group-add uminus′ minus′ zero′ plus′*
shows *group* (| *carrier* = *UNIV*, *mult* = *plus′*, *one* = *zero′*|)
proof (*intro-locales*)
show *monoid* (|*carrier* = *UNIV*, *mult* = *plus′*, *one* = *zero′*|)
using *monoid-add-impl-monoid*
[*OF comm-monoid-add-impl-monoid-add*
[*OF ab-group-add-impl-comm-monoid-add*
[*OF ab-gr-class*]]] **.**
show *group-axioms* (|*carrier* = *UNIV*, *mult* = *plus′*, *one* = *zero′*|)
using *ab-gr-class*
unfolding *group-axioms-def*
unfolding *Units-def*
unfolding *ab-group-add-def*
unfolding *comm-monoid-add-def*
unfolding *ab-group-add-axioms-def*
unfolding *ab-semigroup-add-def*
ab-semigroup-add-axioms-def **by** *auto*+
qed

lemma *monoid-functor-preserv-group*:
assumes *ab-gr*: *ab-group-add uminus′ minus′ zero′ plus′*
shows *group* (*monoid-functor plus′ zero′*)
using *ab-group-class-impl-group* [*OF ab-gr*]
unfolding *monoid-functor-def* [*of plus′ zero′*] **.**

lemma *ab-group-add-impl-comm-group*:
assumes *ab-gr-add*: *ab-group-add uminus′ minus′ zero′ plus′*
shows *comm-group* (| *carrier* = *UNIV*, *mult* = *plus′*, *one* = *zero′*|)
proof (*intro-locales*)
show *monoid* (|*carrier* = *UNIV*, *mult* = *plus′*, *one* = *zero′*|)
using *comm-monoid-add-impl-monoid*
[*OF ab-group-add-impl-comm-monoid-add*
[*OF ab-gr-add*]] **.**
show *comm-monoid-axioms* (|*carrier* = *UNIV*, *mult* = *plus′*, *one* = *zero′*|)
using *ab-gr-add*
unfolding *comm-monoid-axioms-def*
unfolding *ab-group-add-def*
unfolding *comm-monoid-add-def*
unfolding *ab-group-add-axioms-def*
unfolding *ab-semigroup-add-def*
unfolding *ab-semigroup-add-axioms-def* **by** *auto*
show *group-axioms* (|*carrier* = *UNIV*, *mult* = *plus′*, *one* = *zero′*|)
using *ab-gr-add*
unfolding *group-axioms-def*
unfolding *Units-def*
unfolding *ab-group-add-def*
unfolding *comm-monoid-add-def*
unfolding *ab-group-add-axioms-def*
unfolding *ab-semigroup-add-def*

unfolding *ab-semigroup-add-axioms-def* **by** *auto+*
qed

lemma *monoid-functor-preserv-ab-group*:
assumes *ab-gr-add*: *ab-group-add uminus' minus' zero' plus'*
shows *comm-group* (*monoid-functor plus' zero'*)
using *ab-group-add-impl-comm-group* [*OF ab-gr-add*]
unfolding *monoid-functor-def* [*of plus' zero'*] **.**

lemma *diff-group-add-impl-comm-group*:
assumes *diff-gr-add*: *diff-group-add uminus' minus' zero' plus' diff'*
shows *comm-group* (|*carrier* = *UNIV*, *mult* = *plus'*, *one* = *zero'*|)
proof (*rule ab-group-add-impl-comm-group*)
show *ab-group-add uminus' minus' zero' plus'*
using *diff-gr-add*
unfolding *diff-group-add-def* **by** *simp*
qed

lemma *diff-group-add-impl-diff-group*:
assumes *diff-gr-add*: *diff-group-add uminus' minus' zero' prod' diff'*
shows *diff-group* (|*carrier* = *UNIV*, *mult* = *prod'*, *one* = *zero'*, *diff-group.diff* = *diff'* |)
proof (*intro-locales*)
from *diff-group-add-impl-comm-group* [*OF diff-gr-add*]
have *comm-gr*: *comm-group* (|*carrier* = *UNIV*, *mult* = *prod'*, *one* = *zero'*|)
by *simp*
show *monoid* (|*carrier* = *UNIV*, *mult* = *prod'*, *one* = *zero'*, *diff-group.diff* = *diff'*|)
using *comm-gr*
unfolding *comm-group-def*
unfolding *comm-monoid-def*
unfolding *monoid-def* **by** *simp*
show *comm-monoid-axioms* (|*carrier* = *UNIV*, *mult* = *prod'*, *one* = *zero'*, *diff-group.diff* = *diff'*|)
using *comm-gr*
unfolding *comm-group-def*
unfolding *comm-monoid-def*
unfolding *comm-monoid-axioms-def* **by** *simp*
show *group-axioms* (|*carrier* = *UNIV*, *mult* = *prod'*, *one* = *zero'*, *diff-group.diff* = *diff'*|)
using *comm-gr*
unfolding *comm-group-def*
unfolding *group-def*
unfolding *group-axioms-def*
unfolding *Units-def* **by** *simp*
show *diff-group-axioms* (|*carrier* = *UNIV*, *mult* = *prod'*, *one* = *zero'*, *diff-group.diff* = *diff'*|)
using *diff-gr-add*
unfolding *diff-group-add-def*

unfolding *diff-group-add-axioms-def*
unfolding *diff-group-axioms-def*
unfolding *hom-completion-def*
unfolding *hom-def*
unfolding *completion-fun2-def*
unfolding *completion-def* **by** *auto*
qed

definition *diff-group-functor* ::
('a => 'a) => ('a => 'a => 'a) => ('a)
=> ('a => 'a => 'a) => ('a => 'a) => 'a *diff-group*
where *diff-group-functor uminus' minus' zero' prod' diff'* =
(| *carrier* = *UNIV*, *mult* = *prod'*, *one* = *zero'*, *diff-group.diff* = *diff'*|)

lemma *diff-group-functor-preserves*:
assumes *diff-gr-add*: *diff-group-add minus' uminus' zero' prod' diff'*
shows *diff-group* (*diff-group-functor uminus' minus' zero' prod' diff'*)
using *diff-group-add-impl-diff-group* [*OF diff-gr-add*]
unfolding *diff-group-functor-def* [*of uminus' minus' zero' prod' diff'*] **.**

After the previous equivalences between algebraic structures, now we prove the equivalence between the old definitions about homomorphisms and the new ones:

19.2 Homomorphisms and endomorphisms.

definition *homo-ab* :: ('*a*::*comm-monoid-add* => '*b*::*comm-monoid-add*) => *bool*

where *homo-ab f* = (*ALL a b. f* (*a* + *b*) = *f a* + *f b*)

lemma *homo-ab-apply*:
assumes *h-f*: *homo-ab f*
shows *f* (*a* + *b*) = *f a* + *f b*
using *homo-ab-def* [*of f*] *h-f* **by** *simp*

lemma *homo-ab-preserves-hom-completion*:
assumes *homo-ab-f*: *homo-ab f*
shows *f* ∈ *hom-completion* (*monoid-functor* (*op* +) *0*) (*monoid-functor* (*op* +) *0*)
using *homo-ab-f*
unfolding *hom-completion-def*
unfolding *homo-ab-def*
unfolding *monoid-functor-def*
unfolding *completion-fun2-def*
unfolding *completion-def*
unfolding *hom-def* **by** *auto*

lemma *plus-fun-apply*:
(*f* + *g*) (*x*::'*a*::*ab-semigroup-add*) = *f x* + *g x*

using *plus-fun-def* [*of f g*] **by** *simp*

lemma *homo-ab-plus-closed*:
assumes *comm-monoid-add-A*: *comm-monoid-add* (*0*::*'a*::*comm-monoid-add*) *op* +
and *comm-monoid-add-B*: *comm-monoid-add* (*0*::*'b*::*comm-monoid-add*) *op* +
and *x*: *homo-ab* (*x*::*'a*::*comm-monoid-add* => *'b*::*comm-monoid-add*)
and *y*: *homo-ab y*
shows *homo-ab* (*x* + *y*)
proof (*unfold homo-ab-def*, *rule allI*, *rule allI*)
have *ab-semigroup-add-plus*: *ab-semigroup-add* (*op* +::*'b* ⇒ *'b* ⇒ *'b*)
using *comm-monoid-add-B*
unfolding *comm-monoid-add-def* **..**
fix *a b* :: *'a*
show (*x* + *y*) (*a* + *b*) = (*x* + *y*) *a* + (*x* + *y*) *b*
proof −
have (*x* + *y*) (*a* + *b*) = *x* (*a* + *b*) + *y* (*a* + *b*)
using *plus-fun-apply* [*of x y a* + *b*] **.**
also have ... = *x a* + *x b* + (*y a* + *y b*)
unfolding *homo-ab-apply* [*OF x*]
unfolding *homo-ab-apply* [*OF y*] **..**
also have ... = *x a* + (*x b* + *y a* + *y b*)
by (*auto simp add*: *add-assoc*)
also have ... = *x a* + (*y a* + *x b* + *y b*)
by (*auto simp add*: *add-commute add-assoc*)
also have ... = *x a* + *y a* + (*x b* + *y b*)
by (*auto simp add*: *add-assoc*)
also have ... = (*x* + *y*) *a* + (*x* + *y*) *b*
unfolding *sym* [*OF plus-fun-apply* [*of x y a*]]
unfolding *sym* [*OF plus-fun-apply* [*of x y b*]] **..**
finally show *?thesis* **.**
qed
qed

lemma *end-comm-monoid-add-closed*:
assumes *comm-monoid-add*: *comm-monoid-add* (*0*::*'a*::*comm-monoid-add*) *op* +
and *x*: *homo-ab* (*x*::*'a*::*comm-monoid-add* => *'a*)
and *y*: *homo-ab y*
shows *homo-ab* (*x* + *y*)
using *homo-ab-plus-closed* [*OF comm-monoid-add comm-monoid-add x y*] **.**

lemma *comm-monoid-add-impl-homo-abelian-monoid*:
assumes *comm-monoid-add*: *comm-monoid-add* (*0*::*'a*::*comm-monoid-add*) *op* +
shows *abelian-monoid* (|*carrier* = {*f*::*'a*::*comm-monoid-add* => *'a*. *homo-ab f*},

mult = *op* ∘,
one = *id*,
zero = *0*,

add = *op* +|)
proof (*intro abelian-monoidI*, *auto*)
fix *x y* :: *'a* => *'a*
assume *x*: *homo-ab x* **and** *y*: *homo-ab y*
show *homo-ab* (*x* + *y*)
using *end-comm-monoid-add-closed* [*OF comm-monoid-add x y*] **.**
next
show *homo-ab* (*0*::*'a* => *'a*)
unfolding *zero-fun-def homo-ab-def* **by** *simp*
next
fix *x y z* :: *'a* => *'a*
assume *x*: *homo-ab x* **and** *y*: *homo-ab y* **and** *z*: *homo-ab z*
show *x* + *y* + *z* = *x* + (*y* + *z*)
unfolding *plus-fun-def* **by** (*simp only*: *add-assoc*)
next
fix *x y* :: *'a* => *'a*
assume *x*: *homo-ab x* **and** *y*: *homo-ab y*
show *x* + *y* = *y* + *x* **unfolding** *plus-fun-def expand-fun-eq* **by** (*simp add*: *add-ac*)
qed

lemma *ab-group-add-impl-uminus-fun-closed*:
assumes *ab-group-add*: *ab-group-add op* − (λ*x*. − *x*) (*0*::*'a*::*ab-group-add*) *op* +

and *f*: *homo-ab* (*f*::*'a*::*ab-group-add* => *'a*)
shows *homo-ab* (− *f*)
proof (*unfold fun-Compl-def homo-ab-def*, *rule allI*, *rule allI*)
fix *a b* :: *'a*
show − *f* (*a* + *b*) = − *f a* + − *f b*
proof −
have *l-h-s*: *f* (*a* + *b*) + − *f* (*a* + *b*) = *0*
using *sym* [*OF add-commute* [*of* − *f* (*a* + *b*) *f* (*a* + *b*)]]
unfolding *ab-left-minus* [*of f* (*a* + *b*)] **.**
have *f* (*a* + *b*) + (− *f a* + − *f b*) = − *f a* + − *f b* + *f* (*a* + *b*)
using *sym* [*OF add-commute* [*of* − *f a* + − *f b f* (*a* + *b*)]] **.**
also have − *f a* + − *f b* + *f* (*a* + *b*) = − *f a* + − *f b* + (*f a* + *f b*)
unfolding *homo-ab-apply* [*OF f*] **..**
also have . . . = ((− *f a* + − *f b*) + *f a*) + *f b*
unfolding *sym* [*OF add-assoc* [*of* − *f a* + − *f b f a f b*]] **..**
also have . . . = (− *f a* + (− *f b* + *f a*)) + *f b*
unfolding *add-assoc* [*of* − *f a* − *f b f a*] **..**
also have . . . = (− *f a* + (*f a* + − *f b*)) + *f b*
unfolding *add-commute* [*of* − *f b f a*] **..**
also have . . . = ((− *f a* + *f a*) + − *f b*) + *f b*
unfolding *sym* [*OF add-assoc* [*of* − *f a f a* − *f b*]] **..**
also have . . . = *0* + − *f b* + *f b*
unfolding *ab-left-minus* [*of f a*] **..**
also have . . . = *0* + (− *f b* + *f b*)
unfolding *add-assoc* [*of 0* − *f b f b*] **..**

```
    also have ... = 0 + 0 unfolding ab-left-minus [of f b] ..
    also have ... = 0 by simp
    finally have r-h-s: f (a + b) + (− f a + − f b) = 0 .
    with l-h-s have f (a + b) + − f (a + b) = f (a + b) + (− f a + − f b) by
simp
    then have − (f::'a => 'a) ((a::'a) + (b::'a)) + (f (a + b) + − f (a + b))
      = − f (a + b) + (f (a + b) + (− f a + − f b)) by simp
    with sym [OF add-assoc [of − f (a + b) f (a + b) − f (a + b)]]
      sym [OF add-assoc [of − f (a + b) f (a + b) − f a + − f b]]
    have (− (f::'a => 'a) ((a::'a) + (b::'a)) + f (a + b)) + − f (a + b)
      = (− f (a + b) + f (a + b)) + (− f a + − f b) by simp
    with ab-left-minus [of f (a + b)]
    have 0 + − f (a + b) = 0 + (− f a + − f b) by simp
    with left-minus [of − f (a + b)] left-minus [of (− f a + − f b)]
    show ?thesis by simp
  qed
qed

lemma ab-group-add-impl-homo-abelian-group-axioms:
  assumes ab-group-add: ab-group-add op − (λx. − x) (0::'a::ab-group-add) op +
  shows abelian-group-axioms (|carrier = {f::'a::ab-group-add => 'a. homo-ab f},

   mult = op ∘,
   one = id,
   zero = 0,
   add = op +|)
proof (unfold abelian-group-axioms-def, simp)
  show comm-group (|carrier = Collect homo-ab, mult = op +, one = (0::'a =>
'a)|)
  proof (intro-locales)
    show monoid (|carrier = Collect homo-ab, mult = op +, one = (0::'a => 'a)|)
    proof (intro monoidI, simp-all)
      fix x y :: 'a => 'a
      assume x: homo-ab x and y: homo-ab y
      show homo-ab (x + y)
        using ab-group-add end-comm-monoid-add-closed [OF - x y]
        unfolding ab-group-add-def by simp
    next
      show homo-ab (0::'a => 'a)
        unfolding zero-fun-def homo-ab-def by simp
    qed
  next
    show comm-monoid-axioms (|carrier = Collect homo-ab,
      monoid.mult = op +,
      one = (0::'a => 'a)|)
      unfolding comm-monoid-axioms-def by auto
  next
    show group-axioms (|carrier = Collect homo-ab,
      monoid.mult = op +,
```

one = (*0*::′*a* => ′*a*)⦈
    proof (*unfold group-axioms-def Units-def*, *auto*)
      fix *x* :: ′*a* => ′*a*
      assume *x*: *homo-ab x*
      show ∃ *y*::′*a* ⇒ ′*a*. *homo-ab y* ∧ *y* + *x* = *0* ∧ *x* + *y* = *0*
      proof (*rule exI* [*of* - − *x*], *intro conjI*)
        show *homo-ab* (− *x*)
          using *ab-group-add-impl-uminus-fun-closed* [*OF ab-group-add x*] **.**
        show (− *x*) + *x* = *0* **by** *simp*
        show *x* + (− *x*) = *0* **by** *simp*
      qed
    qed
  qed
qed

The previous lemma, *ab-group-add op* − *uminus* (*0*::?′*a*) *op* + ⟹ *abelian-group-axioms* (|*carrier* = {*f*. *homo-ab f*}, *mult* = *op* ∘, *one* = *id*, *zero* = *0*, *add* = *op* +⦈, proves the elements of *homo-ab* to be an abelian monoid under suitable operations.

In order to show that composition gives place to a monoid, the underlying structure needs not to be even a monoid

lemma *homo-monoid*:
  shows *monoid* (|*carrier* = {*f*. *homo-ab f*},
  monoid.mult = *op* ∘,
  one = *id*,
  zero = *0*,
  add = *op* +⦈
  (**is** *monoid ?HOMO-AB*)
proof (*intro monoidI*, *auto*)
  fix *x y* :: ′*a* => ′*a*
  assume *x*: *homo-ab x* **and** *y*: *homo-ab y*
  then show *homo-ab* (*x* ∘ *y*) **unfolding** *homo-ab-def* **by** *simp*
next
  show *homo-ab id* **unfolding** *homo-ab-def* **by** *simp*
next
  fix *x y z* :: ′*a* => ′*a*
  assume *x*: *homo-ab x* **and** *y*: *homo-ab y* **and** *z*: *homo-ab z*
  show *x* ∘ *y* ∘ *z* = *x* ∘ (*y* ∘ *z*) **by** (*simp add*: *expand-fun-eq*)
qed

A couple of lemmas completing the proof of the set *homo-ab* being a ring, with suitable operations

lemma *ab-group-add-impl-homo-ring-axioms*:
  assumes *ab-group-add*: *ab-group-add op* − (λ*x*. − *x*) (*0*::′*a*::*ab-group-add*) *op* +
  shows *ring-axioms* (|*carrier* = {*f*. *homo-ab f*},
  mult = *op* ∘,
  one = *id*,
  zero = (*0*::′*a*::*ab-group-add* => ′*a*),

add = *op* +⦈
proof (*intro ring-axioms.intro*, *auto*)
fix *x y z* :: *'a* => *'a*
assume *x*: *homo-ab x* **and** *y*: *homo-ab y* **and** *z*: *homo-ab z*
show (*x*::*'a* => *'a*) + *y* ∘ *z* = (*x* ∘ *z*) + (*y* ∘ *z*)
unfolding *plus-fun-def*
unfolding *expand-fun-eq* **by** *auto*
show (*z*::*'a* => *'a*) ∘ *x* + *y* = (*z* ∘ *x*) + (*z* ∘ *y*)
using *z*
unfolding *expand-fun-eq plus-fun-def homo-ab-def* **by** *auto*
qed

lemma *ab-group-add-impl-homo-ring*:
assumes *ab-group-add*: *ab-group-add op* − (λ*x*. − *x*) (*0*::*'a*::*ab-group-add*) *op* +
shows *ring* ⦇*carrier* = {*f*::*'a*::*ab-group-add* => *'a*. *homo-ab f*},
mult = *op* ∘,
one = *id*,
zero = (*0*::*'a* => *'a*),
add = *op* +⦈
using *ab-group-add*
using *comm-monoid-add-impl-homo-abelian-monoid*
using *ab-group-add-impl-homo-abelian-group-axioms*
using *homo-monoid*
using *ab-group-add-impl-homo-ring-axioms* **by** (*unfold ab-group-add-def*, *intro-locales*, *auto*)

The following definition includes the notion of differential homomorphism, a homomorphism that additionally commutes with the corresponding differentials.

definition *homo-diff* :: (*'a*::*diff-group-add* => *'b*::*diff-group-add*) => *bool*
where *homo-diff f* = ((*ALL a b*. *f* (*a* + *b*) = *f a* + *f b*) ∧ *f* ∘ *diff* = *diff* ∘ *f*)

lemma *homo-diff-preserves-hom-diff*:
assumes *homo-diff-f*: *homo-diff f*
shows *f* ∈ *hom-diff* (*diff-group-functor* (λ*x*. − *x*) *op* − *0* (*op* +) *diff*)
(*diff-group-functor* (λ*x*. − *x*) *op* − *0* (*op* +) *diff*)
using *homo-diff-f*
unfolding *hom-diff-def*
unfolding *hom-completion-def*
unfolding *homo-diff-def*
unfolding *diff-group-functor-def*
unfolding *completion-fun2-def*
unfolding *completion-def*
unfolding *hom-def*
by *auto*

19.3 Definition of constants.

The following definition of reduction is to be understand as follows: a pair of homomorphisms (f, g) will be a reduction iff: the underlying algebraic structures (given as classes) are, respectively, a differential group class with a perturbation and a homology operator (i.e, class *diff-group-add-pert-hom-bound-exists*) satisfying the local nilpotency condition, and a differential group class (*diff-group-add*), and the homomorphisms *f*, *g* and *h* satisfy the properties required by the usual reduction definition.

In the definition it can be noted the convenience of using overloading symbols provided by the class mechanism.

definition
reduction-class-ext ::
(*'a*::*diff-group-add-pert-hom-bound-exist* => *'b*::*diff-group-add*)
=> (*'b* => *'a*) => *bool*
where *reduction-class-ext f g* =
((*diff-group-add-pert-hom-bound-exist op* − (λx::*'a*. − *x*) *0* (*op* +) *diff pert hom-oper*)
∧ (*diff-group-add op* − (λx::*'b*. − *x*) *0* (*op* +) *diff*)
∧ (*homo-diff f*) ∧ (*homo-diff g*)
∧ (*f* ∘ *g* = *id*)
∧ ((*g* ∘ *f*) + (*diff* ∘ *hom-oper*) + (*hom-oper* ∘ *diff*) = *id*)
∧ (*f* ∘ *hom-oper* = (*0*::*'a* => *'b*))
∧ (*hom-oper* ∘ *g* = (*0*::*'b* => *'a*)))

The previous definition contains all the ingredients required to apply the old proof of the BPL.

lemma *reduction-class-ext-impl-diff-group-add-pert-hom-bound-exist*:
assumes *r-c-e*: *reduction-class-ext* (*f*::*'a*::*diff-group-add-pert-hom-bound-exist* => *'b*::*diff-group-add*) *g*
shows (*diff-group-add-pert-hom-bound-exist op* −
(λx::*'a*::*diff-group-add-pert-hom-bound-exist*. − *x*) *0* (*op* +) *diff pert hom-oper*)
using *r-c-e* **unfolding** *reduction-class-ext-def* **..**

lemma *reduction-class-ext-impl-diff-group-add*:
assumes *r-c-e*: *reduction-class-ext* (*f*::*'a*::*diff-group-add-pert-hom-bound-exist* => *'b*::*diff-group-add*) *g*
shows (*diff-group-add op* − (λx::*'b*::*diff-group-add*. − *x*) *0* (*op* +) *diff*)
using *r-c-e* **unfolding** *reduction-class-ext-def* **by** *fast*

lemma *reduction-class-ext-impl-homo-diff-f*:
assumes *r-c-e*: *reduction-class-ext* (*f*::*'a*::*diff-group-add-pert-hom-bound-exist* => *'b*::*diff-group-add*) *g*
shows *homo-diff f*
using *r-c-e* **unfolding** *reduction-class-ext-def* **by** *fast*

lemma *reduction-class-ext-impl-homo-diff-g*:

assumes *r-c-e*: *reduction-class-ext* (*f*::'*a*::*diff-group-add-pert-hom-bound-exist* => '*b*::*diff-group-add*) *g*
shows *homo-diff g*
using *r-c-e* **unfolding** *reduction-class-ext-def* **by** *fast*

lemma *reduction-class-ext-impl-fg-id*:
assumes *r-c-e*: *reduction-class-ext* (*f*::'*a*::*diff-group-add-pert-hom-bound-exist* => '*b*::*diff-group-add*) *g*
shows $f \circ g = id$
using *r-c-e* **unfolding** *reduction-class-ext-def* **by** *fast*

lemma *reduction-class-ext-impl-gf-dh-hd-id*:
assumes *r-c-e*: *reduction-class-ext* (*f*::'*a*::*diff-group-add-pert-hom-bound-exist* => '*b*::*diff-group-add*) *g*
shows $(g \circ f) + (diff \circ hom\text{-}oper) + (hom\text{-}oper \circ diff) = id$
using *r-c-e* **unfolding** *reduction-class-ext-def* **by** *fast*

lemma *reduction-class-ext-impl-fh-0*:
assumes *r-c-e*: *reduction-class-ext* (*f*::'*a*::*diff-group-add-pert-hom-bound-exist* => '*b*::*diff-group-add*) *g*
shows $f \circ hom\text{-}oper = 0$
using *r-c-e* **unfolding** *reduction-class-ext-def* **by** *fast*

lemma *reduction-class-ext-impl-hg-0*:
assumes *r-c-e*: *reduction-class-ext* (*f*::'*a*::*diff-group-add-pert-hom-bound-exist* => '*b*::*diff-group-add*) *g*
shows $hom\text{-}oper \circ g = 0$
using *r-c-e* **unfolding** *reduction-class-ext-def* **by** *fast*

The following lemma will be useful later, when we verify the premises of the BPL

lemma *hdh-eq-h*:
assumes *r-c-e*: *reduction-class-ext f* (*g*::'*b*::*diff-group-add* => '*a*::*diff-group-add-pert-hom-bound-exist*)
shows (*hom-oper*::'*a*::*diff-group-add-pert-hom-bound-exist* => '*a*) ∘ *diff* ∘ *hom-oper* = *hom-oper*
proof −
have *hom-oper* = *hom-oper* ∘ *id* **by** *simp*
also have ... = $hom\text{-}oper \circ ((g \circ f) + (diff \circ hom\text{-}oper) + (hom\text{-}oper \circ diff))$
using *r-c-e*
unfolding *reduction-class-ext-def* **by** *simp*
also have ... = $hom\text{-}oper \circ (diff \circ hom\text{-}oper)$
using *reduction-class-ext-impl-diff-group-add-pert-hom-bound-exist* [*OF r-c-e*]
using *reduction-class-ext-impl-hg-0* [*OF r-c-e*]
unfolding *diff-group-add-pert-hom-bound-exist-def*
unfolding *diff-group-add-pert-hom-def*
unfolding *diff-group-add-pert-hom-axioms-def*
unfolding *expand-fun-eq plus-fun-def zero-fun-def* **by** *auto*

finally show *?thesis* **by** (*simp add*: *o-assoc*)
qed

lemma *diff-group-add-pert-hom-bound-exist-impl-diff-group-add*:
assumes *d-g*: (*diff-group-add-pert-hom-bound-exist op* −
(λx::′*a*::*diff-group-add-pert-hom-bound-exist*. − *x*) *0* (*op* +) *diff pert hom-oper*)
shows *diff-group-add op* −
(λx::′*a*::*diff-group-add-pert-hom-bound-exist*. − *x*) *0* (*op* +) *diff*
using *d-g*
unfolding *diff-group-add-pert-hom-bound-exist-def*
unfolding *diff-group-add-pert-hom-def*
unfolding *diff-group-add-pert-def* **by** *fast*

The new definition of *reduction-class-ext* preserves the previous definition of reduction in the old approach, *reduction*

lemma *reduction-class-ext-preserves-reduction*:
assumes *r-c-e*: *reduction-class-ext f g*
shows *reduction* (*diff-group-functor* (λx::′*a*::*diff-group-add-pert-hom-bound-exist*.
− *x*)
(*op* −) *0* (*op* +) *diff*)
(*diff-group-functor* (λx::′*b*::*diff-group-add*. − *x*) (*op* −) *0* (*op* +) *diff*)
f g hom-oper
(**is** *reduction ?D ?C f g hom-oper*)
proof (*unfold reduction-def reduction-axioms-def*, *auto*)
have *dga*: *diff-group-add-pert-hom-bound-exist op* − *uminus* (*0*::′*a*) *op* + *diff pert hom-oper*
using *reduction-class-ext-impl-diff-group-add-pert-hom-bound-exist* [*OF r-c-e*] **.**
show *diff-group ?D*
using *diff-group-functor-preserves* [*of op* − *uminus* (*0*::′*a*) *op* + *diff*]
diff-group-add-pert-hom-bound-exist-impl-diff-group-add [*OF dga*] **by** *simp*
next
show *diff-group ?C*
using *r-c-e diff-group-functor-preserves* [*of op* − *uminus* (*0*::′*b*) *op* + *diff*]
unfolding *reduction-class-ext-def* [*of f g*] **by** *simp*
next
show *f* ∈ *hom-diff ?D ?C*
using *r-c-e homo-diff-preserves-hom-diff* [*of f*]
unfolding *reduction-class-ext-def* [*of f g*] **by** *simp*
next
show *g* ∈ *hom-diff ?C ?D*
using *r-c-e homo-diff-preserves-hom-diff* [*of g*]
unfolding *reduction-class-ext-def* [*of f g*] **by** *simp*
next
show *hom-oper* ∈ *hom-completion ?D ?D*
using *diff-group-add-pert-hom.h-hom-ab*
[*of op* − *uminus* (*0*::′*a*) *op* + *diff pert hom-oper*]
using *homo-ab-preserves-hom-completion* [*of* (λx::′*a*. *h x*)]
using *r-c-e*
unfolding *hom-completion-def*

unfolding *hom-def*
unfolding *completion-fun2-def*
unfolding *completion-def*
unfolding *reduction-class-ext-def*
unfolding *diff-group-functor-def*
unfolding *diff-group-add-pert-hom-bound-exist-def* **by** *auto*
next
show $(\lambda x.\ h\ x) \circ (\lambda x.\ h\ x) = (\lambda x{::}'a.\ monoid.one\ ?D)$
using *diff-group-add-pert-hom.h-nilpot*
$[of\ op\ -\ uminus\ (0{::}'a)\ op\ +\ diff\ pert\ hom\text{-}oper]$
using *r-c-e*
unfolding *reduction-class-ext-def*
unfolding *diff-group-add-pert-hom-bound-exist-def*
unfolding *zero-fun-def*
unfolding *reduction-class-ext-def*
unfolding *diff-group-functor-def* **by** *simp*
next
show $f \circ g = (\lambda x{::}'b.\ if\ x \in carrier\ ?C\ then\ id\ x\ else\ monoid.one\ ?C)$
using *r-c-e*
unfolding *reduction-class-ext-def*
unfolding *diff-group-functor-def expand-fun-eq* **by** *simp*
next
show $f \circ (\lambda x.\ h\ x) = (\lambda x{::}'a.\ monoid.one\ ?C)$
using *r-c-e*
unfolding *reduction-class-ext-def diff-group-functor-def zero-fun-def*
by *simp*
next
show $(\lambda x.\ h\ x) \circ g = (\lambda x{::}'b.\ monoid.one\ ?D)$
using *r-c-e*
unfolding *reduction-class-ext-def diff-group-functor-def zero-fun-def*
by *simp*
next
show $(\lambda x{::}'a.\ if\ x \in carrier\ ?D\ then\ monoid.mult\ ?D\ ((g \circ f)\ x)$
$(if\ x \in carrier\ ?D\ then\ monoid.mult\ ?D\ ((diff\text{-}group.diff\ ?D \circ hom\text{-}oper)\ x)$
$((hom\text{-}oper \circ diff\text{-}group.diff\ ?D)\ x)\ else\ monoid.one\ ?D)$
$else\ monoid.one\ ?D) =$
$(\lambda x{::}'a.\ if\ x \in carrier\ ?D\ then\ id\ x\ else\ monoid.one\ ?D)$
using *r-c-e*
unfolding *reduction-class-ext-def*
unfolding *diff-group-functor-def*
unfolding *plus-fun-def*
by (*auto simp add*: *expand-fun-eq add-assoc*)
qed

The new definition of perturbation, included in the definition of *diff-group-add-pert*, also preserves the old definition of perturbation, *analytic-part-local.pert*

lemma *diff-group-add-pert-hom-bound-exist-preserves-pert*:
assumes *diff-group-add-pert-hom-bound-exist*:
diff-group-add-pert-hom-bound-exist (*op* −)

(*λx::'a::diff-group-add-pert-hom-bound-exist. − x*) *0* (*op* +) *diff pert hom-oper*
shows *pert* ∈ *analytic-part-local.pert*
(*diff-group-functor* (*λx::'a::diff-group-add-pert-hom-bound-exist. − x*) (*op* −) *0* (*op* +) *diff*)
(**is** *pert* ∈ *analytic-part-local.pert ?D*)
proof (*unfold pert-def*, *auto*)
show *pert* ∈ *hom-completion ?D ?D*
using *diff-group-add-pert-hom-bound-exist*
unfolding *diff-group-add-pert-hom-bound-exist-def*
unfolding *diff-group-add-pert-hom-def*
unfolding *diff-group-add-pert-def*
unfolding *diff-group-add-pert-axioms-def*
unfolding *diff-group-functor-def*
unfolding *monoid-functor-def*
unfolding *hom-completion-def*
unfolding *completion-fun2-def completion-def hom-def* **by** *simp*
next
have *diff-group-add*: *diff-group-add* (*op* −) *uminus* (*0::'a*) *op* + *diff*
using *diff-group-add-pert-hom-bound-exist-impl-diff-group-add*
[*OF diff-group-add-pert-hom-bound-exist*] **.**
show *diff-group* (|*carrier* = *carrier ?D*, *mult* = *mult ?D*, *one* = *one ?D*,
diff-group.diff = *λx::'a. if x* ∈ *carrier ?D*
then mult ?D (*diff-group.diff ?D x*) (*δ x*) *else one ?D*|)
using *diff-group-add-pert-hom-bound-exist*
using *diff-group-functor-preserves* [*of op* − *uminus* (*0::'a*) *op* + *diff* + *pert*]
unfolding *plus-fun-def*
unfolding *diff-group-add-pert-hom-bound-exist-def*
unfolding *diff-group-add-pert-hom-def*
unfolding *diff-group-add-pert-def*
unfolding *diff-group-add-pert-axioms-def diff-group-functor-def* **by** *auto*
qed

From the premises stated in *diff-group-add-pert-hom-bound-exist*, α is nilpotent

lemma *α-locally-nilpotent*:
assumes *diff-group-add-pert-hom-bound-exist*:
diff-group-add-pert-hom-bound-exist (*op* −)
(*λx::'a::diff-group-add-pert-hom-bound-exist. − x*) *0* (*op* +) *diff pert hom-oper*
shows (*α*ˆ(*local-bound α x*)) (*x::'a::diff-group-add-pert-hom-bound-exist*) = *0*
unfolding *diff-group-add-pert-hom-bound-exist-impl-local-bound-is-Least*
[*OF diff-group-add-pert-hom-bound-exist*, *of x*]
proof (*rule LeastI-ex*)
show ∃ *k*. (*α* ˆ *k*) *x* = (*0::'a*)
proof −
obtain *n* **where** *n-bound*: (*α*ˆ *n*) (*x::'a::diff-group-add-pert-hom-bound-exist*) = *0*
using *diff-group-add-pert-hom-bound-exist-impl-local-bounded-func-alpha*
[*OF diff-group-add-pert-hom-bound-exist*]
unfolding *local-bounded-func-def* **by** *auto*

then show *?thesis* **using** *exI* **by** *auto*
qed
qed

The algebraic structure given by the endomorphisms of a *diff-group-add* with suitable operations is a ring

lemma (**in** *group-add*) **shows** *op* − = (λ*x y*. *x* + (− *y*))
unfolding *expand-fun-eq* **unfolding** *diff-minus* **by** *simp*

lemma (**in** *diff-group-add*) *hom-completion-ring*:
shows *ring* (|*carrier* = *hom-completion*
(*diff-group-functor* (λ*x*::′*a*. − *x*) (*op* −) *0* (*op* +) *diff*)
(*diff-group-functor* (λ*x*::′*a*. − *x*) (*op* −) *0* (*op* +) *diff*),
mult = *op* ∘,
one = λ*x*::′*a*. *if x* ∈ *carrier* (*diff-group-functor* (λ*x*::′*a*. − *x*) (*op* −) *0* (*op* +) *diff*) *then id x*
else one (*diff-group-functor* (λ*x*::′*a*. − *x*) (*op* −) *0* (*op* +) *diff*),
zero = λ*x*::′*a*. *if x* ∈ *carrier* (*diff-group-functor* (λ*x*::′*a*. − *x*) (*op* −) *0* (*op* +) *diff*)
then one (*diff-group-functor* (λ*x*::′*a*. − *x*) (*op* −) *0* (*op* +) *diff*)
else one (*diff-group-functor* (λ*x*::′*a*. − *x*) (*op* −) *0* (*op* +) *diff*),
add = λ(*f*::′*a* ⇒ ′*a*) (*g*::′*a* ⇒ ′*a*) *x*::′*a*.
if x ∈ *carrier* (*diff-group-functor* (λ*x*::′*a*. − *x*) (*op* −) *0* (*op* +) *diff*) *then*
mult (*diff-group-functor* (λ*x*::′*a*. − *x*) (*op* −) *0* (*op* +) *diff*) (*f x*) (*g x*)
else one (*diff-group-functor* (λ*x*::′*a*. − *x*) (*op* −) *0* (*op* +) *diff*)|)
using *diff-group-functor-preserves* [*OF prems*]
using *comm-group.hom-completion-ring*
[*of* (*diff-group-functor* (λ*x*::′*a*. − *x*) *op* − *0* (*op* +) *diff*)]
using *diff-minus*
unfolding *diff-group-def* **by** *simp*

lemma *homo-ab-is-hom-completion*:
assumes *homo-ab-f*: *homo-ab f*
and *diff-group-add*: *diff-group-add* (*op* −)
(λ*x*::′*a*::*diff-group-add*. − *x*) *0* (*op* +) *diff*
shows *f* ∈ *hom-completion*
(*diff-group-functor* (λ*x*::′*a*::*diff-group-add*. − *x*) (*op* −) *0* (*op* +) *diff*)
(*diff-group-functor* (λ*x*::′*a*. − *x*) (*op* −) *0* (*op* +) *diff*)
using *homo-ab-f*
using *diff-group-add*
unfolding *hom-completion-def*
unfolding *homo-ab-def*
unfolding *diff-group-functor-def*
unfolding *hom-def*
unfolding *completion-fun2-def*
unfolding *completion-def* **by** *auto*

lemma *hom-completion-is-homo-ab*:
assumes *f-hom-compl*: *f* ∈ *hom-completion*

(*diff-group-functor* (*λx::'a::diff-group-add. − x*) (*op −*) *0* (*op +*) *diff*)
(*diff-group-functor* (*λx::'a. − x*) (*op −*) *0* (*op +*) *diff*)
and *diff-group-add*: *diff-group-add* (*op −*) (*λx::'a::diff-group-add. − x*) *0* (*op +*) *diff*
shows *homo-ab f*
using *f-hom-compl*
using *diff-group-add*
unfolding *hom-completion-def*
unfolding *homo-ab-def*
unfolding *diff-group-functor-def*
unfolding *hom-def*
unfolding *completion-fun2-def*
unfolding *completion-def* **by** *auto*

lemma *hom-completion-equiv-homo-ab*:
assumes *diff-group-add*: *diff-group-add* (*op −*) (*λx::'a::diff-group-add. − x*) *0* (*op +*) *diff*
shows *homo-ab f* ⟷ *f* ∈ *hom-completion*
(*diff-group-functor* (*λx::'a::diff-group-add. − x*) (*op −*) *0* (*op +*) *diff*)
(*diff-group-functor* (*λx::'a. − x*) (*op −*) *0* (*op +*) *diff*)
using *homo-ab-is-hom-completion* [*of f*]
using *hom-completion-is-homo-ab* [*of f*]
using *diff-group-add* **by** *auto*

Equivalence between the definition of power in the Isabelle Algebra Library, *nat-pow-def*, over the ring of endomorphisms, and the definition of power for functions, definition *fun-pow*

definition *ring-hom-compl* :: (*'a diff-group*) => (*'a* => *'a*) *ring*
where *ring-hom-compl D* == (|*carrier* = *hom-completion D D*,
mult = *op* ∘,
one = *λx::'a. if x ∈ carrier D then id x else one D*,
zero = *λx::'a. if x ∈ carrier D then one D else one D*,
add = *λ(f::'a ⇒ 'a) (g::'a ⇒ 'a) x::'a. if x ∈ carrier D then mult D (f x) (g x) else one D*|)

lemma *ring-nat-pow-equiv-funpow*:
assumes *diff-group-add*: *diff-group-add* (*op −*) (*λx::'a::diff-group-add. − x*) *0* (*op +*) *diff*
and *f-hom-completion*: *f* ∈ *hom-completion*
(*diff-group-functor* (*λx::'a::diff-group-add. − x*) (*op −*) *0* (*op +*) *diff*)
(*diff-group-functor* (*λx::'a. − x*) (*op −*) *0* (*op +*) *diff*)
shows $f\ (\hat{})_{ring\text{-}hom\text{-}compl\ (diff\text{-}group\text{-}functor\ (\lambda x::'a::diff\text{-}group\text{-}add.\ -\ x)\ (op\ -)\ 0\ (op\ +)\ diff)}$ (*n::nat*) = *f^n*
(**is** $f\ (\hat{})_{?R}$ *n* = *f^n*)
proof (*induct n*)
case *0*
show $f(\hat{})_{?R}$ (*0::nat*) = *f^0*

unfolding *ring-hom-compl-def*
unfolding *nat-pow-def*
unfolding *diff-group-functor-def expand-fun-eq* **by** *auto*
next
case *Suc*
fix n :: *nat*
assume *hypo*: $f\ (\hat{\ })_{?R}\ n = f\hat{\ }n$
show $f\ (\hat{\ })_{?R}\ (Suc\ n) = f\hat{\ }(Suc\ n)$
proof $-$
have $f\hat{\ }(Suc\ n) = f \circ (f\hat{\ }n)$ **by** *simp*
also have $\ldots = mult\ ?R\ (f\ (\hat{\ })_{?R}\ (1::nat))\ (f\ (\hat{\ })_{?R}\ n)$
using *hypo analytic-part-local.monoid.nat-pow-1* [*of ?R f*]
using *f-hom-completion*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
unfolding *Ring.ring-def ring-hom-compl-def* **by** *simp*
also have $\ldots = f\ (\hat{\ })_{?R}\ (1 + n)$
using *monoid.nat-pow-mult* [*of ?R f 1 n*]
using *f-hom-completion diff-group-add.hom-completion-ring* [*OF diff-group-add*]
unfolding *Ring.ring-def ring-hom-compl-def* **by** *simp*
also have $\ldots = f\ (\hat{\ })_{?R}\ (Suc\ n)$ **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed

Equivalence between the *uminus* definition in the ring of endomorphisms, and the $-\ ?A = (\lambda x.\ -\ ?A\ x)$

lemma *minus-ring-homo-equal-uminus-fun*:
assumes *diff-group-add*: *diff-group-add* (*op* $-$) (λx::$'a$::*diff-group-add*. $-$ x) *0* (*op* $+$) *diff*
and *homo-ab-f*: *homo-ab f*
shows $\ominus_{ring\text{-}hom\text{-}compl}$ (*diff-group-functor* (λx::$'a$::*diff-group-add*. $-$ x) (*op* $-$) *0* (*op* $+$) *diff*) (f::$'a$::*diff-group-add* => $'a$) = $-$ f
(**is** $\ominus_{?R}\ f = -\ f$)
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
using *abelian-group.minus-equality* [*of ?R f* $-$ *f*]
using *homo-ab-is-hom-completion* [*OF homo-ab-f diff-group-add*]
using *homo-ab-is-hom-completion*
[*OF ab-group-add-impl-uminus-fun-closed*
[*OF - homo-ab-f*]]
using *diff-group-add*
unfolding *ring-hom-compl-def*
[*of diff-group-functor* (λx::$'a$::*diff-group-add*. $-$ x) (*op* $-$) *0* (*op* $+$) *diff*]
unfolding *Ring.ring-def*
unfolding *abelian-group-def*
unfolding *diff-group-add-def*
unfolding *ab-group-add-def*
unfolding *ab-group-add-axioms-def*
unfolding *fun-Compl-def*
unfolding *plus-fun-def zero-fun-def*

unfolding *diff-group-functor-def expand-fun-eq* **by** *auto*

lemma *minus-ring-hom-completion-equal-uminus-fun*:
assumes *diff-group-add*: *diff-group-add* (*op* −) ($\lambda x::'a::$*diff-group-add*. − *x*) *0*
(*op* +) *diff*
and *f-hom-completion*: *f* ∈ *hom-completion*
(*diff-group-functor* ($\lambda x::'a::$*diff-group-add*. − *x*) (*op* −) *0* (*op* +) *diff*)
(*diff-group-functor* ($\lambda x::'a$. − *x*) (*op* −) *0* (*op* +) *diff*)
shows $\ominus_{ring\text{-}hom\text{-}compl}$ (*diff-group-functor* ($\lambda x::'a::$*diff-group-add*. − *x*) (*op* −) *0* (*op* +) *diff*)
f = − *f*
(**is** $\ominus_{?R}$ *f* = − *f*)
using *minus-ring-homo-equal-uminus-fun*
using *hom-completion-equiv-homo-ab*
using *diff-group-add*
using *f-hom-completion* **by** *auto*

lemma α*-in-hom-completion*:
assumes *diff-group-add-pert-hom*:
diff-group-add-pert-hom (*op* −) ($\lambda x::'a::$*diff-group-add-pert-hom*. − *x*)
0 op + *diff pert hom-oper*
shows α ∈ *hom-completion*
(*diff-group-functor* ($\lambda x::'a::$*diff-group-add-pert-hom*. − *x*) *op* − *0 op* + *diff*)
(*diff-group-functor* ($\lambda x::'a$. − *x*) *op* − *0 op* + *diff*)
(**is** α ∈ *hom-completion ?D ?D*)
proof −
let *?R* = *ring-hom-compl ?D*
have *diff-group-add*: *diff-group-add op* − *uminus* (*0*::*'a*) *op* + *diff*
using *diff-group-add-pert-hom* **by** (*intro-locales*)
have *delta-in-R*: *pert* ∈ *carrier ?R* **and** *h-in-R*: *hom-oper* ∈ *carrier ?R*
using *diff-group-add-pert-hom*
unfolding *ring-hom-compl-def*
unfolding *reduction-class-ext-def*
unfolding *diff-group-add-pert-hom-def*
unfolding *diff-group-add-pert-def*
unfolding *diff-group-add-pert-axioms-def*
unfolding *diff-group-add-pert-hom-axioms-def*
unfolding *hom-completion-def*
unfolding *hom-def*
unfolding *Pi-def*
unfolding *completion-fun2-def completion-def*
unfolding *diff-group-functor-def* **by** *auto*
have *deltah-in-R*: *pert* ∘ *hom-oper* ∈ *carrier ?R*
using *delta-in-R*
using *h-in-R*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
using *monoid.m-closed* [*of ?R pert hom-oper*]
unfolding *Ring.ring-def ring-hom-compl-def* **by** *simp*
show *?thesis*
using *deltah-in-R*

using *abelian-group.a-inv-closed* [*OF - deltah-in-R*]
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
using *minus-ring-hom-completion-equal-uminus-fun*
[*of pert* ∘ *hom-oper*, *OF diff-group-add*]
unfolding *ring-hom-compl-def α-def Ring.ring-def fun-Compl-def* **by** *simp*
qed

lemma *β-in-hom-completion*:
assumes *diff-group-add-pert-hom*:
diff-group-add-pert-hom op − (*λx::'a::diff-group-add-pert-hom-bound-exist.* − *x*)
0 op + *diff pert hom-oper*
shows *β* ∈ *hom-completion*
(*diff-group-functor* (*λx::'a::diff-group-add-pert-hom-bound-exist.* − *x*) *op* − *0 op*
+ *diff*)
(*diff-group-functor* (*λx::'a.* − *x*) *op* − *0 op* + *diff*)
(**is** *β* ∈ *hom-completion ?D ?D*)
proof −
let *?R* = *ring-hom-compl ?D*
have *diff-group-add*: *diff-group-add op* − *uminus* (*0::'a*) *op* + *diff*
using *diff-group-add-pert-hom* **by** (*intro-locales*)
have *delta-in-R*: *pert* ∈ *carrier ?R* **and** *h-in-R*: *hom-oper* ∈ *carrier ?R*
using *diff-group-add-pert-hom*
unfolding *ring-hom-compl-def*
unfolding *reduction-class-ext-def*
unfolding *diff-group-add-pert-hom-def*
unfolding *diff-group-add-pert-def*
unfolding *diff-group-add-pert-axioms-def*
unfolding *diff-group-add-pert-hom-axioms-def*
unfolding *hom-completion-def*
unfolding *hom-def*
unfolding *Pi-def*
unfolding *completion-fun2-def completion-def*
unfolding *diff-group-functor-def* **by** *auto*
have *hdelta-in-R*: *hom-oper* ∘ *pert* ∈ *carrier ?R*
using *delta-in-R*
using *h-in-R*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
using *monoid.m-closed* [*of ?R hom-oper pert*]
unfolding *Ring.ring-def ring-hom-compl-def* **by** *simp*
show *?thesis*
using *hdelta-in-R*
using *abelian-group.a-inv-closed* [*OF - hdelta-in-R*]
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
using *minus-ring-hom-completion-equal-uminus-fun*
[*of hom-oper* ∘ *pert*, *OF diff-group-add*]
unfolding *ring-hom-compl-def β-def Ring.ring-def fun-Compl-def* **by** *simp*
qed

Our previous deinition of *reduction-class-ext* satisfies the definition of the locale *local-nilpotent-term*

```
lemma reduction-class-ext-preserves-local-nilpotent-term:
  assumes reduction-class-ext-f-g:
  reduction-class-ext (f::'a::diff-group-add-pert-hom-bound-exist => 'b::diff-group-add)
g
  shows local-nilpotent-term
  (diff-group-functor (λx::'a::diff-group-add-pert-hom-bound-exist. − x) op − 0 op
+ diff)
  (ring-hom-compl (diff-group-functor (λx::'a. − x) op − 0 op + diff))
  α (local-bound α)
  (is local-nilpotent-term ?D ?R α (local-bound α))
proof (intro-locales)
  have diff-group-add-pert-hom-bound-exist:
   diff-group-add-pert-hom-bound-exist op − (λx::'a. − x) 0 op + diff pert hom-oper
    using reduction-class-ext-f-g
    unfolding reduction-class-ext-def [of f g] by simp
  then have diff-group-add-pert-hom:
    diff-group-add-pert-hom op − (λx::'a. − x) 0 op + diff pert hom-oper
    unfolding diff-group-add-pert-hom-bound-exist-def by simp
  have diff-group-add: diff-group-add op − (λx::'a. − x) 0 op + diff
    using diff-group-add-pert-hom-bound-exist-impl-diff-group-add
    [OF diff-group-add-pert-hom-bound-exist]
    by simp
  show monoid ?D
    using diff-group-functor-preserves [OF diff-group-add]
    unfolding diff-group-def comm-group-def comm-monoid-def by fast
  show comm-monoid-axioms ?D
    using diff-group-functor-preserves [OF diff-group-add]
    unfolding diff-group-def comm-group-def comm-monoid-def by fast
  show group-axioms ?D
    using diff-group-functor-preserves [OF diff-group-add]
    unfolding diff-group-def comm-group-def group-def by simp
  show diff-group-axioms ?D
    using diff-group-functor-preserves [OF diff-group-add]
    unfolding diff-group-def by simp
  show abelian-monoid ?R
    using diff-group-add.hom-completion-ring [OF diff-group-add]
    unfolding ring-hom-compl-def Ring.ring-def abelian-group-def by simp
  show abelian-group-axioms ?R
    using diff-group-add.hom-completion-ring [OF diff-group-add]
    unfolding ring-hom-compl-def Ring.ring-def abelian-group-def by simp
  show ring-axioms ?R
    using diff-group-add.hom-completion-ring [OF diff-group-add]
    unfolding Ring.ring-def ring-hom-compl-def by simp
  show monoid ?R
    using diff-group-add.hom-completion-ring [OF diff-group-add]
    unfolding Ring.ring-def abelian-group-def ring-hom-compl-def by simp
  show ring-endomorphisms-axioms ?D ?R
    using diff-group-add.hom-completion-ring [OF diff-group-add]
```

unfolding *ring-hom-compl-def ring-endomorphisms-axioms-def* **by** *simp*
show *local-nilpotent-term-axioms ?D ?R* α (*local-bound* α)
proof (*unfold local-nilpotent-term-axioms-def*, *intro conjI*)
show *alpha-in-R*: $\alpha \in$ *carrier ?R*
using α*-in-hom-completion* [*OF diff-group-add-pert-hom*]
unfolding α*-def Ring.ring-def ring-hom-compl-def* **by** *simp*
show $\forall x \in$ *carrier ?D*. (α (^)$_{?R}$ (*local-bound* α x)) x = *one ?D*
proof (*rule ballI*)
fix x
assume *x-in-D*: $x \in$ *carrier ?D*
show (α (^)$_{?R}$ (*local-bound* α x)) x = *monoid.one ?D*
proof −
have (α (^)$_{?R}$ (*local-bound* α x)) x = (α^(*local-bound* α x)) x
using *ring-nat-pow-equiv-funpow* [*OF diff-group-add*, *of* α] *alpha-in-R*
unfolding *ring-hom-compl-def* **by** *simp*
also have (α ^ *local-bound* α (x::'*a*::*diff-group-add-pert-hom-bound-exist*))
x = *0*
using α*-locally-nilpotent* [*OF diff-group-add-pert-hom-bound-exist*, *of* x::'*a*]
by *simp*
also have ... = *one ?D* **unfolding** *diff-group-functor-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed
show $\forall x$::'*a*. (*local-bound* α x) = (*LEAST n*::*nat*. (α (^)$_{?R}$ n) x = *one ?D*)
using *diff-group-add-pert-hom-bound-exist-impl-local-bounded-func-alpha*
[*OF diff-group-add-pert-hom-bound-exist*]
using *local-bounded-func-impl-terminates-loop* [*of* α::'*a* ⇒ '*a*]
using *local-bound-correct* [*of* α::'*a* ⇒ '*a*]
using *ring-nat-pow-equiv-funpow* [*OF diff-group-add*, *of* α:: '*a* ⇒ '*a*]
using *alpha-in-R*
unfolding *ring-hom-compl-def diff-group-functor-def* **by** *auto*
qed
qed

The following lemma states that the *reduction-class-ext* definition together with *local-bound-exists* satisifies the premises of the *BPL ?D ?R ?h ?C ?f ?g ?*δ *?bound-phi* $\Longrightarrow$ *reduction* (*lemma-2-2-15.D*$'$ *?D ?R ?*δ) (|*carrier* = *carrier ?C*, *mult* = *op* $\otimes_{?C}$, *one* = $\mathbf{1}_{?C}$, *diff-group.diff* = λx. *if* $x \in$ *carrier ?C then diff-group.diff ?C x* $\otimes_{?C}$ (*?f* $\circ$ *?*δ $\circ$ *local-nilpotent-alpha.*Ψ *?D ?R ?h ?*δ $\circ$ *?g*) *x else* $\mathbf{1}_{?C}$|) (*?f* $\circ$ *local-nilpotent-alpha.*Φ *?D ?R ?h ?*δ *?bound-phi*) (*local-nilpotent-alpha.*Ψ *?D ?R ?h ?*δ $\circ$ *?g*) (*lemma-2-2-15.h*$'$ *?D ?R ?h ?*δ *?bound-phi*).

In addition to this result, we also have to prove later that the definitions given in this file for f', g', Φ, are equivalent to the ones given inside of the local BPL

lemma *reduction-class-ext-preserves-BPL*:
assumes *r-c-e*:

reduction-class-ext (*f*::*'a*::*diff-group-add-pert-hom-bound-exist* => *'b*::*diff-group-add*) *g*
shows *BPL*
(*diff-group-functor* (λx::*'a*::*diff-group-add-pert-hom-bound-exist*. $-$ *x*) *op* $-$ *0* *op* + *diff*)
(*ring-hom-compl* (*diff-group-functor* (λx::*'a*. $-$ *x*) *op* $-$ *0* *op* + *diff*))
hom-oper
(*diff-group-functor* (λx::*'b*::*diff-group-add*. $-$ *x*) *op* $-$ *0* *op* + *diff*)
f g pert
(*local-bound* α)
(**is** *BPL ?D ?R hom-oper ?C f g pert* (*local-bound* α))
proof (*intro-locales*)
have *diff-group-add-pert-hom-bound-exist*:
diff-group-add-pert-hom-bound-exist op $-$ (λx::*'a*. $-$ *x*) *0 op* + *diff pert hom-oper*
using *r-c-e*
unfolding *reduction-class-ext-def* [*of f g*] **..**
have *diff-group-add*: *diff-group-add op* $-$ (λx::*'a*. $-$ *x*) *0 op* + *diff*
using *diff-group-add-pert-hom-bound-exist-impl-diff-group-add*
[*OF diff-group-add-pert-hom-bound-exist*] **.**
show *monoid ?D*
using *diff-group-functor-preserves* [*OF diff-group-add*]
unfolding *diff-group-def comm-group-def*
unfolding *comm-monoid-def* **by** *fast*
show *comm-monoid-axioms ?D*
using *diff-group-functor-preserves* [*OF diff-group-add*]
unfolding *diff-group-def comm-group-def*
unfolding *comm-monoid-def* **by** *fast*
show *group-axioms ?D*
using *diff-group-functor-preserves* [*OF diff-group-add*]
unfolding *diff-group-def comm-group-def group-def* **by** *fast*
show *diff-group-axioms ?D*
using *diff-group-functor-preserves* [*OF diff-group-add*]
unfolding *diff-group-def* **..**
have *diff-group-add-C*: *diff-group-add op* $-$ (λx::*'b*. $-$ *x*) *0 op* + *diff*
using *r-c-e*
unfolding *reduction-class-ext-def* **by** *simp*
show *monoid ?C*
using *diff-group-functor-preserves* [*OF diff-group-add-C*]
unfolding *diff-group-def*
unfolding *comm-group-def*
unfolding *comm-monoid-def* **by** *fast*
show *comm-monoid-axioms ?C*
using *diff-group-functor-preserves* [*OF diff-group-add-C*]
unfolding *diff-group-def*
unfolding *comm-group-def*
unfolding *comm-monoid-def* **by** *fast*
show *group-axioms ?C*
using *diff-group-functor-preserves* [*OF diff-group-add-C*]
unfolding *diff-group-def*

unfolding *comm-group-def*
unfolding *group-def* **by** *fast*
show *diff-group-axioms ?C*
using *diff-group-functor-preserves* [*OF diff-group-add-C*]
unfolding *diff-group-def* **by** *fast*
show *abelian-monoid ?R*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
unfolding *ring-hom-compl-def*
unfolding *Ring.ring-def*
unfolding *abelian-group-def* **by** *fast*
show *abelian-group-axioms ?R*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
unfolding *Ring.ring-def*
unfolding *abelian-group-def*
unfolding *ring-hom-compl-def* **by** *fast*
show *ring-axioms ?R*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
unfolding *ring-hom-compl-def*
unfolding *Ring.ring-def* **by** *fast*
show *monoid ?R*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
unfolding *Ring.ring-def*
unfolding *abelian-group-def*
unfolding *ring-hom-compl-def* **by** *fast*
show *ring-endomorphisms-axioms ?D ?R*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
unfolding *ring-hom-compl-def*
unfolding *ring-endomorphisms-axioms-def* **by** *fast*
show *lemma-2-2-14-axioms ?D ?R* ($\lambda x.\ h\ x$)
proof (*unfold lemma-2-2-14-axioms-def ring-hom-compl-def*, *simp*, *intro conjI*)
show *hom-oper* ∈ *hom-completion ?D ?D*
using *r-c-e*
unfolding *reduction-class-ext-def*
unfolding *diff-group-add-pert-hom-bound-exist-def*
unfolding *diff-group-add-pert-hom-def*
unfolding *diff-group-add-pert-hom-axioms-def*
unfolding *hom-completion-def*
unfolding *hom-def*
unfolding *Pi-def*
unfolding *completion-fun2-def*
unfolding *completion-def*
unfolding *diff-group-functor-def* **by** *simp*
show ($\lambda x.\ h\ x$) ∘ ($\lambda x.\ h\ x$) = ($\lambda x{::}'a.\ one\ ?D$)
using *r-c-e*
unfolding *reduction-class-ext-def*
unfolding *diff-group-add-pert-hom-bound-exist-def*
unfolding *diff-group-add-pert-hom-def*
unfolding *diff-group-add-pert-hom-axioms-def*
unfolding *zero-fun-def*

unfolding *diff-group-functor-def* **by** *simp*
show $(\lambda x.\ h\ x) \circ$ *diff-group.diff ?D* $\circ (\lambda x.\ h\ x) = (\lambda x.\ h\ x)$
unfolding *diff-group-functor-def*
apply *simp*
using *hdh-eq-h* [*OF r-c-e*] **.**
qed
show *reduction-axioms ?D ?C f g* $(\lambda x.\ h\ x)$
using *reduction-class-ext-preserves-reduction* [*OF r-c-e*]
unfolding *reduction-def* **by** *fast*
show *alpha-beta-axioms ?D* $(\lambda x.\ \delta\ x)$
using *diff-group-add-pert-hom-bound-exist-preserves-pert*
[*OF diff-group-add-pert-hom-bound-exist*]
unfolding *alpha-beta-axioms-def* **.**
show *local-nilpotent-term-axioms*
?D ?R ($\ominus_{?R}$ *mult ?R* $(\lambda x.\ \delta\ x)$ $(\lambda x.\ h\ x)$) (*local-bound* α)
proof −
have *delta-in-R*: $(\lambda x.\ \delta\ x) \in$ *carrier ?R*
and *h-in-R*: $(\lambda x.\ h\ x) \in$ *carrier ?R*
using *r-c-e*
unfolding *ring-hom-compl-def*
unfolding *reduction-class-ext-def*
unfolding *diff-group-add-pert-hom-bound-exist-def*
unfolding *diff-group-add-pert-hom-def*
unfolding *diff-group-add-pert-hom-axioms-def*
unfolding *diff-group-add-pert-def*
unfolding *diff-group-add-pert-axioms-def*
unfolding *hom-completion-def*
unfolding *hom-def*
unfolding *Pi-def*
unfolding *completion-fun2-def*
unfolding *completion-def*
unfolding *diff-group-functor-def* **by** *simp-all*
have *deltah-in-R*: $(\lambda x.\ \delta\ x) \circ (\lambda x.\ h\ x) \in$ *carrier ?R*
using *delta-in-R h-in-R*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
using *monoid.m-closed* [*of ?R* $\lambda x.\ \delta\ x$ $\lambda x.\ h\ x$]
unfolding *ring-hom-compl-def*
unfolding *Ring.ring-def* **by** *simp*
have *minus-eq*: ($\ominus_{?R}$ *mult ?R* $(\lambda x.\ \delta\ x)$ $(\lambda x.\ h\ x)$)
$= -\ ((\lambda x.\ \delta\ x) \circ (\lambda x.\ h\ x))$
using *deltah-in-R*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
using *abelian-group.a-inv-closed* [*OF - deltah-in-R*]
using *minus-ring-hom-completion-equal-uminus-fun*
[*OF diff-group-add*, *of* $(\lambda x.\ \delta\ x) \circ (\lambda x.\ h\ x)$]
unfolding *ring-hom-compl-def* **by** *simp*
show *?thesis*
using *reduction-class-ext-preserves-local-nilpotent-term*
[*OF r-c-e*]

unfolding *minus-eq*
unfolding *local-nilpotent-term-def*
unfolding *α-def*
unfolding *fun-Compl-def* **unfolding** *o-apply* **..**
qed
qed

The definition of *reduction-class-ext* satisfies the definition of the locale *lemma-2-2-15*.

lemma *reduction-class-ext-preserves-lemma-2-2-15*:
assumes *r-c-e*:
reduction-class-ext (*f*::*'a*::*diff-group-add-pert-hom-bound-exist*
=> *'b*::*diff-group-add*) *g*
shows *lemma-2-2-15*
(*diff-group-functor* (*λx*::*'a*::*diff-group-add-pert-hom-bound-exist*. − *x*)
op − *0 op* + *diff*)
(*ring-hom-compl* (*diff-group-functor* (*λx*::*'a*. − *x*) *op* − *0 op* + *diff*))
hom-oper (*diff-group-functor* (*λx*::*'b*::*diff-group-add*. − *x*) *op* − *0 op* + *diff*)
f g pert
(*local-bound α*)
using *reduction-class-ext-preserves-BPL* [*OF r-c-e*]
unfolding *BPL-def*
unfolding *lemma-2-2-17-def*
unfolding *proposition-2-2-16-def* **..**

The definition of *reduction-class-ext* satisfies the definiton of the locale *local-nilpotent-alpha*

lemma *reduction-class-ext-preserves-local-nilpotent-alpha*:
assumes *r-c-e*:
reduction-class-ext (*f*::*'a*::*diff-group-add-pert-hom-bound-exist*
=> *'b*::*diff-group-add*) *g*
shows *local-nilpotent-alpha*
(*diff-group-functor* (*λx*::*'a*::*diff-group-add-pert-hom-bound-exist*. − *x*)
op − *0 op* + *diff*)
(*ring-hom-compl* (*diff-group-functor* (*λx*::*'a*. − *x*) *op* − *0 op* + *diff*))
(*diff-group-functor* (*λx*::*'b*::*diff-group-add*. − *x*) *op* − *0 op* + *diff*)
f g hom-oper pert
(*local-bound α*)
using *reduction-class-ext-preserves-BPL* [*OF r-c-e*]
unfolding *BPL-def*
unfolding *lemma-2-2-17-def*
unfolding *proposition-2-2-16-def*
unfolding *lemma-2-2-15-def* **by** *fast*

The definition *λx*. *fin-sum α* (*local-bound α x*) *x* in *reduction-class-ext* is equivalent to the previous definition of *power-series* in locale *local-nilpotent-term*.

lemma *reduction-class-ext-preserves-power-series*:
assumes *r-c-e*:

```
  reduction-class-ext (f::'a::diff-group-add-pert-hom-bound-exist
  => 'b::diff-group-add) g
  shows local-nilpotent-term.power-series
  (diff-group-functor (λx::'a::diff-group-add-pert-hom-bound-exist. − x) op − 0 op
+ diff)
  (ring-hom-compl (diff-group-functor (λx::'a. − x) op − 0 op + diff))
  α
  (local-bound α) = (λx::'a. fin-sum α (local-bound α x) x)
  (is local-nilpotent-term.power-series ?D ?R α (local-bound α)
    = (λx. fin-sum α (local-bound α x) x))
proof (unfold expand-fun-eq, rule allI)
  fix x :: 'a
  show local-nilpotent-term.power-series ?D ?R α (local-bound α) x
    = fin-sum α (local-bound α x) x
  proof −
    have local-nilpotent-term: local-nilpotent-term ?D ?R α (local-bound α)
      using reduction-class-ext-preserves-local-nilpotent-term [OF r-c-e] .
    have diff-group-add-pert-hom-bound-exist:
        diff-group-add-pert-hom-bound-exist op − (λx::'a. − x) 0 op + diff pert
hom-oper
      using r-c-e
      unfolding reduction-class-ext-def [of f g] ..
    have diff-group-add-pert-hom:
      diff-group-add-pert-hom op − (λx::'a. − x) 0 op + diff pert hom-oper
      using diff-group-add-pert-hom-bound-exist
      by (intro-locales)
    have diff-group-add: diff-group-add op − (λx::'a. − x) 0 op + diff
      using diff-group-add-pert-hom-bound-exist-impl-diff-group-add
      [OF diff-group-add-pert-hom-bound-exist] .
    have local-nilpotent-term.power-series ?D ?R α (local-bound α) x
      = (⨂ ?D i::nat∈{..local-bound α x}. (α (^) ?R i) x)
      using local-nilpotent-term.power-series-def [OF local-nilpotent-term]
      by simp
    also have (⨂ ?D i::nat∈{..local-bound α x}. (α (^) ?R i) x)
      = (⨂ ?D i::nat∈{..local-bound α x}. (α ^i) x)
      unfolding ring-nat-pow-equiv-funpow [OF diff-group-add
        α-in-hom-completion [OF diff-group-add-pert-hom]] ..
    also have ... = fin-sum α (local-bound α x) x
    proof (induct local-bound α x)
      case 0
      {
        have (⨂ ?D i::nat∈{..0::nat}. (α ^i) x) = (α ^(0::nat)) x
          using comm-monoid.finprod-0 [of ?D]
          using diff-group-functor-preserves [OF diff-group-add]
          unfolding diff-group-def
          unfolding comm-group-def
          unfolding diff-group-functor-def by simp
        also have ... = fin-sum α (0::nat) x by simp
        finally show (⨂ ?D i::nat∈{..0::nat}. (α ^i) x) = fin-sum α (0::nat) x .
```

}
next
case *Suc*
{
fix n :: *nat*
assume *hypo*: ($\bigotimes_{?D} i::nat \in \{..n\}$. $(\alpha\,\hat{}\,i)\ x) = \textit{fin-sum}\ \alpha\ n\ x$
have ($\bigotimes_{?D} i::nat \in \{..Suc\ n\}$. $(\alpha\,\hat{}\,i)\ x)$
= *monoid.mult ?D* $((\alpha\,\hat{}\,(Suc\ n))\ x)$ ($\bigotimes_{?D} i::nat \in \{..n\}$. $(\alpha\,\hat{}\,i)\ x)$
using *comm-monoid.finprod-Suc* [*of ?D*]
using *diff-group-functor-preserves* [*OF diff-group-add*]
unfolding *diff-group-def*
unfolding *comm-group-def*
unfolding *diff-group-functor-def* by *simp*
also have ... = *monoid.mult ?D* $((\alpha\,\hat{}\,(Suc\ n))\ x)$ (*fin-sum* α n x)
unfolding *hypo* ..
also have ... = $((\alpha\,\hat{}\,(Suc\ n))\ x)$ + (*fin-sum* α n x)
unfolding *diff-group-functor-def*
unfolding *monoid.select-convs* (*1*) ..
also have ... = $((\alpha\,\hat{}\,(Suc\ n))$ + (*fin-sum* α (n))) x
unfolding *plus-fun-def* [*of* $((\alpha::'a \Rightarrow 'a)\,\hat{}\,(Suc\ n))$ (*fin-sum* α (n))] ..
also have ... = (*fin-sum* α (*Suc* n) x)
unfolding *fin-sum.simps* (*2*) ..
finally
show ($\bigotimes_{?D} i::nat \in \{..Suc\ (n::nat)\}$. $(\alpha\,\hat{}\,i)\ (x::'a)) = \textit{fin-sum}\ \alpha\ (Suc\ n)\ x$.
}
qed
finally show *?thesis* .
qed
qed

The definition of $\Phi = (\lambda x.\ \textit{fin-sum}\ \alpha\ (\textit{local-bound}\ \alpha\ x)\ x)$ is equivalent to the previous definition of *D-R-C-f-g-h-δ-α-bound-phi*.Φ in *locale-nilpotent-alpha*

lemma *reduction-class-ext-preserves-Φ*:
assumes *r-c-e*:
reduction-class-ext (*f*::'*a*::*diff-group-add-pert-hom-bound-exist*
=> '*b*::*diff-group-add*) *g*
shows *local-nilpotent-alpha*.Φ
(*diff-group-functor* (λ*x*::'*a*::*diff-group-add-pert-hom-bound-exist*. − *x*)
op − *0 op* + *diff*)
(*ring-hom-compl* (*diff-group-functor* (λ*x*::'*a*. − *x*) *op* − *0 op* + *diff*))
hom-oper
pert
(*local-bound* α) = Φ
(**is** *local-nilpotent-alpha*.Φ *?D ?R hom-oper pert* (*local-bound* α) = Φ)
unfolding *local-nilpotent-alpha.phi-def*
[*OF reduction-class-ext-preserves-local-nilpotent-alpha*
[*OF r-c-e*]]
unfolding *local-nilpotent-term.power-series-def* [*OF* -, *of x*]
proof −

show *local-nilpotent-term.power-series ?D ?R*
($\ominus_{?R}$ *monoid.mult ?R pert hom-oper*) (*local-bound* α) = Φ
proof −
have *local-nilpotent-term.power-series ?D ?R*
($\ominus_{?R}$ *monoid.mult ?R* ($\lambda x.\ \delta\ x$) ($\lambda x.\ h\ x$)) (*local-bound* α) =
local-nilpotent-term.power-series ?D ?R α (*local-bound* α)
proof −
have *diff-group-add-pert-hom-bound-exist*:
diff-group-add-pert-hom-bound-exist op − ($\lambda x::'a.\ -\ x$) *0*
op + *diff pert hom-oper*
using *r-c-e*
unfolding *reduction-class-ext-def* [*of f g*] **by** *fast*
have *diff-group-add*: *diff-group-add op* − ($\lambda x::'a.\ -\ x$) *0 op* + *diff*
using *diff-group-add-pert-hom-bound-exist-impl-diff-group-add*
[*OF diff-group-add-pert-hom-bound-exist*] **.**
have *delta-in-R*: *pert* ∈ *carrier ?R* **and** *h-in-R*: ($\lambda x.\ h\ x$) ∈ *carrier ?R*
using *r-c-e*
unfolding *ring-hom-compl-def*
unfolding *reduction-class-ext-def*
unfolding *diff-group-add-pert-hom-bound-exist-def*
unfolding *diff-group-add-pert-hom-def*
unfolding *diff-group-add-pert-hom-axioms-def*
unfolding *diff-group-add-pert-def*
unfolding *diff-group-add-pert-axioms-def*
unfolding *hom-completion-def*
unfolding *hom-def*
unfolding *Pi-def*
unfolding *completion-fun2-def*
unfolding *completion-def*
unfolding *diff-group-functor-def* **by** *auto*
have *deltah-in-R*: ($\lambda x.\ \delta\ x$) ∘ ($\lambda x.\ h\ x$) ∈ *carrier ?R*
using *delta-in-R h-in-R*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
using *monoid.m-closed* [*of ?R* ($\lambda x.\ \delta\ x$) ($\lambda x.\ h\ x$)]
unfolding *Ring.ring-def ring-hom-compl-def* **by** *simp*
have *minus-equiv*: $\ominus_{?R}$ *mult ?R* ($\lambda x.\ \delta\ x$) ($\lambda x.\ h\ x$) = α
using *abelian-group.a-inv-closed* [*OF - deltah-in-R*]
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
using *minus-ring-hom-completion-equal-uminus-fun*
[*of* ($\lambda x.\ \delta\ x$) ∘ ($\lambda x.\ h\ x$), *OF diff-group-add*]
using *deltah-in-R*
unfolding α*-def*
unfolding *ring-hom-compl-def*
unfolding *fun-Compl-def* **by** *simp*
show *?thesis* **unfolding** *minus-equiv* **..**
qed
also have . . . = ($\lambda x::'a.$ (*fin-sum* α (*local-bound* α *x*)) *x*)
unfolding *reduction-class-ext-preserves-power-series*
[*OF r-c-e*] **..**

also have ... = Φ **unfolding** *Φ-def α-def* **..**
finally show *?thesis* **.**
qed
qed

Now, as a corollary, we prove that the previous definition of the output *D-R-h-C-f-g-δ-α-bound-phi.f′* of the *BPL*, is equivalent to the definition $f \circ \Phi$.

corollary *reduction-class-ext-preserves-output-f*:
assumes *r-c-e*: *reduction-class-ext* (*f*::*′a*::*diff-group-add-pert-hom-bound-exist* => *′b*::*diff-group-add*) *g*
shows *f* ∘
local-nilpotent-alpha.Φ
(*diff-group-functor* (λ*x*::*′a*::*diff-group-add-pert-hom-bound-exist*. − *x*) *op* − *0 op* + *diff*)
(*ring-hom-compl* (*diff-group-functor* (λ*x*::*′a*. − *x*) *op* − *0 op* + *diff*))
hom-oper
pert
(*local-bound* α)
= *f* ∘ Φ
unfolding *reduction-class-ext-preserves-Φ* [*OF r-c-e*] **..**

Now, as a corollary, we prove that the previous definition of the output h' of the *BPL*, is equivalent to the definition $h' \equiv$ *hom-oper* ∘ Φ.

corollary *reduction-class-ext-preserves-output-h*:
assumes *r-c-e*:
reduction-class-ext (*f*::*′a*::*diff-group-add-pert-hom-bound-exist* => *′b*::*diff-group-add*) *g*
shows *lemma-2-2-15.h′*
(*diff-group-functor* (λ*x*::*′a*::*diff-group-add-pert-hom-bound-exist*. − *x*) *op* − *0 op* + *diff*)
(*ring-hom-compl* (*diff-group-functor* (λ*x*::*′a*. − *x*) *op* − *0 op* + *diff*))
hom-oper
pert
(*local-bound* α)
= *h′*
unfolding *lemma-2-2-15.h′-def*
[*OF reduction-class-ext-preserves-lemma-2-2-15*
[*OF r-c-e*]]
unfolding *reduction-class-ext-preserves-Φ* [*OF r-c-e*]
unfolding *h′-def*
unfolding *ring-hom-compl-def*
unfolding *monoid.select-convs* (*1*) **..**

The definition of *reduction-class-ext* satisfies the definition of the locale *alpha-beta*

lemma *reduction-class-ext-preserves-alpha-beta*:
assumes *r-c-e*:

reduction-class-ext (*f*::'*a*::*diff-group-add-pert-hom-bound-exist*
=> '*b*::*diff-group-add*) *g*
shows *alpha-beta* (*diff-group-functor* (λ*x*::'*a*. − *x*) *op* − *0 op* + *diff*)
(*ring-hom-compl*
(*diff-group-functor* (λ*x*::'*a*::*diff-group-add-pert-hom-bound-exist*. − *x*) *op* − *0 op*
+ *diff*))
(*diff-group-functor* (λ*x*::'*b*::*diff-group-add*. − *x*) *op* − *0 op* + *diff*)
f
g (λ*x*. *h x*) (λ*x*. δ *x*)
using *reduction-class-ext-preserves-BPL* [*OF r-c-e*]
unfolding *BPL-def*
unfolding *lemma-2-2-17-def*
unfolding *proposition-2-2-16-def*
unfolding *lemma-2-2-15-def*
unfolding *lemma-2-2-14-def*
unfolding *local-nilpotent-alpha-def* **by** *fast*

The new definition of the power series over β = − (*hom-oper* ∘ *diff-group-add-pert-class.pert*) is equivalent to the definition of the power series over β in the previous version.

lemma *reduction-class-ext-preserves-beta-bound*:
assumes *r-c-e*:
reduction-class-ext (*f*::'*a*::*diff-group-add-pert-hom-bound-exist*
=> '*b*::*diff-group-add*) *g*
shows *local-bounded-func* (β::'*a*::*diff-group-add-pert-hom-bound-exist* => '*a*)
proof −
let *?D* = (*diff-group-functor* (λ*x*::'*a*. − *x*) *op* − *0 op* + *diff*)
let *?R* = *ring-hom-compl ?D*
obtain *bound-psi*
where *local-nilp-term*:
local-nilpotent-term ?D ?R (*alpha-beta*.β *?R hom-oper pert*) *bound-psi*
using *local-nilpotent-alpha.bound-psi-exists*
[*OF reduction-class-ext-preserves-local-nilpotent-alpha*
[*OF r-c-e*]] **by** *auto*
have *diff-group-add-pert-hom-bound-exist*:
diff-group-add-pert-hom-bound-exist op − (λ*x*::'*a*. − *x*) *0 op* + *diff pert hom-oper*
using *r-c-e*
unfolding *reduction-class-ext-def* [*of f g*] **by** *fast*
have *diff-group-add*: *diff-group-add op* − (λ*x*::'*a*. − *x*) *0 op* + *diff*
using *diff-group-add-pert-hom-bound-exist-impl-diff-group-add*
[*OF diff-group-add-pert-hom-bound-exist*] **.**
have *delta-in-R*: *pert* ∈ *carrier ?R* **and** *h-in-R*: *hom-oper* ∈ *carrier ?R*
using *r-c-e*
unfolding *ring-hom-compl-def*
unfolding *reduction-class-ext-def*
unfolding *diff-group-add-pert-hom-bound-exist-def*
unfolding *diff-group-add-pert-hom-def*
unfolding *diff-group-add-pert-hom-axioms-def*
unfolding *diff-group-add-pert-def*

unfolding *diff-group-add-pert-axioms-def*
unfolding *hom-completion-def*
unfolding *hom-def*
unfolding *Pi-def*
unfolding *completion-fun2-def*
unfolding *completion-def*
unfolding *diff-group-functor-def* **by** *auto*
have *hdelta-in-R*: $(\lambda x.\ h\ x) \circ (\lambda x.\ \delta\ x) \in$ *carrier ?R*
using *delta-in-R h-in-R*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
using *monoid.m-closed* [*of ?R* $(\lambda x.\ h\ x)$ $(\lambda x.\ \delta\ x)$]
unfolding *Ring.ring-def ring-hom-compl-def* **by** *simp*
have *β-equiv*: *alpha-beta.β ?R* $(\lambda x.\ h\ x)$ $(\lambda x.\ \delta\ x) = \beta$
proof −
have *alpha-beta.β ?R* $(\lambda x.\ h\ x)$ $(\lambda x.\ \delta\ x)$
$= \ominus_{?R}$ *monoid.mult ?R* $(\lambda x.\ h\ x)$ $(\lambda x.\ \delta\ x)$
unfolding *alpha-beta.beta-def*
[*OF reduction-class-ext-preserves-alpha-beta*
[*OF r-c-e*]] **..**
also have $\ominus_{?R}$ *monoid.mult ?R* $(\lambda x.\ h\ x)$ $(\lambda x.\ \delta\ x)$
$= - ((\lambda x.\ h\ x) \circ (\lambda x.\ \delta\ x))$
using *hdelta-in-R*
using *minus-ring-hom-completion-equal-uminus-fun*
[*OF diff-group-add*]
unfolding *ring-hom-compl-def* **by** *simp*
finally show *?thesis* **unfolding** *β-def* **.**
qed
have *lnt*: *local-nilpotent-term ?D ?R β bound-psi*
unfolding *sym* [*OF β-equiv*]
using *local-nilp-term* **.**
have *bound-ex*:$\forall x{::}'a.\ \exists n{::}nat.\ (\beta\ (\hat{\ })_{?R}\ n)\ x = 0$
using *lnt*
unfolding *local-nilpotent-term-def*
unfolding *local-nilpotent-term-axioms-def*
unfolding *diff-group-functor-def* **by** *auto*
have $\forall x{::}'a.\ \exists n{::}nat.\ (\beta\ \hat{\ }\ n)\ x = 0$
using *ring-nat-pow-equiv-funpow*
[*OF diff-group-add β-in-hom-completion*]
using *diff-group-add-pert-hom-bound-exist*
using *bound-ex*
unfolding *diff-group-add-pert-hom-bound-exist-def* **by** *simp*
then show *?thesis*
unfolding *local-bounded-func-def* **.**
qed

lemma *reduction-class-ext-preserves-power-series-β*:
assumes *r-c-e*:
reduction-class-ext (*f*::'*a*::*diff-group-add-pert-hom-bound-exist*
=> '*b*::*diff-group-add*) *g*

shows *local-nilpotent-term.power-series*
(*diff-group-functor* (λx::'*a*::*diff-group-add-pert-hom-bound-exist*. − *x*) *op* − *0 op* + *diff*)
(*ring-hom-compl* (*diff-group-functor* (λx::'*a*. − *x*) *op* − *0 op* + *diff*))
β (*local-bound* β)
= (λx::'*a*. *fin-sum* β (*local-bound* β *x*) *x*)
(**is** *local-nilpotent-term.power-series ?D ?R* β (*local-bound* β)
= (λx::'*a*. *fin-sum* β (*local-bound* β *x*) *x*))
proof (*unfold expand-fun-eq*, *rule allI*)
fix *x* :: '*a*
show *local-nilpotent-term.power-series ?D ?R* β (*local-bound* β) *x*
= *fin-sum* β (*local-bound* β *x*) *x*
proof −
have *diff-group-add-pert-hom-bound-exist*:
diff-group-add-pert-hom-bound-exist op − (λx::'*a*. − *x*) *0 op* + *diff pert hom-oper*
using *r-c-e*
unfolding *reduction-class-ext-def* [*of f g*] **by** *fast*
have *diff-group-add*: *diff-group-add op* − (λx::'*a*. − *x*) *0 op* + *diff*
using *diff-group-add-pert-hom-bound-exist-impl-diff-group-add*
[*OF diff-group-add-pert-hom-bound-exist*] **.**
have *delta-in-R*: *pert* ∈ *carrier ?R* **and** *h-in-R*: *hom-oper* ∈ *carrier ?R*
using *r-c-e*
unfolding *reduction-class-ext-def*
unfolding *ring-hom-compl-def*
unfolding *diff-group-add-pert-hom-bound-exist-def*
unfolding *diff-group-add-pert-hom-def*
unfolding *diff-group-add-pert-hom-axioms-def*
unfolding *diff-group-add-pert-def*
unfolding *diff-group-add-pert-axioms-def*
unfolding *hom-completion-def*
unfolding *hom-def*
unfolding *Pi-def*
unfolding *completion-fun2-def*
unfolding *completion-def*
unfolding *diff-group-functor-def* **by** *auto*
have *hdelta-in-R*: (λx. *h x*) ∘ (λx. δ *x*) ∈ *carrier ?R*
using *delta-in-R h-in-R*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
using *monoid.m-closed* [*of ?R* (λx. *h x*) (λx. δ *x*)]
unfolding *ring-hom-compl-def*
unfolding *Ring.ring-def* **by** *simp*
have β*-equiv*: *alpha-beta*.β *?R* (λx. *h x*) (λx. δ *x*) = β
proof −
have *alpha-beta*.β *?R* (λx. *h x*) (λx. δ *x*)
= $\ominus_{?R}$ *monoid.mult ?R* (λx. *h x*) (λx. δ *x*)
unfolding *alpha-beta.beta-def*
[*OF reduction-class-ext-preserves-alpha-beta*
[*OF r-c-e*]] **..**

also have $\ominus_{?R}$ *monoid.mult ?R* $(\lambda x.\ h\ x)\ (\lambda x.\ \delta\ x)$
$= -\ ((\lambda x.\ h\ x) \circ (\lambda x.\ \delta\ x))$
using *hdelta-in-R*
using *minus-ring-hom-completion-equal-uminus-fun*
[*of* $(\lambda x.\ h\ x) \circ (\lambda x.\ \delta\ x)$, *OF diff-group-add*]
unfolding *ring-hom-compl-def* **by** *simp*
finally show *?thesis* **unfolding** *β-def* **.**
qed
have *bound-equiv*: $(\lambda x.$ *LEAST n.* $(\beta\ (\hat{\ })_{?R}\ n)\ x =$ *one ?D*$) = ($*local-bound* $\beta)$
using *local-bounded-func-impl-local-bound-is-Least*
[*OF reduction-class-ext-preserves-beta-bound*
[*OF r-c-e*]]
using *ring-nat-pow-equiv-funpow* [*OF diff-group-add β-in-hom-completion*]
using *diff-group-add-pert-hom-bound-exist*
unfolding *diff-group-add-pert-hom-bound-exist-def*
unfolding *diff-group-functor-def* **by** (*simp add*: *expand-fun-eq*)
have *local-nilpotent-term*: *local-nilpotent-term ?D ?R* β (*local-bound* β)
using *local-nilpotent-alpha.nilp-alpha-nilp-beta*
[*OF reduction-class-ext-preserves-local-nilpotent-alpha*
[*OF r-c-e*]]
using *local-nilpotent-alpha.nilp-alpha-nilp-beta*
using *β-equiv*
using *bound-equiv* **by** *simp*
have *local-nilpotent-term.power-series ?D ?R* β (*local-bound* β) x
$= (\bigotimes_{?D} i{::}nat \in \{..\text{local-bound}\ \beta\ x\}.\ (\beta\ (\hat{\ })_{?R}\ i)\ x)$
unfolding *local-nilpotent-term.power-series-def* [*OF local-nilpotent-term*] **..**
also have $(\bigotimes_{?D} i{::}nat \in \{..\text{local-bound}\ \beta\ x\}.\ (\beta\ (\hat{\ })_{?R}\ i)\ x)$
$= (\bigotimes_{?D} i{::}nat \in \{..\text{local-bound}\ \beta\ x\}.\ (\beta\hat{\ }i)\ x)$
using *ring-nat-pow-equiv-funpow* [*OF diff-group-add β-in-hom-completion*]
using *diff-group-add-pert-hom-bound-exist*
unfolding *diff-group-add-pert-hom-bound-exist-def* **by** *simp*
also have ... = *fin-sum* β (*local-bound* β x) x
proof (*induct local-bound* β x)
case *0*
{
have $(\bigotimes_{?D} i{::}nat \in \{..0{::}nat\}.\ (\beta\hat{\ }i)\ x) = (\beta\hat{\ }(0{::}nat))\ x$
using *comm-monoid.finprod-0* [*of ?D*]
using *diff-group-functor-preserves* [*OF diff-group-add*]
unfolding *diff-group-def*
unfolding *comm-group-def*
unfolding *diff-group-functor-def* **by** *simp*
also have ... = *fin-sum* β $(0{::}nat)$ x **by** *simp*
finally show $(\bigotimes_{?D} i{::}nat \in \{..0{::}nat\}.\ (\beta\hat{\ }i)\ x) =$ *fin-sum* β $(0{::}nat)$ x **by**
simp
}
next
case *Suc*
{
fix n :: *nat*

assume *hypo*: ($\bigotimes_{?D} i::nat \in \{..n\}$. $(\beta\,\hat{}\,i)\ x$) = *fin-sum* β *n x*
have ($\bigotimes_{?D} i::nat \in \{..Suc\ n\}$. $(\beta\,\hat{}\,i)\ x$)
= *mult ?D* (($\beta\,\hat{}$(*Suc n*)) *x*) ($\bigotimes_{?D} i::nat \in \{..n\}$. $(\beta\,\hat{}\,i)\ x$)
using *comm-monoid.finprod-Suc* [*of ?D*]
using *diff-group-functor-preserves* [*OF diff-group-add*]
unfolding *diff-group-def*
unfolding *comm-group-def*
unfolding *diff-group-functor-def* **by** *simp*
also have ... = *monoid.mult ?D* (($\beta\,\hat{}$(*Suc n*)) *x*) (*fin-sum* β *n x*)
unfolding *hypo* **..**
also have ... = (($\beta\,\hat{}$(*Suc n*)) *x*) + (*fin-sum* β *n x*)
unfolding *diff-group-functor-def*
unfolding *monoid.select-convs* (*1*) **..**
also have ... = (($\beta\,\hat{}$(*Suc n*)) + (*fin-sum* β (*n*))) *x*
unfolding *plus-fun-def* [*of* ((β::'a ⇒ 'a) ^(*Suc n*)) (*fin-sum* β (*n*))] **..**
also have ... = (*fin-sum* β (*Suc n*) *x*) **by** *simp*
finally show ($\bigotimes_{?D} i::nat \in \{..Suc\ (n::nat)\}$. $(\beta\,\hat{}\,i)$ (*x*::'a))
= *fin-sum* β (*Suc n*) *x* **.**
}
qed
finally show *?thesis* **by** *simp*
qed
qed

As well as the equivalence between both definitions of the power series, also the definitions of the bounds are equivalent.

lemma *reduction-class-ext-preserves-bound-psi*:
assumes *r-c-e*:
reduction-class-ext (*f*::'a::*diff-group-add-pert-hom-bound-exist*
=> 'b::*diff-group-add*) *g*
shows *local-nilpotent-alpha.bound-psi*
(*diff-group-functor* (λ*x*::'a::*diff-group-add-pert-hom-bound-exist*. − *x*) *op* − *0 op* + *diff*)
(*ring-hom-compl* (*diff-group-functor* (λ*x*::'a. − *x*) *op* − *0 op* + *diff*))
hom-oper pert
= (*local-bound* β)
(**is** *local-nilpotent-alpha.bound-psi ?D ?R* (λ*x*. *h x*) (λ*x*. δ *x*) = (*local-bound* β))
proof −
have *diff-group-add-pert-hom-bound-exist*:
diff-group-add-pert-hom-bound-exist
op − (λ*x*::'a::*diff-group-add-pert-hom-bound-exist*. − *x*)
0 op + *diff pert hom-oper*
using *r-c-e*
unfolding *reduction-class-ext-def* [*of f g*] **by** *fast*
have *diff-group-add*: *diff-group-add op* − (λ*x*::'a. − *x*) *0 op* + *diff*
using *diff-group-add-pert-hom-bound-exist-impl-diff-group-add*
[*OF diff-group-add-pert-hom-bound-exist*] **.**
have *delta-in-R*: *pert* ∈ *carrier ?R* **and** *h-in-R*: *hom-oper* ∈ *carrier ?R*
using *r-c-e*

unfolding *reduction-class-ext-def*
unfolding *ring-hom-compl-def*
unfolding *diff-group-add-pert-hom-bound-exist-def*
unfolding *diff-group-add-pert-hom-def*
unfolding *diff-group-add-pert-hom-axioms-def*
unfolding *diff-group-add-pert-def*
unfolding *diff-group-add-pert-axioms-def*
unfolding *hom-completion-def*
unfolding *hom-def*
unfolding *Pi-def*
unfolding *completion-fun2-def*
unfolding *completion-def*
unfolding *diff-group-functor-def* **by** *auto*
have *hdelta-in-R*: $(\lambda x.\ h\ x) \circ (\lambda x.\ \delta\ x) \in carrier\ ?R$
using *delta-in-R h-in-R*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
using *monoid.m-closed* [*of* $?R\ (\lambda x.\ h\ x)\ (\lambda x.\ \delta\ x)$]
unfolding *Ring.ring-def ring-hom-compl-def* **by** *simp*
have *local-nilpotent-alpha.bound-psi ?D ?R* $(\lambda x.\ h\ x)\ (\lambda x.\ \delta\ x)$
$= (\lambda x::'a.\ LEAST\ n::nat.\ (alpha\text{-}beta.\beta\ ?R\ (\lambda x.\ h\ x)\ (\lambda x.\ \delta\ x)\ (\hat{\ })_{?R}\ n)\ x =$
one ?D)
using *local-nilpotent-alpha.bound-psi-def*
[*OF reduction-class-ext-preserves-local-nilpotent-alpha*
[*OF r-c-e*]] **by** *simp*
also have ... $= (\lambda x::'a.\ LEAST\ n::nat.\ (\beta\ (\hat{\ })_{?R}\ n)\ x = one\ ?D)$
proof −
have *β-equiv*: *alpha-beta.β ?R* $(\lambda x.\ h\ x)\ (\lambda x.\ \delta\ x) = \beta$
proof −
have *alpha-beta.β ?R* $(\lambda x.\ h\ x)\ (\lambda x.\ \delta\ x)$
$= \ominus_{?R}\ mult\ ?R\ (\lambda x.\ h\ x)\ (\lambda x.\ \delta\ x)$
unfolding *alpha-beta.beta-def*
[*OF reduction-class-ext-preserves-alpha-beta*
[*OF r-c-e*]] **..**
also have $\ominus_{?R}$ *monoid.mult ?R* $(\lambda x.\ h\ x)\ (\lambda x.\ \delta\ x)$
$= -\ ((\lambda x.\ h\ x) \circ (\lambda x.\ \delta\ x))$
using *hdelta-in-R*
using *minus-ring-hom-completion-equal-uminus-fun*
[*of* $(\lambda x.\ h\ x) \circ (\lambda x.\ \delta\ x)$, *OF diff-group-add*]
unfolding *ring-hom-compl-def* **by** *simp*
finally show *?thesis* **unfolding** *β-def* **.**
qed
then show *?thesis* **by** *simp*
qed
also have ... $= (\lambda x::'a.\ LEAST\ n::nat.\ (\beta\ (\hat{\ })_{?R}\ n)\ x = (0::'a))$
unfolding *diff-group-functor-def*
unfolding *monoid.select-convs* **..**
also have ... $= (\lambda x::'a.\ LEAST\ n::nat.\ (\beta \hat{\ } n)\ x = (0::'a))$
using *ring-nat-pow-equiv-funpow*
[*OF diff-group-add β-in-hom-completion*]

using *diff-group-add-pert-hom-bound-exist*
unfolding *diff-group-add-pert-hom-bound-exist-def* **by** *simp*
also have *bound-equiv*: ... = (*local-bound* β)
unfolding *expand-fun-eq*
unfolding *local-bounded-func-impl-local-bound-is-Least*
[*OF reduction-class-ext-preserves-beta-bound*
[*OF r-c-e*]] **by** *fast*
finally show *?thesis* **.**
qed

From the equivalence betwen the power series and the equality of the bounds, it follows the equivalence between the old and the new definition of *D-R-C-f-g-h-*δ*-*α*-bound-phi*.Ψ

lemma *reduction-class-ext-preserves-*Ψ:
assumes *r-c-e*:
reduction-class-ext (*f*::*'a*::*diff-group-add-pert-hom-bound-exist*
=> *'b*::*diff-group-add*) *g*
shows *local-nilpotent-alpha*.Ψ
(*diff-group-functor* (λx::*'a*::*diff-group-add-pert-hom-bound-exist*. − *x*) *op* − *0 op*
+ *diff*)
(*ring-hom-compl* (*diff-group-functor* (λx::*'a*. − *x*) *op* − *0 op* + *diff*))
hom-oper pert
= Ψ
(**is** *local-nilpotent-alpha*.Ψ *?D ?R* (λx. *h x*) (λx. δ *x*) = Ψ)
proof −
have *diff-group-add-pert-hom-bound-exist*:
diff-group-add-pert-hom-bound-exist
op − (λx::*'a*::*diff-group-add-pert-hom-bound-exist*. − *x*)
0 op + *diff pert hom-oper*
using *r-c-e*
unfolding *reduction-class-ext-def* [*of f g*] **by** *fast*
have *diff-group-add*: *diff-group-add op* − (λx::*'a*. − *x*) *0 op* + *diff*
using *diff-group-add-pert-hom-bound-exist-impl-diff-group-add*
[*OF diff-group-add-pert-hom-bound-exist*] **.**
have *delta-in-R*: *pert* ∈ *carrier ?R* **and** *h-in-R*: *hom-oper* ∈ *carrier ?R*
using *r-c-e*
unfolding *reduction-class-ext-def*
unfolding *ring-hom-compl-def*
unfolding *diff-group-add-pert-hom-bound-exist-def*
unfolding *diff-group-add-pert-hom-def*
unfolding *diff-group-add-pert-hom-axioms-def*
unfolding *diff-group-add-pert-def*
unfolding *diff-group-add-pert-axioms-def*
unfolding *hom-completion-def*
unfolding *hom-def*
unfolding *Pi-def*
unfolding *completion-fun2-def*
unfolding *completion-def*
unfolding *diff-group-functor-def* **by** *auto*
have *hdelta-in-R*: (λx. *h x*) ∘ (λx. δ *x*) ∈ *carrier ?R*

using *delta-in-R h-in-R*
using *diff-group-add.hom-completion-ring* [*OF diff-group-add*]
using *monoid.m-closed* [*of ?R* ($\lambda x.\ h\ x$) ($\lambda x.\ \delta\ x$)]
unfolding *ring-hom-compl-def Ring.ring-def* **by** *simp*
have *local-nilpotent-alpha*.Ψ *?D ?R* ($\lambda x.\ h\ x$) ($\lambda x.\ \delta\ x$)
= *local-nilpotent-term.power-series ?D ?R* (*alpha-beta*.β *?R* ($\lambda x.\ h\ x$) ($\lambda x.\ \delta$
x))
(*local-nilpotent-alpha.bound-psi ?D ?R* ($\lambda x.\ h\ x$) ($\lambda x.\ \delta\ x$))
unfolding *local-nilpotent-alpha.psi-def*
[*OF reduction-class-ext-preserves-local-nilpotent-alpha*
[*OF r-c-e*]] **..**
also have ...
= *local-nilpotent-term.power-series ?D ?R* (*alpha-beta*.β *?R*
($\lambda x.\ h\ x$) ($\lambda x.\ \delta\ x$)) (*local-bound* β)
using *reduction-class-ext-preserves-bound-psi* [*OF r-c-e*] **by** *simp*
also have ... = *local-nilpotent-term.power-series ?D ?R* β (*local-bound* β)
proof −
have β*-equiv*: *alpha-beta*.β *?R* ($\lambda x.\ h\ x$) ($\lambda x.\ \delta\ x$) = β
proof −
have *alpha-beta*.β *?R* ($\lambda x.\ h\ x$) ($\lambda x.\ \delta\ x$)
= $\ominus_{?R}$ *monoid.mult ?R* ($\lambda x.\ h\ x$) ($\lambda x.\ \delta\ x$)
using *alpha-beta.beta-def*
[*OF reduction-class-ext-preserves-alpha-beta*
[*OF r-c-e*]] **.**
also have $\ominus_{?R}$ *monoid.mult ?R* ($\lambda x.\ h\ x$) ($\lambda x.\ \delta\ x$) = β
using *hdelta-in-R*
using *minus-ring-hom-completion-equal-uminus-fun*
[*of* ($\lambda x.\ h\ x$) $\circ$ ($\lambda x.\ \delta\ x$), *OF diff-group-add*]
unfolding *ring-hom-compl-def* **by** (*fold* β*-def*, *simp*)
finally show *?thesis* **.**
qed
then show *?thesis* **by** *simp*
qed
also have ... = ($\lambda x.$ *fin-sum* β (*local-bound* β x) x)
using *reduction-class-ext-preserves-power-series-*β [*OF r-c-e*] **.**
also have ... = Ψ **unfolding** Ψ*-def* β*-def* **..**
finally show *?thesis* **.**
qed

Now, as a corollary, we prove the equivalence between the previous definition of the output g of the BPL, and the one in this new approach

corollary *reduction-class-ext-preserves-output-g*:
assumes *r-c-e*:
reduction-class-ext (*f*::'*a*::*diff-group-add-pert-hom-bound-exist*
=> '*b*::*diff-group-add*) *g*
shows *local-nilpotent-alpha*.Ψ
(*diff-group-functor* (λx::'*a*::*diff-group-add-pert-hom-bound-exist*. − *x*) *op* − *0 op*
+ *diff*)
(*ring-hom-compl* (*diff-group-functor* (λx::'*a*. − *x*) *op* − *0 op* + *diff*))

hom-oper pert ∘ *g*
= Ψ ∘ *g*
unfolding *reduction-class-ext-preserves-Ψ* [*OF r-c-e*] **..**

It also follows the equality of the previous definition of $d\ C'$ and the new definition, $dC'\ ?f\ ?g$ = *diff-group-add-class.diff* + (*?f* ∘ *diff-group-add-pert-class.pert* ∘ Ψ ∘ *?g*)

corollary *reduction-class-ext-preserves-output-dC*:
assumes *r-c-e*:
reduction-class-ext (*f*::′*a*::*diff-group-add-pert-hom-bound-exist*
=> ′*b*::*diff-group-add*) *g*
shows (λ*x*::′*b*.
if x ∈ *carrier* (*diff-group-functor* (λ*x*::′*b*::*diff-group-add*. − *x*) *op* − *0 op* + *diff*)

then mult (*diff-group-functor* (λ*x*::′*b*::*diff-group-add*. − *x*) *op* − *0 op* + *diff*)
(*diff-group.diff* (*diff-group-functor* (λ*x*::′*b*::*diff-group-add*. − *x*) *op* − *0 op* + *diff*)
x)
((*f* ∘ *pert* ∘
(*local-nilpotent-alpha*.Ψ
(*diff-group-functor* (λ*x*::′*a*::*diff-group-add-pert-hom-bound-exist*. − *x*) *op* − *0 op*
+ *diff*)
(*ring-hom-compl* (*diff-group-functor* (λ*x*::′*a*. − *x*) *op* − *0 op* + *diff*))
hom-oper pert) ∘ *g*) *x*) *else one* (*diff-group-functor* (λ*x*::′*b*. − *x*) *op* − *0 op* +
diff))
= *dC*′ *f g*
unfolding *reduction-class-ext-preserves-Ψ*
[*OF r-c-e*]
unfolding *dC*′*-def* [*of f g*]
unfolding *diff-group-functor-def*
unfolding *plus-fun-def* **by** *simp*

Now, from he previous equivalences, we are ready to give the proof of the *reduction D*′ ⦇*carrier* = *carrier C*, *mult* = *op* $\otimes_C$, *one* = $\mathbf{1}_C$, *diff-group.diff* = λ*x*. *if x* ∈ *carrier C then diff-group.diff C x* $\otimes_C$ (*f* ∘ δ ∘ *D-R-C-f-g-h-δ-α-bound-phi*.Ψ ∘ *g*) *x else* $\mathbf{1}_C$⦈ (*f* ∘ *D-R-C-f-g-h-δ-α-bound-phi*.Φ) (*D-R-C-f-g-h-δ-α-bound-phi*.Ψ ∘ *g*) *D-R-h-C-f-g-δ-α-bound-phi.h*′ with the new introduced definitions in terms of classes:

lemma assumes *reduction-class-ext-f-g*:
reduction-class-ext (*f*::′*a*::*diff-group-add-pert-hom-bound-exist*
=> ′*b*::*diff-group-add*) *g*
shows *reduction*
(*lemma-2-2-15.D*′
(*diff-group-functor* (λ*x*::′*a*::*diff-group-add-pert-hom-bound-exist*. − *x*) *op* − *0 op*
+ *diff*)
(*ring-hom-compl*
(*diff-group-functor* (λ*x*::′*a*. − *x*) *op* − *0 op* + *diff*)) *pert*)
(⦇*carrier* = *carrier* (*diff-group-functor* (λ*x*::′*b*::*diff-group-add*. − *x*) *op* − *0 op* +
diff),

mult = *mult* (*diff-group-functor* (λ*x*::′*b*. − *x*) *op* − *0* *op* + *diff*),
one = *one* (*diff-group-functor* (λ*x*::′*b*. − *x*) *op* − *0* *op* + *diff*),
diff-group.diff = *dC*′ *f* *g*⦈
(*f*′ *f*)
(*g*′ *g*)
(*h*′)
using *BPL.BPL*
[*OF* *reduction-class-ext-preserves-BPL*
[*OF* *reduction-class-ext-f-g*]]
unfolding *reduction-class-ext-preserves-output-f* [*OF* *reduction-class-ext-f-g*]
unfolding *reduction-class-ext-preserves-output-g* [*OF* *reduction-class-ext-f-g*]
unfolding *reduction-class-ext-preserves-output-h* [*OF* *reduction-class-ext-f-g*]
unfolding *reduction-class-ext-preserves-output-dC* [*OF* *reduction-class-ext-f-g*]
unfolding *f*′*-def* [*of* *f*]
unfolding *g*′*-def* [*of* *g*]
unfolding *h*′*-def*
unfolding *dC*′*-def* [*of* *f* *g*] **.**

end

20 Pretty integer literals for code generation

theory *Code-Integer*
imports *ATP-Linkup*
begin

HOL numeral expressions are mapped to integer literals in target languages, using predefined target language operations for abstract integer operations.

code-type *int*
(*SML IntInf.int*)
(*OCaml Big*′*-int.big*′*-int*)
(*Haskell Integer*)

code-instance *int* :: *eq*
(*Haskell* −)

setup ⟪
fold (*Numeral.add-code* @{*const-name number-int-inst.number-of-int*}
true true) [*SML*, *OCaml*, *Haskell*]
⟫

code-const *Int.Pls* **and** *Int.Min* **and** *Int.Bit0* **and** *Int.Bit1*
(*SML raise*/ *Fail*/ *Pls*
and *raise*/ *Fail*/ *Min*
and !((-);/ *raise*/ *Fail*/ *Bit0*)
and !((-);/ *raise*/ *Fail*/ *Bit1*))
(*OCaml failwith*/ *Pls*
and *failwith*/ *Min*
and !((-);/ *failwith*/ *Bit0*)

and !((-);/ *failwith*/ *Bit1*))
(*Haskell error*/ *Pls*
and *error*/ *Min*
and *error*/ *Bit0*
and *error*/ *Bit1*)

code-const *Int.pred*
(*SML IntInf*.− ((-), *1*))
(*OCaml Big'-int.pred'-big'-int*)
(*Haskell* !(-/ −/ *1*))

code-const *Int.succ*
(*SML IntInf*.+ ((-), *1*))
(*OCaml Big'-int.succ'-big'-int*)
(*Haskell* !(-/ +/ *1*))

code-const *op* + :: *int* ⇒ *int* ⇒ *int*
(*SML IntInf*.+ ((-), (-)))
(*OCaml Big'-int.add'-big'-int*)
(*Haskell* **infixl** *6* +)

code-const *uminus* :: *int* ⇒ *int*
(*SML IntInf*.~)
(*OCaml Big'-int.minus'-big'-int*)
(*Haskell negate*)

code-const *op* − :: *int* ⇒ *int* ⇒ *int*
(*SML IntInf*.− ((-), (-)))
(*OCaml Big'-int.sub'-big'-int*)
(*Haskell* **infixl** *6* −)

code-const *op* ∗ :: *int* ⇒ *int* ⇒ *int*
(*SML IntInf*.∗ ((-), (-)))
(*OCaml Big'-int.mult'-big'-int*)
(*Haskell* **infixl** *7* ∗)

code-const *op* = :: *int* ⇒ *int* ⇒ *bool*
(*SML* !((- : *IntInf.int*) = -))
(*OCaml Big'-int.eq'-big'-int*)
(*Haskell* **infixl** *4* ==)

code-const *op* ≤ :: *int* ⇒ *int* ⇒ *bool*
(*SML IntInf*.<= ((-), (-)))
(*OCaml Big'-int.le'-big'-int*)
(*Haskell* **infix** *4* <=)

code-const *op* < :: *int* ⇒ *int* ⇒ *bool*
(*SML IntInf*.< ((-), (-)))
(*OCaml Big'-int.lt'-big'-int*)

(*Haskell* **infix** *4* <)

code-reserved *SML IntInf*
code-reserved *OCaml Big-int*

end

21 Type of indices

theory *Code-Index*
imports *ATP-Linkup*
begin

Indices are isomorphic to HOL *nat* but mapped to target-language builtin integers

21.1 Datatype of indices

typedef *index* = *UNIV* :: *nat set*
  morphisms *nat-of-index index-of-nat* **by** *rule*

lemma *index-of-nat-nat-of-index* [*simp*]:
  index-of-nat (*nat-of-index k*) = *k*
  by (*rule nat-of-index-inverse*)

lemma *nat-of-index-index-of-nat* [*simp*]:
  nat-of-index (*index-of-nat n*) = *n*
  by (*rule index-of-nat-inverse*)
    (*unfold index-def*, *rule UNIV-I*)

lemma *index*:
  ($\bigwedge$*n*::*index*. *PROP P n*) ≡ ($\bigwedge$*n*::*nat*. *PROP P* (*index-of-nat n*))
proof
  fix *n* :: *nat*
  assume $\bigwedge$*n*::*index*. *PROP P n*
  then show *PROP P* (*index-of-nat n*) **.**
next
  fix *n* :: *index*
  assume $\bigwedge$*n*::*nat*. *PROP P* (*index-of-nat n*)
  then have *PROP P* (*index-of-nat* (*nat-of-index n*)) **.**
  then show *PROP P n* **by** *simp*
qed

lemma *index-case*:
  assumes $\bigwedge$*n*. *k* = *index-of-nat n* $\Longrightarrow$ *P*
  shows *P*
  by (*rule assms* [*of nat-of-index k*]) *simp*

lemma *index-induct-raw*:

```
  assumes ⋀n. P (index-of-nat n)
  shows P k
proof −
  from assms have P (index-of-nat (nat-of-index k)) .
  then show ?thesis by simp
qed

lemma nat-of-index-inject [simp]:
  nat-of-index k = nat-of-index l ⟷ k = l
  by (rule nat-of-index-inject)

lemma index-of-nat-inject [simp]:
  index-of-nat n = index-of-nat m ⟷ n = m
  by (auto intro!: index-of-nat-inject simp add: index-def)

instantiation index :: zero
begin

definition [simp, code func del]:
  0 = index-of-nat 0

instance ..

end

definition [simp]:
  Suc-index k = index-of-nat (Suc (nat-of-index k))

lemma index-induct: P 0 ⟹ (⋀k. P k ⟹ P (Suc-index k)) ⟹ P k
proof −
  assume P 0 then have init: P (index-of-nat 0) by simp
  assume ⋀k. P k ⟹ P (Suc-index k)
    then have ⋀n. P (index-of-nat n) ⟹ P (Suc-index (index-of-nat (n))) .
    then have step: ⋀n. P (index-of-nat n) ⟹ P (index-of-nat (Suc n)) by simp
  from init step have P (index-of-nat (nat-of-index k))
    by (induct nat-of-index k) simp-all
  then show P k by simp
qed

lemma Suc-not-Zero-index: Suc-index k ≠ 0
  by simp

lemma Zero-not-Suc-index: 0 ≠ Suc-index k
  by simp

lemma Suc-Suc-index-eq: Suc-index k = Suc-index l ⟷ k = l
  by simp

rep-datatype index
```

distinct *Suc-not-Zero-index Zero-not-Suc-index*
inject *Suc-Suc-index-eq*
induction *index-induct*

lemmas [*code func del*] = *index.recs index.cases*

declare *index-case* [*case-names nat*, *cases type*: *index*]
declare *index-induct* [*case-names nat*, *induct type*: *index*]

lemma [*code func*]:
index-size = *nat-of-index*
proof (*rule ext*)
fix *k*
have *index-size k* = *nat-size* (*nat-of-index k*)
by (*induct k rule*: *index.induct*) (*simp-all del*: *zero-index-def Suc-index-def*, *simp-all*)
also have *nat-size* (*nat-of-index k*) = *nat-of-index k* **by** (*induct nat-of-index k*) *simp-all*
finally show *index-size k* = *nat-of-index k* **.**
qed

lemma [*code func*]:
size = *nat-of-index*
proof (*rule ext*)
fix *k*
show *size k* = *nat-of-index k*
by (*induct k*) (*simp-all del*: *zero-index-def Suc-index-def*, *simp-all*)
qed

lemma [*code func*]:
k = *l* ⟷ *nat-of-index k* = *nat-of-index l*
by (*cases k*, *cases l*) *simp*

21.2 Indices as datatype of ints

instantiation *index* :: *number*
begin

definition
number-of = *index-of-nat o nat*

instance ..

end

lemma *nat-of-index-number* [*simp*]:
nat-of-index (*number-of k*) = *number-of k*
by (*simp add*: *number-of-index-def nat-number-of-def number-of-is-id*)

code-datatype *number-of* :: *int* ⇒ *index*

21.3 Basic arithmetic

instantiation *index* :: {*minus*, *ordered-semidom*, *Divides.div*, *linorder*}
begin

lemma *zero-index-code* [*code inline*, *code func*]:
  (*0*::*index*) = *Numeral0*
  by (*simp add*: *number-of-index-def Pls-def*)
lemma [*code post*]: *Numeral0* = (*0*::*index*)
  using *zero-index-code* **..**

definition [*simp*, *code func del*]:
  (*1*::*index*) = *index-of-nat 1*

lemma *one-index-code* [*code inline*, *code func*]:
  (*1*::*index*) = *Numeral1*
  by (*simp add*: *number-of-index-def Pls-def Bit1-def*)
lemma [*code post*]: *Numeral1* = (*1*::*index*)
  using *one-index-code* **..**

definition [*simp*, *code func del*]:
  n + *m* = *index-of-nat* (*nat-of-index n* + *nat-of-index m*)

lemma *plus-index-code* [*code func*]:
  index-of-nat n + *index-of-nat m* = *index-of-nat* (*n* + *m*)
  by *simp*

definition [*simp*, *code func del*]:
  n − *m* = *index-of-nat* (*nat-of-index n* − *nat-of-index m*)

definition [*simp*, *code func del*]:
  n ∗ *m* = *index-of-nat* (*nat-of-index n* ∗ *nat-of-index m*)

lemma *times-index-code* [*code func*]:
  index-of-nat n ∗ *index-of-nat m* = *index-of-nat* (*n* ∗ *m*)
  by *simp*

definition [*simp*, *code func del*]:
  n div m = *index-of-nat* (*nat-of-index n div nat-of-index m*)

definition [*simp*, *code func del*]:
  n mod m = *index-of-nat* (*nat-of-index n mod nat-of-index m*)

lemma *div-index-code* [*code func*]:
  index-of-nat n div index-of-nat m = *index-of-nat* (*n div m*)
  by *simp*

lemma *mod-index-code* [*code func*]:
index-of-nat n mod index-of-nat m = *index-of-nat* (*n mod m*)
by *simp*

definition [*simp*, *code func del*]:
n ≤ *m* ⟷ *nat-of-index n* ≤ *nat-of-index m*

definition [*simp*, *code func del*]:
n < *m* ⟷ *nat-of-index n* < *nat-of-index m*

lemma *less-eq-index-code* [*code func*]:
index-of-nat n ≤ *index-of-nat m* ⟷ *n* ≤ *m*
by *simp*

lemma *less-index-code* [*code func*]:
index-of-nat n < *index-of-nat m* ⟷ *n* < *m*
by *simp*

instance by *default* (*auto simp add*: *left-distrib index*)

end

lemma *Suc-index-minus-one*: *Suc-index n* − *1* = *n* **by** *simp*

lemma *index-of-nat-code* [*code*]:
index-of-nat = *of-nat*
proof
fix *n* :: *nat*
have *of-nat n* = *index-of-nat n*
by (*induct n*) *simp-all*
then show *index-of-nat n* = *of-nat n*
by (*rule sym*)
qed

lemma *index-not-eq-zero*: *i* ≠ *index-of-nat 0* ⟷ *i* ≥ *1*
by (*cases i*) *auto*

definition
nat-of-index-aux :: *index* ⇒ *nat* ⇒ *nat*
where
nat-of-index-aux i n = *nat-of-index i* + *n*

lemma *nat-of-index-aux-code* [*code*]:
nat-of-index-aux i n = (*if i* = *0 then n else nat-of-index-aux* (*i* − *1*) (*Suc n*))
by (*auto simp add*: *nat-of-index-aux-def index-not-eq-zero*)

lemma *nat-of-index-code* [*code*]:
nat-of-index i = *nat-of-index-aux i 0*
by (*simp add*: *nat-of-index-aux-def*)

21.4 ML interface

ML ⟪
structure Index =
struct

fun mk k = HOLogic.mk-number @{typ index} k;

end;
⟫

21.5 Specialized *op −*, *op div* and *op mod* operations

definition
  minus-index-aux :: *index* ⇒ *index* ⇒ *index*
where
  [*code func del*]: *minus-index-aux* = *op* −

lemma [*code func*]: *op* − = *minus-index-aux*
  using *minus-index-aux-def* **..**

definition
  div-mod-index :: *index* ⇒ *index* ⇒ *index* × *index*
where
  [*code func del*]: *div-mod-index n m* = (*n div m*, *n mod m*)

lemma [*code func*]:
  div-mod-index n m = (*if m* = *0 then* (*0*, *n*) *else* (*n div m*, *n mod m*))
  unfolding *div-mod-index-def* **by** *auto*

lemma [*code func*]:
  n div m = *fst* (*div-mod-index n m*)
  unfolding *div-mod-index-def* **by** *simp*

lemma [*code func*]:
  n mod m = *snd* (*div-mod-index n m*)
  unfolding *div-mod-index-def* **by** *simp*

21.6 Code serialization

Implementation of indices by bounded integers

code-type *index*
  (*SML int*)
  (*OCaml int*)
  (*Haskell Int*)

code-instance *index* :: *eq*
  (*Haskell* −)

setup ⟪
  fold (*Numeral.add-code* @{*const-name number-index-inst.number-of-index*}
    false false) [*SML*, *OCaml*, *Haskell*]
⟫

code-reserved *SML Int int*
code-reserved *OCaml Pervasives int*

code-const *op* + :: *index* ⇒ *index* ⇒ *index*
  (*SML Int*.+/ ((-),/ (-)))
  (*OCaml Pervasives*.(+))
  (*Haskell* **infixl** *6* +)

code-const *minus-index-aux* :: *index* ⇒ *index* ⇒ *index*
  (*SML Int.max*/ (-/ −/ -,/ *0* : *int*))
  (*OCaml Pervasives.max*/ (-/ −/ -)/ (*0* : *int*))
  (*Haskell max*/ (-/ −/ -)/ (*0* :: *Int*))

code-const *op* ∗ :: *index* ⇒ *index* ⇒ *index*
  (*SML Int*.∗/ ((-),/ (-)))
  (*OCaml Pervasives*.(∗))
  (*Haskell* **infixl** *7* ∗)

code-const *div-mod-index*
  (*SML* (*fn n* => *fn m* =>/ (*n div m*, *n mod m*)))
  (*OCaml* (*fun n* −> *fun m* −>/ (*n* '/ *m*, *n mod m*)))
  (*Haskell divMod*)

code-const *op* = :: *index* ⇒ *index* ⇒ *bool*
  (*SML* !((- : *Int.int*) = -))
  (*OCaml* !((- : *int*) = -))
  (*Haskell* **infixl** *4* ==)

code-const *op* ≤ :: *index* ⇒ *index* ⇒ *bool*
  (*SML Int*.<=/ ((-),/ (-)))
  (*OCaml* !((- : *int*) <= -))
  (*Haskell* **infix** *4* <=)

code-const *op* < :: *index* ⇒ *index* ⇒ *bool*
  (*SML Int*.</ ((-),/ (-)))
  (*OCaml* !((- : *int*) < -))
  (*Haskell* **infix** *4* <)

end

22 Implementation of natural numbers by target-language integers

theory *Efficient-Nat*
imports *Code-Integer Code-Index*
begin

When generating code for functions on natural numbers, the canonical representation using *0* and *Suc* is unsuitable for computations involving large numbers. The efficiency of the generated code can be improved drastically by implementing natural numbers by target-language integers. To do this, just include this theory.

22.1 Basic arithmetic

Most standard arithmetic functions on natural numbers are implemented using their counterparts on the integers:

code-datatype *number-nat-inst.number-of-nat*

lemma *zero-nat-code* [*code*, *code unfold*]:
  0 = (*Numeral0* :: *nat*)
  by *simp*
lemmas [*code post*] = *zero-nat-code* [*symmetric*]

lemma *one-nat-code* [*code*, *code unfold*]:
  1 = (*Numeral1* :: *nat*)
  by *simp*
lemmas [*code post*] = *one-nat-code* [*symmetric*]

lemma *Suc-code* [*code*]:
  Suc n = *n* + *1*
  by *simp*

lemma *plus-nat-code* [*code*]:
  n + *m* = *nat* (*of-nat n* + *of-nat m*)
  by *simp*

lemma *minus-nat-code* [*code*]:
  n − *m* = *nat* (*of-nat n* − *of-nat m*)
  by *simp*

lemma *times-nat-code* [*code*]:
  n ∗ *m* = *nat* (*of-nat n* ∗ *of-nat m*)
  unfolding *of-nat-mult* [*symmetric*] **by** *simp*

Specialized *op div* and *op mod* operations.

definition

divmod-aux :: *nat* ⇒ *nat* ⇒ *nat* × *nat*
where
[*code func del*]: *divmod-aux* = *divmod*

lemma [*code func*]:
divmod n m = (*if m* = *0 then* (*0*, *n*) *else divmod-aux n m*)
unfolding *divmod-aux-def divmod-div-mod* **by** *simp*

lemma *divmod-aux-code* [*code*]:
divmod-aux n m = (*nat* (*of-nat n div of-nat m*), *nat* (*of-nat n mod of-nat m*))
unfolding *divmod-aux-def divmod-div-mod zdiv-int* [*symmetric*] *zmod-int* [*symmetric*]
by *simp*

lemma *eq-nat-code* [*code*]:
$n = m \longleftrightarrow$ (*of-nat n* :: *int*) = *of-nat m*
by *simp*

lemma *less-eq-nat-code* [*code*]:
$n \leq m \longleftrightarrow$ (*of-nat n* :: *int*) ≤ *of-nat m*
by *simp*

lemma *less-nat-code* [*code*]:
$n < m \longleftrightarrow$ (*of-nat n* :: *int*) < *of-nat m*
by *simp*

22.2 Case analysis

Case analysis on natural numbers is rephrased using a conditional expression:

lemma [*code func*, *code unfold*]:
nat-case = (λ*f g n*. *if n* = *0 then f else g* (*n* − *1*))
by (*auto simp add*: *expand-fun-eq dest*!: *gr0-implies-Suc*)

22.3 Preprocessors

In contrast to *Suc n*, the term *n* + *1* is no longer a constructor term. Therefore, all occurrences of this term in a position where a pattern is expected (i.e. on the left-hand side of a recursion equation or in the arguments of an inductive relation in an introduction rule) must be eliminated. This can be accomplished by applying the following transformation rules:

lemma *Suc-if-eq*: (⋀*n*. *f* (*Suc n*) = *h n*) ⟹ *f 0* = *g* ⟹
f n = (*if n* = *0 then g else h* (*n* − *1*))
by (*case-tac n*) *simp-all*

lemma *Suc-clause*: (⋀*n*. *P n* (*Suc n*)) ⟹ *n* ≠ *0* ⟹ *P* (*n* − *1*) *n*
by (*case-tac n*) *simp-all*

The rules above are built into a preprocessor that is plugged into the code generator. Since the preprocessor for introduction rules does not know anything about modes, some of the modes that worked for the canonical representation of natural numbers may no longer work.

22.4 Target language setup

For ML, we map *nat* to target language integers, where we assert that values are always non-negative.

```
code-type nat
  (SML int)
  (OCaml Big'-int.big'-int)

types-code
  nat (int)
attach (term-of) ⟪
val term-of-nat = HOLogic.mk-number HOLogic.natT;
⟫
attach (test) ⟪
fun gen-nat i =
  let val n = random-range 0 i
  in (n, fn () => term-of-nat n) end;
⟫
```

For Haskell we define our own *nat* type. The reason is that we have to distinguish type class instances for *nat* and *int*.

```
code-include Haskell Nat ⟪
newtype Nat = Nat Integer deriving (Show, Eq);

instance Num Nat where {
  fromInteger k = Nat (if k >= 0 then k else 0);
  Nat n + Nat m = Nat (n + m);
  Nat n − Nat m = fromInteger (n − m);
  Nat n ∗ Nat m = Nat (n ∗ m);
  abs n = n;
  signum - = 1;
  negate n = error negate Nat;
};

instance Ord Nat where {
  Nat n <= Nat m = n <= m;
  Nat n < Nat m = n < m;
};

instance Real Nat where {
  toRational (Nat n) = toRational n;
};

instance Enum Nat where {
```

toEnum k = *fromInteger* (*toEnum k*);
fromEnum (*Nat n*) = *fromEnum n*;
};

instance Integral Nat where {
toInteger (*Nat n*) = *n*;
divMod n m = *quotRem n m*;
quotRem (*Nat n*) (*Nat m*) = (*Nat k*, *Nat l*) *where* (*k*, *l*) = *quotRem n m*;
};
⟫

code-reserved *Haskell Nat*

code-type *nat*
(*Haskell Nat*)

code-instance *nat* :: *eq*
(*Haskell* −)

Natural numerals.

lemma [*code inline*, *symmetric*, *code post*]:
nat (*number-of i*) = *number-nat-inst.number-of-nat i*
— this interacts as desired with *number-of ?v* = *nat* (*number-of ?v*)
by (*simp add*: *number-nat-inst.number-of-nat*)

setup ⟪
fold (*Numeral.add-code* @{*const-name number-nat-inst.number-of-nat*}
true false) [*SML*, *OCaml*, *Haskell*]
⟫

Since natural numbers are implemented using integers in ML, the coercion function *of-nat* of type *nat* ⇒ *int* is simply implemented by the identity function. For the *nat* function for converting an integer to a natural number, we give a specific implementation using an ML function that returns its input value, provided that it is non-negative, and otherwise returns *0*.

definition
int :: *nat* ⇒ *int*
where
[*code func del*]: *int* = *of-nat*

lemma *int-code′* [*code func*]:
int (*number-of l*) = (*if neg* (*number-of l* :: *int*) *then 0 else number-of l*)
unfolding *int-nat-number-of* [*folded int-def*] **..**

lemma *nat-code′* [*code func*]:
nat (*number-of l*) = (*if neg* (*number-of l* :: *int*) *then 0 else number-of l*)
by *auto*

lemma *of-nat-int* [*code unfold*]:

of-nat = *int* **by** (*simp add*: *int-def*)
declare *of-nat-int* [*symmetric*, *code post*]

code-const *int*
(*SML* -)
(*OCaml* -)

consts-code
int ((-))
nat (⟨**module**⟩*nat*)
attach ⟪
fun nat i = *if i* < *0 then 0 else i*;
⟫

code-const *nat*
(*SML IntInf.max/* (*/0*,/ -))
(*OCaml Big'-int.max'-big'-int/ Big'-int.zero'-big'-int*)

For Haskell, things are slightly different again.

code-const *int* **and** *nat*
(*Haskell toInteger* **and** *fromInteger*)

Conversion from and to indices.

code-const *index-of-nat*
(*SML IntInf.toInt*)
(*OCaml Big'-int.int'-of'-big'-int*)
(*Haskell toEnum*)

code-const *nat-of-index*
(*SML IntInf.fromInt*)
(*OCaml Big'-int.big'-int'-of'-int*)
(*Haskell fromEnum*)

Using target language arithmetic operations whenever appropriate

code-const *op* + :: *nat* ⇒ *nat* ⇒ *nat*
(*SML IntInf.*+ ((-), (-)))
(*OCaml Big'-int.add'-big'-int*)
(*Haskell* **infixl** *6* +)

code-const *op* ∗ :: *nat* ⇒ *nat* ⇒ *nat*
(*SML IntInf.*∗ ((-), (-)))
(*OCaml Big'-int.mult'-big'-int*)
(*Haskell* **infixl** *7* ∗)

code-const *divmod-aux*
(*SML IntInf.divMod/* ((-),/ (-)))
(*OCaml Big'-int.quomod'-big'-int*)
(*Haskell divMod*)

code-const *op* = :: *nat* ⇒ *nat* ⇒ *bool*
 (*SML* !((- : *IntInf.int*) = -))
 (*OCaml Big′-int.eq′-big′-int*)
 (*Haskell* **infixl** *4* ==)

code-const *op* ≤ :: *nat* ⇒ *nat* ⇒ *bool*
 (*SML IntInf.*<= ((-), (-)))
 (*OCaml Big′-int.le′-big′-int*)
 (*Haskell* **infix** *4* <=)

code-const *op* < :: *nat* ⇒ *nat* ⇒ *bool*
 (*SML IntInf.*< ((-), (-)))
 (*OCaml Big′-int.lt′-big′-int*)
 (*Haskell* **infix** *4* <)

consts-code
 0 (*0*)
 Suc ((- +/ *1*))
 op + :: *nat* ⇒ *nat* ⇒ *nat* ((- +/ -))
 op ∗ :: *nat* ⇒ *nat* ⇒ *nat* ((- ∗/ -))
 op ≤ :: *nat* ⇒ *nat* ⇒ *bool* ((- <=/ -))
 op < :: *nat* ⇒ *nat* ⇒ *bool* ((- </ -))

Module names

code-modulename *SML*
 Nat Integer
 Divides Integer
 Efficient-Nat Integer

code-modulename *OCaml*
 Nat Integer
 Divides Integer
 Efficient-Nat Integer

code-modulename *Haskell*
 Nat Integer
 Divides Integer
 Efficient-Nat Integer

hide *const int*

end

theory *example-Z4Z2*
 imports
 BPL-classes-2008
 Efficient-Nat

begin

23 An example of the BPL: a reduction from Z^4 to Z^2

We fit a concrete example of the BPL into the code previously generated

The example is the following: we have a "big" differential group, $D = Z^4$, with componentwise addition, and a differential defined as $d_D x = (0, 2 * fst(x), 0, thrd(x))$; the homotopy operator is given by $hx = (0, 0, frthx, 0)$ and the perturbation is $\delta_D x = (0, fst(x) + thrd(x), 0, fst(x))$; the nilpotency condition in this example is globally satisfied for $n = 2$

This means that $\forall x. (\delta_D \circ h)^2(x) = 0$; we will later prove this in Isabelle.

The small differential group is defined as $C = Z^2$, with componentwise addition, and a differential defined as $d_C(x) = (0, fst(x) + fst(x))$

Then, $f(x) = (fst(x), snd(x))$, $g(x) = (fst(x), snd(x), 0, 0)$ and $h(x) = (0, 0, frth(x), 0)$

In order to apply the example in Isabelle, we first define a type representing the 4-tuples, which will be our representation of Z^4

The following definitions and concrete syntax have been mainly extracted from file *Product-Type.thy*, resembling the ones given there for pairs.

The generic product was not instantiable with a single parameter type (*int* in our case), since this gave place to a too restrictive instance. This is why we produced our own single parameterized product type for pairs and four tuples.

The data type has been defined just to allow us to generate code, which means that a very small number of facts are available about it

23.1 Type definition for Z^2

The following type definition represents tuples. We will use it to represent Z^2. There exists already a type representing products in HOL, but it cannot be *instantiated* with a single parameter type (in our case, Z), since the type obtained is too restrictive with respect to the type representation.

Therefore, we defined our own product type with only a single type parameter.

This type will be also used later to obtain a type representing Z^4.

datatype *'a SProd* = *SPair 'a 'a*

Some basic definitions over the previous type:

definition *fst-spair* :: *'a SProd* => *'a*
  where *fst-spair p* = (*THE a. EX b. p* = *SPair a b*)

definition *snd-spair* :: *'a SProd* => *'a*
  where *snd-spair p* == (*THE b. EX a. p* = *SPair a b*)

We omit the previous definitions from the code generator, since they do not have an executable content.

lemmas [*code del*] = *fst-spair-def snd-spair-def*

23.2 Concrete syntax

Special syntax for the produced type:

translations [(*x*, *y*)] == *SPair x y*

instance *SProd* :: (*type*) *type* **..**

23.3 Lemmas and proof tool setup

lemma *SPair-eq* [*iff*]:
  ([(*a*, *b*)] = [(*a'*, *b'*)]) = (*a* = *a'* & *b* = *b'*)
  by *simp*

lemma *fst-spair-conv* [*simp*,*code*]:
  fst-spair [(*a*, *b*)] = *a*
  unfolding *fst-spair-def* **by** *blast*

lemma *snd-spair-conv* [*simp*,*code*]:
  snd-spair [(*a*, *b*)] = *b*
  unfolding *snd-spair-def* **by** *blast*

lemma *fst-spair-eqD*:
  fst-spair [(*x*, *y*)] = *a* ==> *x* = *a*
  by *simp*

lemma *snd-spair-eqD*:
  snd-spair [(*x*, *y*)] = *a* ==> *y* = *a*
  by *simp*

lemma *surjective-spair*:
  p = [(*fst-spair p*, *snd-spair p*)]
  — Do not add as rewrite rule: invalidates some proofs in IMP
  by (*cases p*) *simp*

lemma *split-SPair-all*:

```
  (!!x. PROP P x) == (!!a b. PROP P [(a, b)])
proof
  fix a b
  assume !!x. PROP P x
  then show PROP P [(a, b)] .
next
  fix x
  assume !!a b. PROP P [(a, b)]
  show PROP P x
    using surjective-spair [of x::'a SProd]
    using ‹PROP P [(fst-spair (x::'a SProd), snd-spair x)]›
    by simp
qed
```

We now prove that the introduced datatype is an instance of the type classes needed in the BPL

As far as *int* is not a type class, but a type constructor, we will use *Ring.ring* as a type class.

Therefore, we prove that our type constructor *SProd* with suitable operations is a valid instance of the type class (*ring*) *diff-group-add*

Being *int* a valid instance of *Ring.ring*, we can then, in the code generation phase, replace *Ring.ring* with the concrete structure *int*

```
instance SProd :: (eq) eq ..

lemma [code func]:
  [(x1::'a::eq, x2)] = [(y1, y2)] ⟷ x1 = y1 ∧ x2 = y2
  by auto

instantiation SProd :: ({eq,ring}) ab-semigroup-add
begin

definition SProd-plus-def:
  a + b = [( (fst-spair a + fst-spair b), (snd-spair a + snd-spair b))]

instance
by default (simp-all add: split-SPair-all SProd-plus-def)

end

instantiation SProd :: ({eq,ring}) comm-monoid-add
begin

definition SProd-zero-def: 0 = [(0, 0)]

instance by default (simp-all add: split-SPair-all SProd-plus-def SProd-zero-def)
```

end

instantiation *SProd* :: ({*eq*,*ring*}) *ab-group-add*
begin

definition *SProd-uminus-def*:
$- \, x = [(\, - \, fst\text{-}spair \, x, \, - \, snd\text{-}spair \, x)]$

definition *SProd-minus-def*:
$(x{::}'a \; SProd) - y = (x + (- \, y))$

instance
apply *default*
unfolding *split-SPair-all*
unfolding *SProd-zero-def*
unfolding *SProd-minus-def*
unfolding *SProd-uminus-def*
unfolding *SProd-plus-def* **by** *simp-all*

end

instantiation *SProd*:: ({*eq*,*ring*}) *diff-group-add*
begin

definition *SProd-diff-def*:
$diff \; x \equiv [(\; 0, \, fst\text{-}spair \; x + fst\text{-}spair \; x)]$

instance
apply *default*
unfolding *split-SPair-all*
unfolding *expand-fun-eq*
unfolding *o-apply*
unfolding *SProd-diff-def*
unfolding *SProd-plus-def*
unfolding *SProd-zero-def* **by** *simp-all*

end

23.4 Type definition for Z^4

datatype *'a Quad-type-const* = *Quad 'a SProd 'a SProd*

23.5 Definitions over the given type.

definition *fst-quad* :: *'a Quad-type-const* => *'a*
where *fst-quad p* == *THE a. EX b c e. p* = *Quad* (*SPair a b*) (*SPair c e*)

definition *snd-quad* :: *'a Quad-type-const* => *'a*
where *snd-quad p* == *THE b. EX a c e. p* = *Quad* (*SPair a b*) (*SPair c e*)

definition *thrd-quad* :: *'a Quad-type-const* => *'a*
where *thrd-quad p* == *THE c. EX a b e. p* = *Quad* (*SPair a b*) (*SPair c e*)

definition *frth-quad* :: *'a Quad-type-const* => *'a*
where *frth-quad p* == *THE e. EX a b c. p* = *Quad* (*SPair a b*) (*SPair c e*)

We delete the previous definitions from the code genrator setup, soince they do not have an executable meaning.

lemmas [*code del*] = *fst-quad-def snd-quad-def thrd-quad-def frth-quad-def*

23.6 Concrete syntax.

translations
[(*x*, *y*, *z*, *t*)] == *Quad* (*SPair x y*) (*SPair z t*)

instance *Quad-type-const* :: (*type*) *type* ..

23.7 Lemmas and proof tool setup.

lemma [(*a*, *b*, *c*, *e*)] = [(*a'*, *b'*, *c'*, *e'*)]
= (*a* = *a'* & *b* = *b'* & *c* = *c'* & *e* = *e'*)
by *simp*

lemma *Quad-eq* [*iff*]:
([(*a*, *b*, *c*, *e*)] = [(*a'*, *b'*, *c'*, *e'*)])
= (*a* = *a'* & *b* = *b'* & *c* = *c'* & *e* = *e'*)
by *auto*

lemma *fst-quad-conv* [*simp*,*code*]:
fst-quad [(*a*, *b*, *c*, *e*)] = *a*
unfolding *fst-quad-def* **by** *blast*

lemma *snd-quad-conv* [*simp*,*code*]:
snd-quad [(*a*, *b*, *c*, *e*)] = *b*
unfolding *snd-quad-def* **by** *blast*

lemma *thrd-quad-conv* [*simp*,*code*]:
thrd-quad [(*a*, *b*, *c*, *e*)] = *c*
unfolding *thrd-quad-def* **by** *blast*

lemma *frth-quad-conv* [*simp*,*code*]:
frth-quad [(*a*, *b*, *c*, *e*)] = *e*
unfolding *frth-quad-def* **by** *blast*

lemma *fst-quad-eqD*:
fst-quad [(*x*, *y*, *z*, *t*)] = *a* ==> *x* = *a*
by *simp*

lemma *snd-quad-eqD*:

```
  snd-quad [(x, y, z, t)] = a ==> y = a
  by simp

lemma thrd-quad-eqD:
  thrd-quad [(x, y, z, t)] = a ==> z = a
  by simp

lemma frth-quad-eqD:
  frth-quad [(x, y, z, t)] = a ==> t = a
  by simp

lemma surjective-quad:
  p = [(fst-quad p, snd-quad p, thrd-quad p, frth-quad p)]
proof (cases p)
  fix SProd1 SProd2
  show p = Quad SProd1 SProd2 ⟹
    p = Quad
    (SPair (fst-quad p) (snd-quad p))
    (SPair (thrd-quad p) (frth-quad p))
    by (cases SProd1, cases SProd2, auto simp add: surjective-spair)
qed

lemma split-Quad-all:
  (!!x. PROP P x) == (!!a b c e. PROP P [(a, b, c, e)])
proof
  fix a b c e
  assume !!x. PROP P x
  then show PROP P [(a, b, c, e)] .
next
  fix x
  assume !!a b c e. PROP P [(a, b, c, e)]
  from ‹PROP P [(fst-quad (x::'a Quad-type-const),
    snd-quad x, thrd-quad x, frth-quad x)]›
    sym [OF surjective-quad [of x::'a Quad-type-const]]
  show PROP P x by simp
qed
```

We now prove that the introduced four tuples data type is an instance of the type classes needed in the BPL

As far as *int* is not a type class, but a type constructor, we will use *ring* as a type class.

Therefore, we prove that our type constructor *Quad-type-const* with suitable operations is a valid instance of the type class (*ring*) *diff-group-add-pert-hom-bound-exist*

Being *int* a valid instance of *ring*, we can then, in the code generation phase, replace *ring* by its concrete structure *int*

Note that giving an instance of *diff-group-add-pert-hom-bound-exist* requires proving that (*ring*) *diff-group-add-pert-hom-bound-exist* is a differential group, with a perturbation and a homotopy operator, which also satisfy the nilpotency condition. This type class contains all definitions involved in the specification of the series Φ and Ψ

instance *Quad-type-const* :: (*eq*) *eq* **..**

lemma [*code func*]:
  [(*x1*::′*a*::*eq*, *x2*, *x3*, *x4*)] = [(*y1*, *y2*, *y3*, *y4*)]
  ⟷ *x1* = *y1* ∧ *x2* = *y2* ∧ *x3* = *y3* ∧ *x4* = *y4*
  by *auto*

instantiation *Quad-type-const* :: ({*eq*,*ring*}) *ab-semigroup-add*
begin

definition *Quad-type-const-plus-def*:
  a + *b* = [((*fst-quad a* + *fst-quad b*), (*snd-quad a* + *snd-quad b*),
  (*thrd-quad a* + *thrd-quad b*), (*frth-quad a* + *frth-quad b*))]

instance
  by *default* (*simp-all add*: *split-Quad-all Quad-type-const-plus-def*)

end

instantiation *Quad-type-const* :: ({*eq*,*ring*}) *comm-monoid-add*
begin

definition *Quad-type-const-zero-def*:
  0 ≡ [(*0*, *0*, *0*, *0*)]

instance by *default* (*simp-all add*: *split-Quad-all*
  Quad-type-const-plus-def Quad-type-const-zero-def)

end

instantiation *Quad-type-const* :: ({*eq*,*ring*}) *ab-group-add*
begin

definition *Quad-type-const-uminus-def*:
  − *x* = [(− *fst-quad x*, − *snd-quad x*, − *thrd-quad x*, − *frth-quad x*)]

definition *Quad-type-const-minus-def*:
  (*x*::′*a Quad-type-const*) − *y* = *x* + (− *y*)

instance
  apply *default*
  unfolding *Quad-type-const-uminus-def*
  unfolding *Quad-type-const-plus-def*
  unfolding *Quad-type-const-zero-def*
  unfolding *Quad-type-const-minus-def*

unfolding *Quad-type-const-uminus-def*
unfolding *Quad-type-const-plus-def* **by** *simp-all*

end

instantiation *Quad-type-const* :: ({*eq*,*ring*}) *diff-group-add*
begin

definition *Quad-type-const-diff-def*:
diff x ≡ [(*0*, *fst-quad x* + *fst-quad x*, *0*, *thrd-quad x*)]

instance
apply *default*
unfolding *expand-fun-eq*
unfolding *o-apply*
unfolding *Quad-type-const-diff-def*
unfolding *Quad-type-const-plus-def*
unfolding *Quad-type-const-zero-def*
unfolding *Quad-type-const-minus-def*
unfolding *Quad-type-const-uminus-def* **by** *simp-all*

end

instantiation *Quad-type-const* :: ({*eq*,*ring*}) *diff-group-add-pert*
begin

definition *Quad-type-const-pert-def*:
pert x ≡ [(*0*, *fst-quad x* + *thrd-quad x*, *0*, *fst-quad x*)]

instance apply *default*
unfolding *Quad-type-const-pert-def*
unfolding *Quad-type-const-plus-def*
apply *simp*
unfolding *diff-group-add-def*
unfolding *diff-group-add-axioms-def*
unfolding *ab-group-add-def*
unfolding *ab-group-add-axioms-def*
unfolding *comm-monoid-add-def*
unfolding *comm-monoid-add-axioms-def*
unfolding *ab-semigroup-add-def*
unfolding *ab-semigroup-add-axioms-def*
unfolding *semigroup-add-def*
unfolding *Quad-type-const-plus-def*
unfolding *Quad-type-const-zero-def*
unfolding *Quad-type-const-uminus-def Quad-type-const-minus-def*
unfolding *Quad-type-const-diff-def*
unfolding *Quad-type-const-plus-def*
unfolding *expand-fun-eq*
by (*simp add*: *sym* [*OF surjective-quad*])

end

instantiation *Quad-type-const* :: ({*eq*,*ring*}) *diff-group-add-pert-hom*
begin

definition *Quad-type-const-hom-oper-def*:
hom-oper $x \equiv [(\ 0, \ 0, \ frth\text{-}quad \ x, \ 0)]$

instance apply *default*
unfolding *split-Quad-all*
unfolding *Quad-type-const-hom-oper-def*
unfolding *Quad-type-const-plus-def*
unfolding *Quad-type-const-zero-def*
unfolding *Quad-type-const-minus-def*
unfolding *Quad-type-const-uminus-def*
unfolding *expand-fun-eq* **by** *auto*

end

lemma *funpow-2*:
shows $f\hat{\ }(2::nat) = f \circ f$
unfolding *numerals* (*3*)
unfolding *funpow.simps* (*2*)
unfolding *funpow.simps* (*1*)
unfolding *o-id* **..**

instantiation *Quad-type-const*
:: ({*eq*,*ring*}) *diff-group-add-pert-hom-bound-exist*
begin

instance proof *default*
show $\forall x::'a$ *Quad-type-const*. $\exists n. (\alpha \ \hat{} \ n) \ x = 0$
proof (*rule allI*)
fix $x :: 'a$ *Quad-type-const*
show $\exists n. (\alpha \ \hat{} \ n) \ x = (0)$
unfolding *α-def*
unfolding *Quad-type-const-pert-def*
unfolding *Quad-type-const-hom-oper-def*
unfolding *Quad-type-const-uminus-def*
unfolding *Quad-type-const-zero-def*
apply (*rule exI* [*of* - *2*::*nat*])
unfolding *funpow-2* **by** *simp*
qed
qed

end

23.8 Code generation and examples of execution

definition *foos* :: *int Quad-type-const*
  where *foos* = Φ [((*5*::*int*), *3*, *8*, *9*)]

definition *foos-2*:: *int Quad-type-const*
  where *foos-2* = Φ [((*5*::*int*), (− *6*), *8*, *9*)]

definition *foos-3*:: *int Quad-type-const*
  where *foos-3* = Ψ [((*5*::*int*), *3*, *8*, *9*)]

definition *foos-4*:: *int Quad-type-const*
  where *foos-4* = Ψ [((*5*::*int*), (− *6*), *8*, *9*)]

definition *foos-local-bound-alpha* :: *nat*
  where *foos-local-bound-alpha* = *local-bound* α [((*5*::*int*), *3*, *8*, *9*)]

definition *foos-f* :: *int Quad-type-const* => *int SProd*
  where *foos-f* = *f′* (λ*x*::*int Quad-type-const*.
  (*SPair* (*fst-quad x*) (*snd-quad x*)))

definition *foos-f2*:: *int SProd*
  where *foos-f2* = *foos-f* [((*4*::*int*), *23*, *17*, *1*)]

definition *foos-g*:: *int SProd* => *int Quad-type-const*
  where *foos-g* = *g′* (λ*x*::*int SProd*.
  [(*fst-spair x*, *snd-spair x*, *0*, *0*)])

definition *foos-g2*:: *int Quad-type-const*
  where *foos-g2* = *foos-g* [((*15*::*int*), *8*)]

definition *foos-h*:: *int Quad-type-const* => *int Quad-type-const*
  where *foos-h* = (*h′*::(*int Quad-type-const* => *int Quad-type-const*))

definition *foos-h2*:: *int Quad-type-const*
  where *foos-h2* = *foos-h* [((*12*::*int*), *22*, *19*, *6*)]

definition *foos-dC′*:: (*int SProd* => *int SProd*)
  where *foos-dC′* =
  dC′ (λ*x*::*int Quad-type-const*. *SPair* (*fst-quad x*) (*snd-quad x*))
  (λ*x*::*int SProd*. [(*fst-spair x*, *snd-spair x*, *0*, *0*)])

definition *foos-dC′2* :: *int SProd*
  where *foos-dC′2* = *foos-dC′* [((*5*::*int*), *3*)]

export-code
  foos
  foos-2
  foos-local-bound-alpha
  foos-3

foos-4
foos-f
foos-f2 foos-g foos-g2 foos-h foos-h2 foos-dC′ foos-dC′2
in *SML* **module-name** *ExampleZ4Z2* **file** *example-Z4Z2.ML*

use *example-Z4Z2.ML*
ML *open ExampleZ4Z2*

ML *foos*
ML *foos-2*
ML *foos-3*
ML *foos-4*
ML *foos-f2*
ML *foos-g2*
ML *foos-h2*
ML *foos-dC′2*
ML *foos-local-bound-alpha*

end