

Free Groups

Joachim Breitner

December 14, 2021

Abstract

Free Groups are, in a sense, the most generic kind of group. They are defined over a set of generators with no additional relations in between them. They play an important role in the definition of group presentations and in other fields.

This theory provides the definition of Free Group as the set of fully canceled words in the generators. The universal property is proven, as well as some isomorphisms results about Free Groups.

Contents

1	Cancelation of words of generators and their inverses	1
1.1	Auxiliary results	1
1.1.1	Auxiliary results about relations	1
1.2	Definition of the <i>canceling</i> relation	2
1.2.1	Simple results about canceling	2
1.3	Definition of the <i>cancel-to</i> relation	2
1.3.1	Existence of the normal form	4
1.3.2	Some properties of cancelation	8
1.4	Definition of normalization	10
1.5	Normalization preserves generators	12
1.6	Normalization and renaming generators	13
2	Generators	15
2.1	The subgroup generated by a set	16
2.2	Generators and homomorphisms	17
2.3	Sets of generators	17
2.4	Product of a list of group elements	19
2.5	Isomorphisms	20
3	The Free Group	21
3.1	Inversion	21
3.2	The definition	24
3.3	The universal property	25

4	The Unit Group	30
5	The group C2	31
6	Isomorphisms of Free Groups	32
6.1	The Free Group over the empty set	32
6.2	The Free Group over one generator	32
6.3	Free Groups over isomorphic sets of generators	36
6.4	Bases of isomorphic free groups	39
7	The Ping Pong lemma	43

1 Cancellation of words of generators and their inverses

```

theory Cancellation
imports
  HOL-Proofs-Lambda.Commutation
begin

```

This theory defines cancellation via relations. The one-step relation *cancelsto-1* a b describes that b is obtained from a by removing exactly one pair of generators, while *cancelsto* is the reflexive transitive hull of that relation. Due to confluence, this relation has a normal form, allowing for the definition of *normalize*.

1.1 Auxiliary results

Some lemmas that would be useful in a more general setting are collected beforehand.

1.1.1 Auxiliary results about relations

These were helpfully provided by Andreas Lochbihler.

```

theorem lconfluent-confluent:
   $\llbracket \text{wfP } (R^{\hat{-} - 1}); \bigwedge a b c. R a b \implies R a c \implies \exists d. R^{\hat{*} *} b d \wedge R^{\hat{*} *} c d \rrbracket \implies$ 
  confluent  $R$ 
by(auto simp add: diamond-def commute-def square-def intro: newman)

```

```

lemma confluentD:
   $\llbracket \text{confluent } R; R^{\hat{*} *} a b; R^{\hat{*} *} a c \rrbracket \implies \exists d. R^{\hat{*} *} b d \wedge R^{\hat{*} *} c d$ 
by(auto simp add: commute-def diamond-def square-def)

```

```

lemma tranclp-DomainP:  $R^{\hat{+} +} a b \implies \text{Domainp } R a$ 
by(auto elim: converse-tranclpE)

```

lemma *confluent-unique-normal-form*:

$\llbracket \text{confluent } R; R^{\widehat{**}} a b; R^{\widehat{**}} a c; \neg \text{DomainP } R b; \neg \text{DomainP } R c \rrbracket \implies b = c$
by(*fastforce dest!*: *confluentD*[of *R a b c*] *dest*: *tranclp-DomainP rtranclpD*[**where** *a=b*] *rtranclpD*[**where** *a=c*])

1.2 Definition of the *canceling* relation

type-synonym *'a g-i* = (*bool* × *'a*)

type-synonym *'a word-g-i* = *'a g-i list*

These type aliases encode the notion of a “generator or its inverse” (*'a g-i*) and the notion of a “word in generators and their inverses” (*'a word-g-i*), which form the building blocks of Free Groups.

definition *canceling* :: *'a g-i* ⇒ *'a g-i* ⇒ *bool*

where *canceling a b* = ((*snd a* = *snd b*) ∧ (*fst a* ≠ *fst b*))

1.2.1 Simple results about canceling

A generators cancels with its inverse, either way. The relation is symmetric.

lemma *cancel-cancel*: $\llbracket \text{canceling } a b; \text{canceling } b c \rrbracket \implies a = c$

by (*auto intro: prod-eqI simp add:canceling-def*)

lemma *cancel-sym*: *canceling a b* ⇒ *canceling b a*

by (*simp add:canceling-def*)

lemma *cancel-sym-neg*: ¬*canceling a b* ⇒ ¬*canceling b a*

by (*rule classical, simp add:canceling-def*)

1.3 Definition of the *cancel-to* relation

First, we define the function that removes the *i*th and (*i+1*)st element from a word of generators, together with basic properties.

definition *cancel-at* :: *nat* ⇒ *'a word-g-i* ⇒ *'a word-g-i*

where *cancel-at i l* = *take i l* @ *drop (2+i) l*

lemma *cancel-at-length*[*simp*]:

$1+i < \text{length } l \implies \text{length } (\text{cancel-at } i l) = \text{length } l - 2$

by(*auto simp add: cancel-at-def*)

lemma *cancel-at-nth1*[*simp*]:

$\llbracket n < i; 1+i < \text{length } l \rrbracket \implies (\text{cancel-at } i l) ! n = l ! n$

by(*auto simp add: cancel-at-def nth-append*)

lemma *cancel-at-nth2*[*simp*]:

assumes $n \geq i$ **and** $n < \text{length } l - 2$

shows $(\text{cancel-at } i l) ! n = l ! (n + 2)$

proof –

from $\langle n \geq i \rangle$ **and** $\langle n < \text{length } l - 2 \rangle$

```

have  $i = \min (\text{length } l) i$ 
  by auto
with  $\langle n \geq i \rangle$  and  $\langle n < \text{length } l - 2 \rangle$ 
show  $(\text{cancel-at } i \ l) ! n = l ! (n + 2)$ 
  by(auto simp add: cancel-at-def nth-append nth-via-drop)
qed

```

Then we can define the relation *cancel-to-1-at* $i \ a \ b$ which specifies that b can be obtained by a by canceling the i th and $(i+1)$ st position.

Based on that, we existentially quantify over the position i to obtain the relation *cancel-to-1*, of which *cancel-to* is the reflexive and transitive closure.

A word is *canceled* if it can not be canceled any futher.

```

definition cancel-to-1-at ::  $\text{nat} \Rightarrow 'a \ \text{word-g-i} \Rightarrow 'a \ \text{word-g-i} \Rightarrow \text{bool}$ 
where  $\text{cancel-to-1-at } i \ l1 \ l2 = (0 \leq i \wedge (1+i) < \text{length } l1$ 
   $\wedge \text{canceling } (l1 ! i) (l1 ! (1+i))$ 
   $\wedge (l2 = \text{cancel-at } i \ l1))$ 

```

```

definition cancel-to-1 ::  $'a \ \text{word-g-i} \Rightarrow 'a \ \text{word-g-i} \Rightarrow \text{bool}$ 
where  $\text{cancel-to-1 } l1 \ l2 = (\exists i. \text{cancel-to-1-at } i \ l1 \ l2)$ 

```

```

definition cancel-to ::  $'a \ \text{word-g-i} \Rightarrow 'a \ \text{word-g-i} \Rightarrow \text{bool}$ 
where  $\text{cancel-to} = \text{cancel-to-1}^{\wedge**}$ 

```

```

lemma cancel-to-trans [trans]:
   $[[ \text{cancel-to } a \ b; \text{cancel-to } b \ c ] \Longrightarrow \text{cancel-to } a \ c$ 
by (auto simp add:cancel-to-def)

```

```

definition canceled ::  $'a \ \text{word-g-i} \Rightarrow \text{bool}$ 
where  $\text{canceled } l = (\neg \text{Domainp } \text{cancel-to-1 } l)$ 

```

```

lemma cancel-to-1-unfold:
  assumes  $\text{cancel-to-1 } x \ y$ 
  obtains  $xs1 \ x1 \ x2 \ xs2$ 
  where  $x = xs1 @ x1 \# x2 \# xs2$ 
  and  $y = xs1 @ xs2$ 
  and  $\text{canceling } x1 \ x2$ 

```

proof –

```

  assume  $a: (\wedge xs1 \ x1 \ x2 \ xs2. [[x = xs1 @ x1 \# x2 \# xs2; y = xs1 @ xs2; \text{canceling } x1 \ x2]] \Longrightarrow \text{thesis})$ 

```

```

  from  $\langle \text{cancel-to-1 } x \ y \rangle$ 

```

```

  obtain  $i$  where  $\text{cancel-to-1-at } i \ x \ y$ 

```

```

  unfolding cancel-to-1-def by auto

```

```

  hence  $\text{canceling } (x ! i) (x ! \text{Suc } i)$ 

```

```

  and  $y = (\text{take } i \ x) @ (\text{drop } (\text{Suc } (\text{Suc } i)) \ x)$ 

```

```

  and  $x = (\text{take } i \ x) @ x ! i \# x ! \text{Suc } i \# (\text{drop } (\text{Suc } (\text{Suc } i)) \ x)$ 

```

```

  unfolding cancel-at-def and cancel-to-1-at-def by (auto simp add: Cons-nth-drop-Suc)

```

```

  with  $a$  show thesis by blast

```

qed

lemma *cancels-to-1-fold*:

canceling $x1\ x2 \implies \text{cancels-to-1 } (xs1\ @\ x1\ \#\ x2\ \#\ xs2)\ (xs1\ @\ xs2)$
unfolding *cancels-to-1-def* **and** *cancels-to-1-at-def* **and** *cancel-at-def*
by (*rule-tac* $x=length\ xs1$ **in** exI , *auto simp add:nth-append*)

1.3.1 Existence of the normal form

One of two steps to show that we have a normal form is the following lemma, guaranteeing that by canceling, we always end up at a fully canceled word.

lemma *canceling-terminates*: $wfP\ (\text{cancels-to-1}^{\wedge--}1)$

proof –

have $wf\ (\text{measure length})$ **by** *auto*

moreover

have $\{(x, y). \text{cancels-to-1 } y\ x\} \subseteq \text{measure length}$

by (*auto simp add: cancels-to-1-def cancel-at-def cancels-to-1-at-def*)

ultimately

have $wf\ \{(x, y). \text{cancels-to-1 } y\ x\}$

by(*rule wf-subset*)

thus *?thesis* **by** (*simp add:wfP-def*)

qed

The next two lemmas prepare for the proof of confluence. It does not matter in which order we cancel, we can obtain the same result.

lemma *canceling-neighbor*:

assumes *cancels-to-1-at* $i\ l\ a$ **and** *cancels-to-1-at* $(\text{Suc } i)\ l\ b$

shows $a = b$

proof –

from $\langle \text{cancels-to-1-at } i\ l\ a \rangle$

have *canceling* $(l\ !\ i)\ (l\ !\ \text{Suc } i)$ **and** $i < \text{length } l$

by (*auto simp add: cancels-to-1-at-def*)

from $\langle \text{cancels-to-1-at } (\text{Suc } i)\ l\ b \rangle$

have *canceling* $(l\ !\ \text{Suc } i)\ (l\ !\ \text{Suc } (\text{Suc } i))$ **and** $\text{Suc } (\text{Suc } i) < \text{length } l$

by (*auto simp add: cancels-to-1-at-def*)

from $\langle \text{canceling } (l\ !\ i)\ (l\ !\ \text{Suc } i) \rangle$ **and** $\langle \text{canceling } (l\ !\ \text{Suc } i)\ (l\ !\ \text{Suc } (\text{Suc } i)) \rangle$

have $l\ !\ i = l\ !\ \text{Suc } (\text{Suc } i)$ **by** (*rule cancel-cancel*)

from $\langle \text{cancels-to-1-at } (\text{Suc } i)\ l\ b \rangle$

have $b = \text{take } (\text{Suc } i)\ l\ @\ \text{drop } (\text{Suc } (\text{Suc } (\text{Suc } i)))\ l$

by (*simp add: cancels-to-1-at-def cancel-at-def*)

also from $\langle i < \text{length } l \rangle$

have $\dots = \text{take } i\ l\ @\ [l\ !\ i]\ @\ \text{drop } (\text{Suc } (\text{Suc } (\text{Suc } i)))\ l$

by(*auto simp add: take-Suc-conv-app-nth*)

also from $\langle l\ !\ i = l\ !\ \text{Suc } (\text{Suc } i) \rangle$

have $\dots = \text{take } i\ l\ @\ [l\ !\ \text{Suc } (\text{Suc } i)]\ @\ \text{drop } (\text{Suc } (\text{Suc } (\text{Suc } i)))\ l$

by *simp*
 also from $\langle \text{Suc } (\text{Suc } i) < \text{length } l \rangle$
 have $\dots = \text{take } i \ l @ \ \text{drop } (\text{Suc } (\text{Suc } i)) \ l$
 by (*simp add: Cons-nth-drop-Suc*)
 also from $\langle \text{cancels-to-1-at } i \ l \ a \rangle$ have $\dots = a$
 by (*simp add: cancels-to-1-at-def cancel-at-def*)
 finally show $a = b$ by(*rule sym*)
 qed

lemma *canceling-indep*:

assumes *cancels-to-1-at* $i \ l \ a$ and *cancels-to-1-at* $j \ l \ b$ and $j > \text{Suc } i$
 obtains c where *cancels-to-1-at* $(j - 2) \ a \ c$ and *cancels-to-1-at* $i \ b \ c$
 proof(*atomize-elim*)

from $\langle \text{cancels-to-1-at } i \ l \ a \rangle$
 have $\text{Suc } i < \text{length } l$
 and *canceling* $(l ! i) \ (l ! \text{Suc } i)$
 and $a = \text{cancel-at } i \ l$
 and $\text{length } a = \text{length } l - 2$
 and $\text{min } (\text{length } l) \ i = i$
 by (*auto simp add:cancels-to-1-at-def*)
 from $\langle \text{cancels-to-1-at } j \ l \ b \rangle$
 have $\text{Suc } j < \text{length } l$
 and *canceling* $(l ! j) \ (l ! \text{Suc } j)$
 and $b = \text{cancel-at } j \ l$
 and $\text{length } b = \text{length } l - 2$
 by (*auto simp add:cancels-to-1-at-def*)

let $?c = \text{cancel-at } (j - 2) \ a$
 from $\langle j > \text{Suc } i \rangle$
 have $\text{Suc } (\text{Suc } (j - 2)) = j$
 and $\text{Suc } (\text{Suc } (\text{Suc } j - 2)) = \text{Suc } j$
 by *auto*
 with $\langle \text{min } (\text{length } l) \ i = i \rangle$ and $\langle j > \text{Suc } i \rangle$ and $\langle \text{Suc } j < \text{length } l \rangle$
 have $(l ! j) = (\text{cancel-at } i \ l ! (j - 2))$
 and $(l ! (\text{Suc } j)) = (\text{cancel-at } i \ l ! \text{Suc } (j - 2))$
 by(*auto simp add:cancel-at-def simp add:nth-append*)

with $\langle \text{cancels-to-1-at } i \ l \ a \rangle$
 and $\langle \text{cancels-to-1-at } j \ l \ b \rangle$
 have *canceling* $(a ! (j - 2)) \ (a ! \text{Suc } (j - 2))$
 by(*auto simp add:cancels-to-1-at-def*)

with $\langle j > \text{Suc } i \rangle$ and $\langle \text{Suc } j < \text{length } l \rangle$ and $\langle \text{length } a = \text{length } l - 2 \rangle$
 have *cancels-to-1-at* $(j - 2) \ a \ ?c$ by (*auto simp add: cancels-to-1-at-def*)

from $\langle \text{length } b = \text{length } l - 2 \rangle$ and $\langle j > \text{Suc } i \rangle$ and $\langle \text{Suc } j < \text{length } l \rangle$
 have $\text{Suc } i < \text{length } b$ by *auto*

moreover from $\langle b = \text{cancel-at } j \ l \rangle$ and $\langle j > \text{Suc } i \rangle$ and $\langle \text{Suc } i < \text{length } l \rangle$

have $(b ! i) = (l ! i)$ **and** $(b ! \text{Suc } i) = (l ! \text{Suc } i)$
by $(\text{auto simp add:cancel-at-def nth-append})$
with $\langle \text{canceling } (l ! i) (l ! \text{Suc } i) \rangle$
have $\text{canceling } (b ! i) (b ! \text{Suc } i)$ **by** simp

moreover from $\langle j > \text{Suc } i \rangle$ **and** $\langle \text{Suc } j < \text{length } l \rangle$
have $\text{min } i j = i$
and $\text{min } (j - 2) i = i$
and $\text{min } (\text{length } l) j = j$
and $\text{min } (\text{length } l) i = i$
and $\text{Suc } (\text{Suc } (j - 2)) = j$
by auto
with $\langle a = \text{cancel-at } i l \rangle$ **and** $\langle b = \text{cancel-at } j l \rangle$ **and** $\langle \text{Suc } (\text{Suc } (j - 2)) = j \rangle$
have $\text{cancel-at } (j - 2) a = \text{cancel-at } i b$
by $(\text{auto simp add:cancel-at-def take-drop})$

ultimately have $\text{cancels-to-1-at } i b (\text{cancel-at } (j - 2) a)$
by $(\text{auto simp add:cancels-to-1-at-def})$

with $\langle \text{cancels-to-1-at } (j - 2) a ?c \rangle$
show $\exists c. \text{cancels-to-1-at } (j - 2) a c \wedge \text{cancels-to-1-at } i b c$ **by** blast
qed

This is the confluence lemma

lemma $\text{confluent-cancels-to-1: confluent cancels-to-1}$
proof $(\text{rule } l\text{confluent-confluent})$
show $\text{wfP } \text{cancels-to-1}^{-1-1}$ **by** $(\text{rule } \text{canceling-terminates})$
next
fix $a b c$
assume $\text{cancels-to-1 } a b$
then obtain i **where** $\text{cancels-to-1-at } i a b$
by $(\text{simp add: cancels-to-1-def})(\text{erule } \text{exE})$
assume $\text{cancels-to-1 } a c$
then obtain j **where** $\text{cancels-to-1-at } j a c$
by $(\text{simp add: cancels-to-1-def})(\text{erule } \text{exE})$

show $\exists d. \text{cancels-to-1}^{**} b d \wedge \text{cancels-to-1}^{**} c d$
proof $(\text{cases } i=j)$
assume $i=j$
from $\langle \text{cancels-to-1-at } i a b \rangle$
have $b = \text{cancel-at } i a$ **by** $(\text{simp add:cancels-to-1-at-def})$
moreover from $\langle i=j \rangle$
have $\dots = \text{cancel-at } j a$ **by** (clarify)
moreover from $\langle \text{cancels-to-1-at } j a c \rangle$
have $\dots = c$ **by** $(\text{simp add:cancels-to-1-at-def})$
ultimately have $b = c$ **by** (simp)
hence $\text{cancels-to-1}^{**} b b$
and $\text{cancels-to-1}^{**} c b$ **by** auto
thus $\exists d. \text{cancels-to-1}^{**} b d \wedge \text{cancels-to-1}^{**} c d$ **by** blast

```

next
  assume  $i \neq j$ 
  show ?thesis
  proof (cases  $j = \text{Suc } i$ )
    assume  $j = \text{Suc } i$ 
    with  $\langle \text{cancels-to-1-at } i \ a \ b \rangle$  and  $\langle \text{cancels-to-1-at } j \ a \ c \rangle$ 
    have  $b = c$  by (auto elim: canceling-neighbor)
    hence  $\text{cancels-to-1}^{**} \ b \ b$ 
    and  $\text{cancels-to-1}^{**} \ c \ b$  by auto
    thus  $\exists d. \text{cancels-to-1}^{**} \ b \ d \wedge \text{cancels-to-1}^{**} \ c \ d$  by blast
  next
  assume  $j \neq \text{Suc } i$ 
  show ?thesis
  proof (cases  $i = \text{Suc } j$ )
    assume  $i = \text{Suc } j$ 
    with  $\langle \text{cancels-to-1-at } i \ a \ b \rangle$  and  $\langle \text{cancels-to-1-at } j \ a \ c \rangle$ 
    have  $c = b$  by (auto elim: canceling-neighbor)
    hence  $\text{cancels-to-1}^{**} \ b \ b$ 
    and  $\text{cancels-to-1}^{**} \ c \ b$  by auto
    thus  $\exists d. \text{cancels-to-1}^{**} \ b \ d \wedge \text{cancels-to-1}^{**} \ c \ d$  by blast
  next
  assume  $i \neq \text{Suc } j$ 
  show ?thesis
  proof (cases  $i < j$ )
    assume  $i < j$ 
    with  $\langle j \neq \text{Suc } i \rangle$  have  $\text{Suc } i < j$  by auto
    with  $\langle \text{cancels-to-1-at } i \ a \ b \rangle$  and  $\langle \text{cancels-to-1-at } j \ a \ c \rangle$ 
    obtain  $d$  where  $\text{cancels-to-1-at } (j - 2) \ b \ d$  and  $\text{cancels-to-1-at } i \ c \ d$ 
    by (erule canceling-indep)
    hence  $\text{cancels-to-1} \ b \ d$  and  $\text{cancels-to-1} \ c \ d$ 
    by (auto simp add: cancels-to-1-def)
    thus  $\exists d. \text{cancels-to-1}^{**} \ b \ d \wedge \text{cancels-to-1}^{**} \ c \ d$  by (auto)
  next
  assume  $\neg i < j$ 
  with  $\langle j \neq \text{Suc } i \rangle$  and  $\langle i \neq j \rangle$  and  $\langle i \neq \text{Suc } j \rangle$  have  $\text{Suc } j < i$  by auto
  with  $\langle \text{cancels-to-1-at } i \ a \ b \rangle$  and  $\langle \text{cancels-to-1-at } j \ a \ c \rangle$ 
  obtain  $d$  where  $\text{cancels-to-1-at } (i - 2) \ c \ d$  and  $\text{cancels-to-1-at } j \ b \ d$ 
  by (erule canceling-indep)
  hence  $\text{cancels-to-1} \ b \ d$  and  $\text{cancels-to-1} \ c \ d$ 
  by (auto simp add: cancels-to-1-def)
  thus  $\exists d. \text{cancels-to-1}^{**} \ b \ d \wedge \text{cancels-to-1}^{**} \ c \ d$  by (auto)
qed
qed
qed
qed

```

And finally, we show that there exists a unique normal form for each word.


```

lemma norm-form-unig:
  assumes cancels-to a b
    and cancels-to a c
    and canceled b
    and canceled c
  shows  $b = c$ 
proof –
  have confluent cancels-to-1 by (rule confluent-cancels-to-1)
  moreover
  from  $\langle \text{cancels-to } a \ b \rangle$  have  $\text{cancels-to-1}^{\wedge**} a \ b$  by (simp add: cancels-to-def)
  moreover
  from  $\langle \text{cancels-to } a \ c \rangle$  have  $\text{cancels-to-1}^{\wedge**} a \ c$  by (simp add: cancels-to-def)
  moreover
  from  $\langle \text{canceled } b \rangle$  have  $\neg \text{Domainp cancels-to-1 } b$  by (simp add: canceled-def)
  moreover
  from  $\langle \text{canceled } c \rangle$  have  $\neg \text{Domainp cancels-to-1 } c$  by (simp add: canceled-def)
  ultimately
  show  $b = c$ 
    by (rule confluent-unique-normal-form)
qed

```

1.3.2 Some properties of cancelation

Distributivity rules of cancelation and *append*.

```

lemma cancel-to-1-append:
  assumes cancels-to-1 a b
  shows cancels-to-1 (l@a@l') (l@b@l')
proof –
  from  $\langle \text{cancels-to-1 } a \ b \rangle$  obtain  $i$  where cancels-to-1-at i a b
    by(simp add: cancels-to-1-def)(erule exE)
  hence cancels-to-1-at (length l + i) (l@a@l') (l@b@l')
    by (auto simp add:cancels-to-1-at-def nth-append cancel-at-def)
  thus cancels-to-1 (l@a@l') (l@b@l')
    by (auto simp add: cancels-to-1-def)
qed

```

```

lemma cancel-to-append:
  assumes cancels-to a b
  shows cancels-to (l@a@l') (l@b@l')
using assms
unfolding cancels-to-def
proof(induct)
  case base show ?case by (simp add:cancels-to-def)
next
  case (step b c)
  from  $\langle \text{cancels-to-1 } b \ c \rangle$ 
  have cancels-to-1 (l @ b @ l') (l @ c @ l') by (rule cancel-to-1-append)
  with  $\langle \text{cancels-to-1}^{\wedge**} (l @ a @ l') (l @ b @ l') \rangle$  show ?case
    by (auto simp add:cancels-to-def)

```

qed

```
lemma cancels-to-append2:
  assumes cancels-to a a'
    and cancels-to b b'
  shows cancels-to (a@b) (a'@b')
using <cancels-to a a'>
unfolding cancels-to-def
proof(induct)
  case base
  from <cancels-to b b'> have cancels-to (a@b@[]) (a'@b'@[])
    by (rule cancel-to-append)
  thus ?case unfolding cancels-to-def by simp
next
  case (step ba c)
  from <cancels-to-1 ba c> have cancels-to-1 ([]@ba@b') ([]@c@b')
    by(rule cancel-to-1-append)
  with <cancels-to-1^** (a @ b) (ba @ b')>
  show ?case unfolding cancels-to-def by simp
qed
```

The empty list is canceled, a one letter word is canceled and a word is trivially canceled from itself.

```
lemma empty-canceled[simp]: canceled []
by(auto simp add: canceled-def cancels-to-1-def cancels-to-1-at-def)
```

```
lemma singleton-canceled[simp]: canceled [a]
by(auto simp add: canceled-def cancels-to-1-def cancels-to-1-at-def)
```

```
lemma cons-canceled:
  assumes canceled (a#x)
  shows canceled x
proof(rule ccontr)
  assume  $\neg$  canceled x
  hence Domainp cancels-to-1 x by (simp add:canceled-def)
  then obtain x' where cancels-to-1 x x' by auto
  then obtain xs1 x1 x2 xs2
    where x: x = xs1 @ x1 # x2 # xs2
    and canceling x1 x2 by (rule cancels-to-1-unfold)
  hence cancels-to-1 ((a#xs1) @ x1 # x2 # xs2) ((a#xs1) @ xs2)
    by (auto intro:cancels-to-1-fold simp del:append-Cons)
  with x
  have cancels-to-1 (a#x) (a#xs1 @ xs2)
    by simp
  hence  $\neg$  canceled (a#x) by (auto simp add:canceled-def)
  thus False using <canceled (a#x)> by contradiction
qed
```

```
lemma cancels-to-self[simp]: cancels-to l l
```

by (simp add: cancels-to-def)

1.4 Definition of normalization

Using the THE construct, we can define the normalization function *normalize* as the unique fully canceled word that the argument cancels to.

definition *normalize* :: 'a word-g-i \Rightarrow 'a word-g-i
where *normalize* l = (THE l'. cancels-to l l' \wedge canceled l')

Some obvious properties of the normalize function, and other useful lemmas.

lemma

shows *normalized-canceled*[simp]: canceled (normalize l)
and *normalized-cancels-to*[simp]: cancels-to l (normalize l)

proof –

let ?Q = {l'. cancels-to-1[^]** l l'}
have l \in ?Q **by** (auto) **hence** $\exists x. x \in ?Q$ **by** (rule exI)

have wfP *cancels-to-1*[^]--1

by (rule canceling-terminates)

hence $\forall Q. (\exists x. x \in Q) \longrightarrow (\exists z \in Q. \forall y. \text{cancels-to-1 } z y \longrightarrow y \notin Q)$

by (simp add: wfP-eq-minimal)

hence $(\exists x. x \in ?Q) \longrightarrow (\exists z \in ?Q. \forall y. \text{cancels-to-1 } z y \longrightarrow y \notin ?Q)$

by (erule-tac x=?Q in allE)

then obtain l' **where** l' \in ?Q **and** *minimal*: $\bigwedge y. \text{cancels-to-1 } l' y \Longrightarrow y \notin ?Q$
by auto

from $\langle l' \in ?Q \rangle$ **have** *cancels-to* l l' **by** (auto simp add: cancels-to-def)

have canceled l'

proof(rule ccontr)

assume \neg canceled l' **hence** *Domainp* *cancels-to-1* l' **by** (simp add: canceled-def)

then obtain y **where** *cancels-to-1* l' y **by** auto

with $\langle \text{cancels-to } l l' \rangle$ **have** *cancels-to* l y **by** (auto simp add: cancels-to-def)

from $\langle \text{cancels-to-1 } l' y \rangle$ **have** $y \notin ?Q$ **by**(rule minimal)

hence \neg *cancels-to-1*[^]** l y **by** auto

hence \neg *cancels-to* l y **by** (simp add: cancels-to-def)

with $\langle \text{cancels-to } l y \rangle$ **show** False **by** contradiction

qed

from $\langle \text{cancels-to } l l' \rangle$ **and** $\langle \text{canceled } l' \rangle$

have *cancels-to* l l' \wedge canceled l' **by** simp

hence *cancels-to* l (normalize l) \wedge canceled (normalize l)

unfolding *normalize-def*

proof (rule theI)

fix l'a

assume *cancels-to* l l'a \wedge canceled l'a

thus l'a = l' **using** $\langle \text{cancels-to } l l' \wedge \text{canceled } l' \rangle$ **by** (auto elim: norm-form-uniq)

```

qed
thus canceled (normalize l) and cancels-to l (normalize l) by auto
qed

```

```

lemma normalize-discover:
  assumes canceled l'
    and cancels-to l l'
  shows normalize l = l'
proof -
  from ⟨canceled l'⟩ and ⟨cancels-to l l'⟩
  have cancels-to l l' ∧ canceled l' by auto
  thus ?thesis unfolding normalize-def by (auto elim: norm-form-uniq)
qed

```

Words, related by cancelation, have the same normal form.

```

lemma normalize-canceled[simp]:
  assumes cancels-to l l'
  shows normalize l = normalize l'
proof(rule normalize-discover)
  show canceled (normalize l') by (rule normalized-canceled)
next
  have cancels-to l' (normalize l') by (rule normalized-cancels-to)
  with ⟨cancels-to l l'⟩
  show cancels-to l (normalize l') by (rule cancels-to-trans)
qed

```

Normalization is idempotent.

```

lemma normalize-idemp[simp]:
  assumes canceled l
  shows normalize l = l
using assms
by(rule normalize-discover)(rule cancels-to-self)

```

This lemma lifts the distributivity results from above to the normalize function.

```

lemma normalize-append-cancel-to:
  assumes cancels-to l1 l1'
    and cancels-to l2 l2'
  shows normalize (l1 @ l2) = normalize (l1' @ l2')
proof(rule normalize-discover)
  show canceled (normalize (l1' @ l2')) by (rule normalized-canceled)
next
  from ⟨cancels-to l1 l1'⟩ and ⟨cancels-to l2 l2'⟩
  have cancels-to (l1 @ l2) (l1' @ l2') by (rule cancels-to-append2)
  also
  have cancels-to (l1' @ l2') (normalize (l1' @ l2')) by (rule normalized-cancels-to)
  finally
  show cancels-to (l1 @ l2) (normalize (l1' @ l2')).
qed

```

1.5 Normalization preserves generators

Somewhat obvious, but still required to formalize Free Groups, is the fact that canceling a word of generators of a specific set (and their inverses) results in a word in generators from that set.

lemma *cancel-to-1-preserves-generators*:

assumes *cancel-to-1* $l\ l'$
and $l \in \text{lists } (UNIV \times \text{gens})$
shows $l' \in \text{lists } (UNIV \times \text{gens})$

proof –

from *assms* **obtain** i **where** $l' = \text{cancel-at } i\ l$
unfolding *cancel-to-1-def* **and** *cancel-to-1-at-def* **by** *auto*
hence $l' = \text{take } i\ l @ \text{drop } (2 + i)\ l$ **unfolding** *cancel-at-def* .
hence $\text{set } l' = \text{set } (\text{take } i\ l @ \text{drop } (2 + i)\ l)$ **by** *simp*
moreover
have $\dots = \text{set } (\text{take } i\ l @ \text{drop } (2 + i)\ l)$ **by** *auto*
moreover
have $\dots \subseteq \text{set } (\text{take } i\ l) \cup \text{set } (\text{drop } (2 + i)\ l)$ **by** *auto*
moreover
have $\dots \subseteq \text{set } l$ **by** (*auto dest: in-set-takeD in-set-dropD*)
ultimately
have $\text{set } l' \subseteq \text{set } l$ **by** *simp*
thus *?thesis* **using** *assms(2)* **by** *auto*

qed

lemma *cancel-to-preserves-generators*:

assumes *cancel-to* $l\ l'$
and $l \in \text{lists } (UNIV \times \text{gens})$
shows $l' \in \text{lists } (UNIV \times \text{gens})$

using *assms* **unfolding** *cancel-to-def* **by** (*induct, auto dest:cancel-to-1-preserves-generators*)

lemma *normalize-preserves-generators*:

assumes $l \in \text{lists } (UNIV \times \text{gens})$
shows *normalize* $l \in \text{lists } (UNIV \times \text{gens})$

proof –

have *cancel-to* l (*normalize* l) **by** *simp*
thus *?thesis* **using** *assms* **by**(*rule cancel-to-preserves-generators*)

qed

Two simplification lemmas about lists.

lemma *empty-in-lists[simp]*:

$[] \in \text{lists } A$ **by** *auto*

lemma *lists-empty[simp]*: $\text{lists } \{\} = \{[]\}$

by *auto*

1.6 Normalization and renaming generators

Renaming the generators, i.e. mapping them through an injective function, commutes with normalization. Similarly, replacing generators by their inverses and vica-versa commutes with normalization. Both operations are similar enough to be handled at once here.

lemma *rename-gens-cancel-at*: $\text{cancel-at } i \text{ (map } f \text{ } l) = \text{map } f \text{ (cancel-at } i \text{ } l)$
unfolding *cancel-at-def* **by** (*auto simp add:take-map drop-map*)

lemma *rename-gens-cancels-to-1*:

assumes *inj f*

and *cancels-to-1 l l'*

shows *cancels-to-1 (map (map-prod f g) l) (map (map-prod f g) l')*

proof –

from $\langle \text{cancels-to-1 } l \text{ } l' \rangle$

obtain *ls1 l1 l2 ls2*

where $l = \text{ls1 } @ \text{ } l1 \# \text{ } l2 \# \text{ } ls2$

and $l' = \text{ls1 } @ \text{ } ls2$

and *canceling l1 l2*

by (*rule cancels-to-1-unfold*)

from $\langle \text{canceling } l1 \text{ } l2 \rangle$

have $\text{fst } l1 \neq \text{fst } l2$ **and** $\text{snd } l1 = \text{snd } l2$

unfolding *canceling-def* **by** *auto*

from $\langle \text{fst } l1 \neq \text{fst } l2 \rangle$ **and** $\langle \text{inj } f \rangle$

have $f \text{ (fst } l1) \neq f \text{ (fst } l2)$ **by** (*auto dest!:inj-on-contrad*)

hence $\text{fst (map-prod } f \text{ } g \text{ } l1) \neq \text{fst (map-prod } f \text{ } g \text{ } l2)$ **by** *auto*

moreover

from $\langle \text{snd } l1 = \text{snd } l2 \rangle$

have $\text{snd (map-prod } f \text{ } g \text{ } l1) = \text{snd (map-prod } f \text{ } g \text{ } l2)$ **by** *auto*

ultimately

have *canceling (map-prod f g (l1)) (map-prod f g (l2))*

unfolding *canceling-def* **by** *auto*

hence *cancels-to-1 (map (map-prod f g) ls1 @ map-prod f g l1 # map-prod f g l2 # map (map-prod f g) ls2) (map (map-prod f g) ls1 @ map (map-prod f g) ls2)*

by (*rule cancels-to-1-fold*)

with $\langle l = \text{ls1 } @ \text{ } l1 \# \text{ } l2 \# \text{ } ls2 \rangle$ **and** $\langle l' = \text{ls1 } @ \text{ } ls2 \rangle$

show *cancels-to-1 (map (map-prod f g) l) (map (map-prod f g) l')*

by *simp*

qed

lemma *rename-gens-cancels-to*:

assumes *inj f*

and *cancels-to l l'*

shows *cancels-to (map (map-prod f g) l) (map (map-prod f g) l')*

using $\langle \text{cancels-to } l \text{ } l' \rangle$

unfolding *cancels-to-def*

proof (*induct rule:rtranclp-induct*)

case (*step x z*)

from $\langle \text{cancels-to-1 } x \ z \rangle$ **and** $\langle \text{inj } f \rangle$
have $\text{cancels-to-1 } (\text{map } (\text{map-prod } f \ g) \ x) \ (\text{map } (\text{map-prod } f \ g) \ z)$
by $\neg(\text{rule rename-gens-cancels-to-1})$
with $\langle \text{cancels-to-1}^{\wedge} ** (\text{map } (\text{map-prod } f \ g) \ l) \ (\text{map } (\text{map-prod } f \ g) \ x) \rangle$
show $\text{cancels-to-1}^{\wedge} ** (\text{map } (\text{map-prod } f \ g) \ l) \ (\text{map } (\text{map-prod } f \ g) \ z)$ **by** *auto*
qed(*auto*)

lemma *rename-gens-canceled*:

assumes $\text{inj-on } g \ (\text{snd } \text{'set } l)$
and $\text{canceled } l$
shows $\text{canceled } (\text{map } (\text{map-prod } f \ g) \ l)$
unfolding *canceled-def*
proof

have *different-images*: $\bigwedge f \ a \ b. f \ a \neq f \ b \implies a \neq b$ **by** *auto*

assume $\text{Domainp } \text{cancels-to-1 } (\text{map } (\text{map-prod } f \ g) \ l)$
then obtain l' **where** $\text{cancels-to-1 } (\text{map } (\text{map-prod } f \ g) \ l) \ l'$ **by** *auto*
then obtain i **where** $\text{Suc } i < \text{length } l$
and $\text{canceling } (\text{map } (\text{map-prod } f \ g) \ l \ ! \ i) \ (\text{map } (\text{map-prod } f \ g) \ l \ ! \ \text{Suc } i)$
by(*auto simp add:cancels-to-1-def cancels-to-1-at-def*)

hence $f \ (\text{fst } (l \ ! \ i)) \neq f \ (\text{fst } (l \ ! \ \text{Suc } i))$
and $g \ (\text{snd } (l \ ! \ i)) = g \ (\text{snd } (l \ ! \ \text{Suc } i))$
by(*auto simp add:canceling-def*)

from $\langle f \ (\text{fst } (l \ ! \ i)) \neq f \ (\text{fst } (l \ ! \ \text{Suc } i)) \rangle$
have $\text{fst } (l \ ! \ i) \neq \text{fst } (l \ ! \ \text{Suc } i)$ **by** $\neg(\text{erule } \text{different-images})$
moreover

from $\langle \text{Suc } i < \text{length } l \rangle$
have $\text{snd } (l \ ! \ i) \in \text{snd } \text{'set } l$ **and** $\text{snd } (l \ ! \ \text{Suc } i) \in \text{snd } \text{'set } l$ **by** *auto*
with $\langle g \ (\text{snd } (l \ ! \ i)) = g \ (\text{snd } (l \ ! \ \text{Suc } i)) \rangle$
have $\text{snd } (l \ ! \ i) = \text{snd } (l \ ! \ \text{Suc } i)$
using $\langle \text{inj-on } g \ (\text{image } \text{snd } (\text{set } l)) \rangle$
by (*auto dest: inj-onD*)

ultimately

have $\text{canceling } (l \ ! \ i) \ (l \ ! \ \text{Suc } i)$ **unfolding** *canceling-def* **by** *simp*

with $\langle \text{Suc } i < \text{length } l \rangle$

have $\text{cancels-to-1-at } i \ l \ (\text{cancel-at } i \ l)$

unfolding *cancels-to-1-at-def* **by** *auto*

hence $\text{cancels-to-1 } l \ (\text{cancel-at } i \ l)$

unfolding *cancels-to-1-def* **by** *auto*

hence $\neg \text{canceled } l$

unfolding *canceled-def* **by** *auto*

with $\langle \text{canceled } l \rangle$ **show** *False* **by** *contradiction*

qed

lemma *rename-gens-normalize*:

assumes $\text{inj } f$
and $\text{inj-on } g \ (\text{snd } \text{'set } l)$

```

shows normalize (map (map-prod f g) l) = map (map-prod f g) (normalize l)
proof(rule normalize-discover)
from ⟨inj-on g (image snd (set l))⟩
have inj-on g (image snd (set (normalize l)))
proof (rule subset-inj-on)

  have UNIV-snd:  $\bigwedge A. A \subseteq UNIV \times snd \text{ ` } A$ 
  proof fix A and x::'c×'d assume x∈A
    hence (fst x,snd x)∈ (UNIV × snd ` A)
    by -(rule, auto)
  thus x∈ (UNIV × snd ` A) by simp
  qed

  have l ∈ lists (set l) by auto
  hence l ∈ lists (UNIV × snd ` set l)
    by (rule subsetD[OF lists-mono[OF UNIV-snd], of l set l])
  hence normalize l ∈ lists (UNIV × snd ` set l)
    by (rule normalize-preserves-generators[of - snd ` set l])
  thus snd ` set (normalize l) ⊆ snd ` set l
    by (auto simp add: lists-eq-set)
  qed
thus canceled (map (map-prod f g) (normalize l)) by(rule rename-gens-canceled,simp)
next
  from ⟨inj f⟩
  show cancels-to (map (map-prod f g) l) (map (map-prod f g) (normalize l))
    by (rule rename-gens-cancels-to, simp)
qed

end

```

2 Generators

```

theory Generators
imports
  HOL-Algebra.Group
  HOL-Algebra.Lattice
begin

```

This theory is not specific to Free Groups and could be moved to a more general place. It defines the subgroup generated by a set of generators and that homomorphisms agree on the generated subgroup if they agree on the generators.

```

notation subgroup (infix ≤ 80)

```

2.1 The subgroup generated by a set

The span of a set of subgroup generators, i.e. the generated subgroup, can be defined inductively or as the intersection of all subgroups containing the

generators. Here, we define it inductively and proof the equivalence

inductive-set *gen-span* :: ('a,'b) monoid-scheme \Rightarrow 'a set \Rightarrow 'a set ($\langle \cdot \rangle_1$)

for *G* **and** *gens*

where *gen-one* [*intro!*, *simp*]: $\mathbf{1}_G \in \langle gens \rangle_G$

| *gen-gens*: $x \in gens \Longrightarrow x \in \langle gens \rangle_G$

| *gen-inv*: $x \in \langle gens \rangle_G \Longrightarrow inv_G x \in \langle gens \rangle_G$

| *gen-mult*: $\llbracket x \in \langle gens \rangle_G; y \in \langle gens \rangle_G \rrbracket \Longrightarrow x \otimes_G y \in \langle gens \rangle_G$

lemma (**in group**) *gen-span-closed*:

assumes $gens \subseteq carrier\ G$

shows $\langle gens \rangle_G \subseteq carrier\ G$

proof

fix *x*

from *assms* **show** $x \in \langle gens \rangle_G \Longrightarrow x \in carrier\ G$

by $-(induct\ rule:gen-span.induct, auto)$

qed

lemma (**in group**) *gen-subgroup-is-subgroup*:

$gens \subseteq carrier\ G \Longrightarrow \langle gens \rangle_G \leq G$

by(*rule subgroupI*)(*auto intro:gen-span.intros simp add:gen-span-closed*)

lemma (**in group**) *gen-subgroup-is-smallest-containing*:

assumes $gens \subseteq carrier\ G$

shows $\bigcap \{H. H \leq G \wedge gens \subseteq H\} = \langle gens \rangle_G$

proof

show $\langle gens \rangle_G \subseteq \bigcap \{H. H \leq G \wedge gens \subseteq H\}$

proof(*rule Inf-greatest*)

fix *H*

assume $H \in \{H. H \leq G \wedge gens \subseteq H\}$

hence $H \leq G$ **and** $gens \subseteq H$ **by** *auto*

show $\langle gens \rangle_G \subseteq H$

proof

fix *x*

from $\langle H \leq G \rangle$ **and** $\langle gens \subseteq H \rangle$

show $x \in \langle gens \rangle_G \Longrightarrow x \in H$

unfolding *subgroup-def*

by $-(induct\ rule:gen-span.induct, auto)$

qed

qed

next

from $\langle gens \subseteq carrier\ G \rangle$

have $\langle gens \rangle_G \leq G$ **by** (*rule gen-subgroup-is-subgroup*)

moreover

have $gens \subseteq \langle gens \rangle_G$ **by** (*auto intro:gen-span.intros*)

ultimately

show $\bigcap \{H. H \leq G \wedge gens \subseteq H\} \subseteq \langle gens \rangle_G$

by(*auto intro:Inter-lower*)

qed

2.2 Generators and homomorphisms

Two homomorphisms agreeing on some elements agree on the span of those elements.

lemma *hom-unique-on-span*:

assumes *group* G
and *group* H
and $\text{gens} \subseteq \text{carrier } G$
and $h \in \text{hom } G \ H$
and $h' \in \text{hom } G \ H$
and $\forall g \in \text{gens}. h \ g = h' \ g$
shows $\forall x \in \langle \text{gens} \rangle_G. h \ x = h' \ x$

proof

interpret G : *group* G **by** *fact*
interpret H : *group* H **by** *fact*
interpret h : *group-hom* $G \ H \ h$ **by** *unfold-locales fact*
interpret h' : *group-hom* $G \ H \ h'$ **by** *unfold-locales fact*

fix x

from $\langle \text{gens} \subseteq \text{carrier } G \rangle$ **have** $\langle \text{gens} \rangle_G \subseteq \text{carrier } G$ **by** (*rule* $G.\text{gen-span-closed}$)

with *assms* **show** $x \in \langle \text{gens} \rangle_G \implies h \ x = h' \ x$ **apply** $-$

proof(*induct rule:gen-span.induct*)

case (*gen-mult* $x \ y$)

hence $x: x \in \text{carrier } G$ **and** $y: y \in \text{carrier } G$ **and**

$hx: h \ x = h' \ x$ **and** $hy: h \ y = h' \ y$ **by** *auto*

thus $h \ (x \otimes_G y) = h' \ (x \otimes_G y)$ **by** *simp*

qed *auto*

qed

2.3 Sets of generators

There is no definition for “*gens* is a generating set of G ”. This is easily expressed by $\langle \text{gens} \rangle = \text{carrier } G$.

The following is an application of *hom-unique-on-span* on a generating set of the whole group.

lemma (*in group*) *hom-unique-by-gens*:

assumes *group* H
and $\text{gens}: \langle \text{gens} \rangle_G = \text{carrier } G$
and $h \in \text{hom } G \ H$
and $h' \in \text{hom } G \ H$
and $\forall g \in \text{gens}. h \ g = h' \ g$
shows $\forall x \in \text{carrier } G. h \ x = h' \ x$

proof

fix x

from gens **have** $\text{gens} \subseteq \text{carrier } G$ **by** (*auto intro:gen-span.gen-gens*)

with *assms* **and** *group-axioms* **have** $r: \forall x \in \langle \text{gens} \rangle_G. h \ x = h' \ x$

by $-(\text{erule } \text{hom-unique-on-span}, \text{ auto})$

with gens show $x \in \text{carrier } G \implies h x = h' x$ **by auto**
qed

lemma (in *group-hom*) *hom-span*:

assumes $\text{gens} \subseteq \text{carrier } G$

shows $h' (\langle \text{gens} \rangle_G) = \langle h' \text{ gens} \rangle_H$

proof(rule *Set.set-eqI*, rule *iffI*)

from $\langle \text{gens} \subseteq \text{carrier } G \rangle$

have $\langle \text{gens} \rangle_G \subseteq \text{carrier } G$ **by** (rule *G.gen-span-closed*)

fix y

assume $y \in h' \langle \text{gens} \rangle_G$

then obtain x **where** $x \in \langle \text{gens} \rangle_G$ **and** $y = h x$ **by auto**

from $\langle x \in \langle \text{gens} \rangle_G \rangle$

have $h x \in \langle h' \text{ gens} \rangle_H$

proof(*induct x*)

case (*gen-inv x*)

hence $x \in \text{carrier } G$ **and** $h x \in \langle h' \text{ gens} \rangle_H$

using $\langle \langle \text{gens} \rangle_G \subseteq \text{carrier } G \rangle$

by auto

thus *?case* **by** (*auto intro:gen-span.intros*)

next

case (*gen-mult x y*)

hence $x \in \text{carrier } G$ **and** $h x \in \langle h' \text{ gens} \rangle_H$

and $y \in \text{carrier } G$ **and** $h y \in \langle h' \text{ gens} \rangle_H$

using $\langle \langle \text{gens} \rangle_G \subseteq \text{carrier } G \rangle$

by auto

thus *?case* **by** (*auto intro:gen-span.intros*)

qed(*auto intro: gen-span.intros*)

with $\langle y = h x \rangle$

show $y \in \langle h' \text{ gens} \rangle_H$ **by simp**

next

fix x

show $x \in \langle h' \text{ gens} \rangle_H \implies x \in h' \langle \text{gens} \rangle$

proof(*induct x rule:gen-span.induct*)

case (*gen-inv y*)

then obtain x **where** $y = h x$ **and** $x \in \langle \text{gens} \rangle$ **by auto**

moreover

hence $x \in \text{carrier } G$ **using** $\langle \text{gens} \subseteq \text{carrier } G \rangle$

by (*auto dest:G.gen-span-closed*)

ultimately show *?case*

by (*auto intro:hom-inv[THEN sym] rev-image-eqI gen-span.gen-inv simp*

del:group-hom.hom-inv hom-inv)

next

case (*gen-mult y y'*)

then obtain x **and** x'

where $y = h x$ **and** $x \in \langle \text{gens} \rangle$

and $y' = h x'$ **and** $x' \in \langle \text{gens} \rangle$ **by auto**

moreover

hence $x \in \text{carrier } G$ **and** $x' \in \text{carrier } G$ **using** $\langle \text{gens} \subseteq \text{carrier } G \rangle$
by $(\text{auto dest: } G.\text{gen-span-closed})$
ultimately show $?case$
by $(\text{auto intro:hom-mult}[\text{THEN sym}] \text{ rev-image-eqI gen-span.gen-mult simp del:group-hom.hom-mult hom-mult})$
qed $(\text{auto intro:rev-image-eqI intro:gen-span.intros})$
qed

2.4 Product of a list of group elements

Not strictly related to generators of groups, this is still a general group concept and not related to Free Groups.

abbreviation (in monoid) $m\text{-concat}$
where $m\text{-concat } l \equiv \text{foldr } (\otimes) l \mathbf{1}$

lemma (in monoid) $m\text{-concat-closed}[\text{simp}]$:
 $set\ l \subseteq \text{carrier } G \implies m\text{-concat } l \in \text{carrier } G$
by $(\text{induct } l, \text{auto})$

lemma (in monoid) $m\text{-concat-append}[\text{simp}]$:
assumes $set\ a \subseteq \text{carrier } G$
and $set\ b \subseteq \text{carrier } G$
shows $m\text{-concat } (a@b) = m\text{-concat } a \otimes m\text{-concat } b$
using $assms$
by $(\text{induct } a)(\text{auto simp add: } m\text{-assoc})$

lemma (in monoid) $m\text{-concat-cons}[\text{simp}]$:
 $\llbracket x \in \text{carrier } G ; set\ xs \subseteq \text{carrier } G \rrbracket \implies m\text{-concat } (x\#\text{xs}) = x \otimes m\text{-concat } xs$
by $(\text{induct } xs)(\text{auto simp add: } m\text{-assoc})$

lemma (in monoid) nat-pow-mult1l :
assumes $x: x \in \text{carrier } G$
shows $x \otimes x [\wedge] n = x [\wedge] \text{Suc } n$
proof –
have $x \otimes x [\wedge] n = x [\wedge] (1::\text{nat}) \otimes x [\wedge] n$ **using** x **by** auto
also have $\dots = x [\wedge] (1 + n)$ **using** x
by $(\text{auto dest:nat-pow-mult simp del:One-nat-def})$
also have $\dots = x [\wedge] \text{Suc } n$ **by** simp
finally show $x \otimes x [\wedge] n = x [\wedge] \text{Suc } n$.
qed

lemma (in monoid) $m\text{-concat-power}[\text{simp}]$: $x \in \text{carrier } G \implies m\text{-concat } (\text{replicate } n\ x) = x [\wedge] n$
by $(\text{induct } n, \text{auto simp add:nat-pow-mult1l})$

2.5 Isomorphisms

A nicer way of proving that something is a group homomorphism or isomorphism.

lemma *group-homI*[*intro*]:

assumes *range*: $h \text{ ' (carrier } g1) \subseteq \text{carrier } g2$

and *hom*: $\forall x \in \text{carrier } g1. \forall y \in \text{carrier } g1. h (x \otimes_{g1} y) = h x \otimes_{g2} h y$

shows $h \in \text{hom } g1 \ g2$

proof –

have $h \in \text{carrier } g1 \rightarrow \text{carrier } g2$ **using** *range* **by** *auto*

thus $h \in \text{hom } g1 \ g2$ **using** *hom* **unfolding** *hom-def* **by** *auto*

qed

lemma (**in** *group-hom*) *hom-injI*:

assumes $\forall x \in \text{carrier } G. h x = \mathbf{1}_H \longrightarrow x = \mathbf{1}_G$

shows *inj-on* h (*carrier* G)

unfolding *inj-on-def*

proof(*rule ballI*, *rule ballI*, *rule impI*)

fix x

fix y

assume $x: x \in \text{carrier } G$

and $y: y \in \text{carrier } G$

and $h x = h y$

hence $h (x \otimes \text{inv } y) = \mathbf{1}_H$ **and** $x \otimes \text{inv } y \in \text{carrier } G$

by *auto*

with *assms*

have $x \otimes \text{inv } y = \mathbf{1}$ **by** *auto*

thus $x = y$ **using** x **and** y

by(*auto dest: G.inv-equality*)

qed

lemma (**in** *group-hom*) *group-hom-isoI*:

assumes *inj1*: $\forall x \in \text{carrier } G. h x = \mathbf{1}_H \longrightarrow x = \mathbf{1}_G$

and *surj*: $h \text{ ' (carrier } G) = \text{carrier } H$

shows $h \in \text{iso } G \ H$

proof –

from *inj1*

have *inj-on* h (*carrier* G)

by(*auto intro: hom-injI*)

hence *bij*: *bij-betw* h (*carrier* G) (*carrier* H)

using *surj* **unfolding** *bij-betw-def* **by** *auto*

thus *?thesis*

unfolding *iso-def* **by** *auto*

qed

lemma *group-isoI*[*intro*]:

assumes G : *group* G

and H : *group* H

and *inj1*: $\forall x \in \text{carrier } G. h x = \mathbf{1}_H \longrightarrow x = \mathbf{1}_G$

```

    and surj: h ` (carrier G) = carrier H
    and hom:  $\forall x \in \text{carrier } G. \forall y \in \text{carrier } G. h (x \otimes_G y) = h x \otimes_H h y$ 
  shows  $h \in \text{iso } G H$ 
proof -
  from surj
  have  $h \in \text{carrier } G \rightarrow \text{carrier } H$ 
    by auto
  then interpret group-hom G H h using G and H and hom
    by (auto intro!: group-hom.intro group-hom-axioms.intro)
  show ?thesis
    using assms unfolding hom-def by (auto intro: group-hom-isoI)
qed
end

```

3 The Free Group

```

theory FreeGroups
imports
  HOL-Algebra.Group
  Cancelation
  Generators
begin

```

Based on the work in *Free-Groups.Cancelation*, the free group is now easily defined over the set of fully canceled words with the corresponding operations.

3.1 Inversion

To define the inverse of a word, we first create a helper function that inverts a single generator, and show that it is self-inverse.

```

definition inv1 :: 'a g-i  $\Rightarrow$  'a g-i
where inv1 = apfst Not

```

```

lemma inv1-inv1: inv1  $\circ$  inv1 = id
by (simp add: fun-eq-iff comp-def inv1-def)

```

```

lemmas inv1-inv1-simp [simp] = inv1-inv1[unfolded id-def]

```

```

lemma snd-inv1: snd  $\circ$  inv1 = snd
by(simp add: fun-eq-iff comp-def inv1-def)

```

The inverse of a word is obtained by reversing the order of the generators and inverting each generator using *inv1*. Some properties of *inv-fg* are noted.

```

definition inv-fg :: 'a word-g-i  $\Rightarrow$  'a word-g-i
where inv-fg l = rev (map inv1 l)

```

```

lemma cancelling-inf[simp]: canceling (inv1 a) (inv1 b) = canceling a b

```

by(*simp add: canceling-def inv1-def*)

lemma *inv-idemp*: $inv\text{-}fg\ (inv\text{-}fg\ l) = l$
by (*auto simp add:inv-fg-def rev-map*)

lemma *inv-fg-cancel*: $normalize\ (l\ @\ inv\text{-}fg\ l) = []$
proof(*induct l rule:rev-induct*)
case *Nil* **thus** *?case*
by (*auto simp add: inv-fg-def*)
next
case (*snoc x xs*)
have *canceling x (inv1 x)* **by** (*simp add:inv1-def canceling-def*)
moreover
let *?i = length xs*
have *Suc ?i < length xs + 1 + 1 + length xs*
by *auto*
moreover
have $inv\text{-}fg\ (xs\ @\ [x]) = [inv1\ x]\ @\ inv\text{-}fg\ xs$
by (*auto simp add:inv-fg-def*)
ultimately
have *cancels-to-1-at ?i (xs @ [x] @ (inv-fg (xs @ [x]))) (xs @ inv-fg xs)*
by (*auto simp add:cancels-to-1-at-def cancel-at-def nth-append*)
hence *cancels-to-1 (xs @ [x] @ (inv-fg (xs @ [x]))) (xs @ inv-fg xs)*
by (*auto simp add: cancels-to-1-def*)
hence *cancels-to (xs @ [x] @ (inv-fg (xs @ [x]))) (xs @ inv-fg xs)*
by (*auto simp add:cancels-to-def*)
with $\langle normalize\ (xs\ @\ (inv\text{-}fg\ xs)) = [] \rangle$
show $normalize\ ((xs\ @\ [x])\ @\ (inv\text{-}fg\ (xs\ @\ [x]))) = []$
by *auto*
qed

lemma *inv-fg-cancel2*: $normalize\ (inv\text{-}fg\ l\ @\ l) = []$
proof–
have $normalize\ (inv\text{-}fg\ l\ @\ inv\text{-}fg\ (inv\text{-}fg\ l)) = []$ **by** (*rule inv-fg-cancel*)
thus $normalize\ (inv\text{-}fg\ l\ @\ l) = []$ **by** (*simp add: inv-idemp*)
qed

lemma *canceled-rev*:
assumes *canceled l*
shows *canceled (rev l)*
proof(*rule ccontr*)
assume \neg *canceled (rev l)*
hence *Domainp cancels-to-1 (rev l)* **by** (*simp add: canceled-def*)
then obtain *l' where cancels-to-1 (rev l) l'* **by** *auto*
then obtain *i where cancels-to-1-at i (rev l) l'* **by** (*auto simp add:cancels-to-1-def*)
hence *Suc i < length (rev l)*
and *canceling (rev l ! i) (rev l ! Suc i)*
by (*auto simp add:cancels-to-1-at-def*)
let *?x = length l - i - 2*

```

from ⟨Suc i < length (rev l)⟩
have Suc ?x < length l by auto
moreover
from ⟨Suc i < length (rev l)⟩
have i < length l and length l - Suc i = Suc(length l - Suc (Suc i)) by auto
hence rev l ! i = l ! Suc ?x and rev l ! Suc i = l ! ?x
  by (auto simp add: rev-nth map-nth)
with ⟨canceling (rev l ! i) (rev l ! Suc i)⟩
have canceling (l ! Suc ?x) (l ! ?x) by auto
hence canceling (l ! ?x) (l ! Suc ?x) by (rule cancel-sym)
hence canceling (l ! ?x) (l ! Suc ?x) by simp
ultimately
have cancels-to-1-at ?x l (cancel-at ?x l)
  by (auto simp add: cancels-to-1-at-def)
hence cancels-to-1 l (cancel-at ?x l)
  by (auto simp add: cancels-to-1-def)
hence ¬canceled l
  by (auto simp add: canceled-def)
with ⟨canceled l⟩ show False by contradiction
qed

```

```

lemma inv-fg-closure1:
  assumes canceled l
  shows canceled (inv-fg l)
unfolding inv-fg-def and inv1-def and apfst-def
proof –
  have inj Not by (auto intro: injI)
  moreover
  have inj-on id (snd ‘ set l) by auto
  ultimately
  have canceled (map (map-prod Not id) l)
    using ⟨canceled l⟩
    by -(rule rename-gens-canceled)
  thus canceled (rev (map (map-prod Not id) l)) by (rule canceled-rev)
qed

```

```

lemma inv-fg-closure2:
  l ∈ lists (UNIV × gens) ⇒ inv-fg l ∈ lists (UNIV × gens)
  by (auto iff: lists-eq-set simp add: inv1-def inv-fg-def)

```

3.2 The definition

Finally, we can define the Free Group over a set of generators, and show that it is indeed a group.

definition *free-group* :: *'a set => ((bool * 'a) list) monoid (F₁)*

where

```

Fgens ≡ ⟨
  carrier = {l ∈ lists (UNIV × gens). canceled l },
  mult = λ x y. normalize (x @ y),

```


one = []
)

lemma *occurring-gens-in-element*:

$x \in \text{carrier } \mathcal{F}_{gens} \implies x \in \text{lists } (UNIV \times gens)$

by (*auto simp add:free-group-def*)

theorem *free-group-is-group*: group \mathcal{F}_{gens}

proof

fix $x y$

assume $x \in \text{carrier } \mathcal{F}_{gens}$ **hence** $x: x \in \text{lists } (UNIV \times gens)$ **by**
 (*rule occurring-gens-in-element*)

assume $y \in \text{carrier } \mathcal{F}_{gens}$ **hence** $y: y \in \text{lists } (UNIV \times gens)$ **by**
 (*rule occurring-gens-in-element*)

from x **and** y

have $x \otimes_{\mathcal{F}_{gens}} y \in \text{lists } (UNIV \times gens)$

by (*auto intro!: normalize-preserves-generators simp add:free-group-def append-in-lists-conv*)

thus $x \otimes_{\mathcal{F}_{gens}} y \in \text{carrier } \mathcal{F}_{gens}$

by (*auto simp add:free-group-def*)

next

fix $x y z$

have *cancels-to* $(x @ y)$ (*normalize* $(x @ (y::'a \text{ word-g-i}))$)

and *cancels-to* z ($z::'a \text{ word-g-i}$)

by *auto*

hence *normalize* (*normalize* $(x @ y) @ z$) = *normalize* $((x @ y) @ z)$

by (*rule normalize-append-cancel-to[THEN sym]*)

also

have $\dots = \text{normalize } (x @ (y @ z))$ **by** *auto*

also

have *cancels-to* $(y @ z)$ (*normalize* $(y @ (z::'a \text{ word-g-i}))$)

and *cancels-to* x ($x::'a \text{ word-g-i}$)

by *auto*

hence *normalize* $(x @ (y @ z)) = \text{normalize } (x @ \text{normalize } (y @ z))$

by $-(\text{rule normalize-append-cancel-to})$

finally

show $x \otimes_{\mathcal{F}_{gens}} y \otimes_{\mathcal{F}_{gens}} z =$

$x \otimes_{\mathcal{F}_{gens}} (y \otimes_{\mathcal{F}_{gens}} z)$

by (*auto simp add:free-group-def*)

next

show $1_{\mathcal{F}_{gens}} \in \text{carrier } \mathcal{F}_{gens}$

by (*auto simp add:free-group-def*)

next

fix x

assume $x \in \text{carrier } \mathcal{F}_{gens}$

thus $1_{\mathcal{F}_{gens}} \otimes_{\mathcal{F}_{gens}} x = x$

by (*auto simp add:free-group-def*)

```

next
  fix x
  assume x ∈ carrier  $\mathcal{F}_{gens}$ 
  thus  $x \otimes_{\mathcal{F}_{gens}} \mathbf{1}_{\mathcal{F}_{gens}} = x$ 
    by (auto simp add:free-group-def)
next
show carrier  $\mathcal{F}_{gens} \subseteq Units \mathcal{F}_{gens}$ 
proof (simp add:free-group-def Units-def, rule subsetI)
  fix x :: 'a word-g-i
  let ?x' = inv-fg x
  assume x ∈ {y ∈ lists (UNIV × gens). canceled y}
  hence ?x' ∈ lists (UNIV × gens) ∧ canceled ?x'
    by (auto elim:inv-fg-closure1 simp add:inv-fg-closure2)
  moreover
  have normalize (?x' @ x) = []
  and normalize (x @ ?x') = []
    by (auto simp add:inv-fg-cancel inv-fg-cancel2)
  ultimately
  have ∃ y. y ∈ lists (UNIV × gens) ∧
    canceled y ∧
    normalize (y @ x) = [] ∧ normalize (x @ y) = []
    by auto
  with ⟨x ∈ {y ∈ lists (UNIV × gens). canceled y}⟩
  show x ∈ {y ∈ lists (UNIV × gens). canceled y ∧
    (∃ x. x ∈ lists (UNIV × gens) ∧
    canceled x ∧
    normalize (x @ y) = [] ∧ normalize (y @ x) = [])}
    by auto
qed
qed

```

lemma *inv-is-inv-fg*[simp]:

$$x \in carrier \mathcal{F}_{gens} \implies inv_{\mathcal{F}_{gens}} x = inv\text{-}fg\ x$$

by (rule group.inv-equality, auto simp add:free-group-is-group, auto simp add: free-group-def inv-fg-cancel inv-fg-cancel2 inv-fg-closure1 inv-fg-closure2)

3.3 The universal property

Free Groups are important due to their universal property: Every map of the set of generators to another group can be extended uniquely to an homomorphism from the Free Group.

definition *insert* (ι)

$$\text{where } \iota\ g = [(False, g)]$$

lemma *insert-closed*:

$$g \in gens \implies \iota\ g \in carrier \mathcal{F}_{gens}$$

by (auto simp add:insert-def free-group-def)

definition (in group) *lift-gi*
 where $\text{lift-gi } f \text{ } gi = (\text{if fst } gi \text{ then inv } (f \text{ } (\text{snd } gi)) \text{ else } f \text{ } (\text{snd } gi))$

lemma (in group) *lift-gi-closed*:
 assumes $cl: f \in \text{gens} \rightarrow \text{carrier } G$
 and $\text{snd } gi \in \text{gens}$
 shows $\text{lift-gi } f \text{ } gi \in \text{carrier } G$
 using *assms* by (auto simp add:lift-gi-def)

definition (in group) *lift*
 where $\text{lift } f \text{ } w = m\text{-concat } (\text{map } (\text{lift-gi } f) \text{ } w)$

lemma (in group) *lift-nil[simp]*: $\text{lift } f \text{ } [] = \mathbf{1}$
 by (auto simp add:lift-def)

lemma (in group) *lift-closed[simp]*:
 assumes $cl: f \in \text{gens} \rightarrow \text{carrier } G$
 and $x \in \text{lists } (UNIV \times \text{gens})$
 shows $\text{lift } f \text{ } x \in \text{carrier } G$
proof –
 have $\text{set } (\text{map } (\text{lift-gi } f) \text{ } x) \subseteq \text{carrier } G$
 using $\langle x \in \text{lists } (UNIV \times \text{gens}) \rangle$
 by (auto simp add:lift-gi-closed[OF cl])
 thus $\text{lift } f \text{ } x \in \text{carrier } G$
 by (auto simp add:lift-def)
qed

lemma (in group) *lift-append[simp]*:
 assumes $cl: f \in \text{gens} \rightarrow \text{carrier } G$
 and $x \in \text{lists } (UNIV \times \text{gens})$
 and $y \in \text{lists } (UNIV \times \text{gens})$
 shows $\text{lift } f \text{ } (x @ y) = \text{lift } f \text{ } x \otimes \text{lift } f \text{ } y$
proof –
 from $\langle x \in \text{lists } (UNIV \times \text{gens}) \rangle$
 have $\text{set } (\text{map } \text{snd } x) \subseteq \text{gens}$ by auto
 hence $\text{set } (\text{map } (\text{lift-gi } f) \text{ } x) \subseteq \text{carrier } G$
 by (induct x)(auto simp add:lift-gi-closed[OF cl])
moreover
 from $\langle y \in \text{lists } (UNIV \times \text{gens}) \rangle$
 have $\text{set } (\text{map } \text{snd } y) \subseteq \text{gens}$ by auto
 hence $\text{set } (\text{map } (\text{lift-gi } f) \text{ } y) \subseteq \text{carrier } G$
 by (induct y)(auto simp add:lift-gi-closed[OF cl])
ultimately
 show $\text{lift } f \text{ } (x @ y) = \text{lift } f \text{ } x \otimes \text{lift } f \text{ } y$
 by (auto simp add:lift-def m-assoc simp del:set-map foldr-append)
qed

lemma (in group) *lift-cancels-to*:
 assumes *cancels-to* $x \text{ } y$

```

    and  $x \in \text{lists } (UNIV \times \text{gens})$ 
    and  $cl: f \in \text{gens} \rightarrow \text{carrier } G$ 
  shows  $\text{lift } f \ x = \text{lift } f \ y$ 
using assms
unfolding cancels-to-def
proof(induct rule:rtranclp-induct)
  case (step y z)
    from  $\langle \text{cancels-to-1}^{**} \ x \ y \rangle$ 
    and  $\langle x \in \text{lists } (UNIV \times \text{gens}) \rangle$ 
    have  $y \in \text{lists } (UNIV \times \text{gens})$ 
      by  $-(\text{rule } \text{cancels-to-preserves-generators}, \text{simp add: } \text{cancels-to-def})$ 
    hence  $\text{lift } f \ x = \text{lift } f \ y$ 
      using step by auto
    also
    from  $\langle \text{cancels-to-1} \ y \ z \rangle$ 
    obtain  $ys1 \ y1 \ y2 \ ys2$ 
      where  $y: y = ys1 \ @ \ y1 \ \# \ y2 \ \# \ ys2$ 
      and  $z = ys1 \ @ \ ys2$ 
      and canceling y1 y2
    by (rule cancels-to-1-unfold)
    have  $\text{lift } f \ y = \text{lift } f \ (ys1 \ @ \ [y1] \ @ \ [y2] \ @ \ ys2)$ 
      using y by simp
    also
    from y and cl and  $\langle y \in \text{lists } (UNIV \times \text{gens}) \rangle$ 
    have  $\text{lift } f \ (ys1 \ @ \ [y1] \ @ \ [y2] \ @ \ ys2)$ 
      =  $\text{lift } f \ ys1 \ \otimes \ (\text{lift } f \ [y1] \ \otimes \ \text{lift } f \ [y2]) \ \otimes \ \text{lift } f \ ys2$ 
      by (auto intro:lift-append[OF cl] simp del: append-Cons simp add:m-assoc
iff:lists-eq-set)
    also
    from cl[THEN funcset-image]
    and y and  $\langle y \in \text{lists } (UNIV \times \text{gens}) \rangle$ 
    and  $\langle \text{canceling } y1 \ y2 \rangle$ 
    have  $(\text{lift } f \ [y1] \ \otimes \ \text{lift } f \ [y2]) = \mathbf{1}$ 
      by (auto simp add:lift-def lift-gi-def canceling-def iff:lists-eq-set)
    hence  $\text{lift } f \ ys1 \ \otimes \ (\text{lift } f \ [y1] \ \otimes \ \text{lift } f \ [y2]) \ \otimes \ \text{lift } f \ ys2$ 
      =  $\text{lift } f \ ys1 \ \otimes \ \mathbf{1} \ \otimes \ \text{lift } f \ ys2$ 
      by simp
    also
    from y and  $\langle y \in \text{lists } (UNIV \times \text{gens}) \rangle$ 
    and cl
    have  $\text{lift } f \ ys1 \ \otimes \ \mathbf{1} \ \otimes \ \text{lift } f \ ys2 = \text{lift } f \ (ys1 \ @ \ ys2)$ 
      by (auto intro:lift-append iff:lists-eq-set)
    also
    from  $\langle z = ys1 \ @ \ ys2 \rangle$ 
    have  $\text{lift } f \ (ys1 \ @ \ ys2) = \text{lift } f \ z$  by simp
    finally show  $\text{lift } f \ x = \text{lift } f \ z$  .
qed auto

```

lemma (in group) *lift-is-hom*:

assumes $cl: f \in gens \rightarrow carrier\ G$
shows $lift\ f \in hom\ \mathcal{F}_{gens}\ G$
proof –
{
 fix x
 assume $x \in carrier\ \mathcal{F}_{gens}$
 hence $x \in lists\ (UNIV \times gens)$
 unfolding $free\ group\ def$ **by** $simp$
 hence $lift\ f\ x \in carrier\ G$
 by $(induct\ x, auto\ simp\ add:lift-def\ lift-gi-closed[OF\ cl])$
}
moreover
{ **fix** x
 assume $x \in carrier\ \mathcal{F}_{gens}$
 fix y
 assume $y \in carrier\ \mathcal{F}_{gens}$

 from $\langle x \in carrier\ \mathcal{F}_{gens} \rangle$ **and** $\langle y \in carrier\ \mathcal{F}_{gens} \rangle$
 have $x \in lists\ (UNIV \times gens)$ **and** $y \in lists\ (UNIV \times gens)$
 by $(auto\ simp\ add:free-group-def)$

 have $cancels\ to\ (x\ @\ y)\ (normalize\ (x\ @\ y))$ **by** $simp$
 from $\langle x \in lists\ (UNIV \times gens) \rangle$ **and** $\langle y \in lists\ (UNIV \times gens) \rangle$
 and $lift\ cancels\ to[THEN\ sym, OF\ \langle cancels\ to\ (x\ @\ y)\ (normalize\ (x\ @\ y)) \rangle]$
and cl
 have $lift\ f\ (x\ \otimes_{\mathcal{F}_{gens}}\ y) = lift\ f\ (x\ @\ y)$
 by $(auto\ simp\ add:free-group-def\ iff:lists-eq-set)$
 also
 from $\langle x \in lists\ (UNIV \times gens) \rangle$ **and** $\langle y \in lists\ (UNIV \times gens) \rangle$ **and** cl
 have $lift\ f\ (x\ @\ y) = lift\ f\ x\ \otimes\ lift\ f\ y$
 by $simp$
 finally
 have $lift\ f\ (x\ \otimes_{\mathcal{F}_{gens}}\ y) = lift\ f\ x\ \otimes\ lift\ f\ y .$
}
ultimately
show $lift\ f \in hom\ \mathcal{F}_{gens}\ G$
 by $auto$
qed

lemma $gens\ span\ free\ group:$
shows $\langle \iota\ 'gens \rangle_{\mathcal{F}_{gens}} = carrier\ \mathcal{F}_{gens}$
proof
 interpret $group\ \mathcal{F}_{gens}$ **by** $(rule\ free\ group\ is\ group)$
 show $\langle \iota\ 'gens \rangle_{\mathcal{F}_{gens}} \subseteq carrier\ \mathcal{F}_{gens}$
 by $(rule\ gen\ span\ closed, auto\ simp\ add:insert-def\ free-group-def)$

 show $carrier\ \mathcal{F}_{gens} \subseteq \langle \iota\ 'gens \rangle_{\mathcal{F}_{gens}}$
 proof
 fix x

```

show  $x \in \text{carrier } \mathcal{F}_{gens} \implies x \in \langle \iota \text{ ' gens} \rangle_{\mathcal{F}_{gens}}$ 
proof(induct x)
case Nil
  have  $\text{one } \mathcal{F}_{gens} \in \langle \iota \text{ ' gens} \rangle_{\mathcal{F}_{gens}}$ 
    by simp
  thus  $[] \in \langle \iota \text{ ' gens} \rangle_{\mathcal{F}_{gens}}$ 
    by (simp add:free-group-def)
next
case (Cons a x)
  from  $\langle a \# x \in \text{carrier } \mathcal{F}_{gens} \rangle$ 
  have  $x \in \text{carrier } \mathcal{F}_{gens}$ 
    by (auto intro:cons-canceled simp add:free-group-def)
  hence  $x \in \langle \iota \text{ ' gens} \rangle_{\mathcal{F}_{gens}}$ 
    using Cons by simp
  moreover

  from  $\langle a \# x \in \text{carrier } \mathcal{F}_{gens} \rangle$ 
  have  $\text{snd } a \in \text{gens}$ 
    by (auto simp add:free-group-def)
  hence  $\text{isa: } \iota (\text{snd } a) \in \langle \iota \text{ ' gens} \rangle_{\mathcal{F}_{gens}}$ 
    by (auto simp add:insert-def intro:gen-gens)
  have  $[a] \in \langle \iota \text{ ' gens} \rangle_{\mathcal{F}_{gens}}$ 
  proof(cases fst a)
    case False
      hence  $[a] = \iota (\text{snd } a)$  by (cases a, auto simp add:insert-def)
      with isa show  $[a] \in \langle \iota \text{ ' gens} \rangle_{\mathcal{F}_{gens}}$  by simp
    next
    case True
      from  $\langle \text{snd } a \in \text{gens} \rangle$ 
      have  $\iota (\text{snd } a) \in \text{carrier } \mathcal{F}_{gens}$ 
        by (auto simp add:free-group-def insert-def)
      with True
      have  $[a] = \text{inv}_{\mathcal{F}_{gens}} (\iota (\text{snd } a))$ 
        by (cases a, auto simp add:insert-def inv-fg-def inv1-def)
      moreover
      from isa
      have  $\text{inv}_{\mathcal{F}_{gens}} (\iota (\text{snd } a)) \in \langle \iota \text{ ' gens} \rangle_{\mathcal{F}_{gens}}$ 
        by (auto intro:gen-inv)
      ultimately
      show  $[a] \in \langle \iota \text{ ' gens} \rangle_{\mathcal{F}_{gens}}$ 
        by simp
  qed
  ultimately
  have  $\text{mult } \mathcal{F}_{gens} [a] x \in \langle \iota \text{ ' gens} \rangle_{\mathcal{F}_{gens}}$ 
    by (auto intro:gen-mult)
  with
   $\langle a \# x \in \text{carrier } \mathcal{F}_{gens} \rangle$ 
  show  $a \# x \in \langle \iota \text{ ' gens} \rangle_{\mathcal{F}_{gens}}$  by (simp add:free-group-def)

```

```

    qed
  qed
qed

lemma (in group) lift-is-unique:
  assumes group G
  and cl: f ∈ gens → carrier G
  and h ∈ hom ℱgens G
  and ∀ g ∈ gens. h (ι g) = f g
  shows ∀ x ∈ carrier ℱgens. h x = lift f x
  unfolding gens-span-free-group[THEN sym]
  proof(rule hom-unique-on-span[of ℱgens G])
    show group ℱgens by (rule free-group-is-group)
  next
    show group G by fact
  next
    show ι ' gens ⊆ carrier ℱgens
      by(auto intro:insert-closed)
  next
    show h ∈ hom ℱgens G by fact
  next
    show lift f ∈ hom ℱgens G by (rule lift-is-hom[OF cl])
  next
    from ⟨∀ g ∈ gens. h (ι g) = f g⟩ and cl[THEN funcset-image]
    show ∀ g ∈ ι ' gens. h g = lift f g
      by(auto simp add:insert-def lift-def lift-gi-def)
  qed
end

```

4 The Unit Group

```

theory UnitGroup
imports
  HOL-Algebra.Group
  Generators
begin

```

There is, up to isomorphisms, only one group with one element.

definition *unit-group* :: *unit monoid*

where

```

unit-group ≡ ⟨
  carrier = UNIV,
  mult = λ x y. (),
  one = ()
⟩

```

theorem *unit-group-is-group*: *group unit-group*
 by (rule groupI, auto simp add:unit-group-def)

```

theorem (in group) unit-group-unique:
  assumes card (carrier G) = 1
  shows  $\exists h. h \in iso\ G\ unit\text{-}group$ 
proof –
  from assms obtain x where carrier G = {x} by (auto dest: card-eq-SucD)
  hence ( $\lambda x. ()$ )  $\in iso\ G\ unit\text{-}group$ 
  by –(rule group-isoI, auto simp add:unit-group-is-group is-group, simp add:unit-group-def)
  thus ?thesis by auto
qed

end
theory C2
imports HOL-Algebra.Group
begin

```

5 The group C2

The two-element group is defined over the set of boolean values. This allows to use the equality of boolean values as the group operation.

```

definition C2
  where C2 = ( $\langle carrier = UNIV, mult = (=), one = True \rangle$ )

```

```

lemma [simp]: ( $\otimes_{C2}$ ) = (=)
  unfolding C2-def by simp

```

```

lemma [simp]:  $1_{C2} = True$ 
  unfolding C2-def by simp

```

```

lemma [simp]: carrier C2 = UNIV
  unfolding C2-def by simp

```

```

lemma C2-is-group: group C2
  unfolding C2-def
  by (rule groupI, auto simp add:Units-def)

```

```

end

```

6 Isomorphisms of Free Groups

```

theory Isomorphisms
imports
  UnitGroup
  HOL-Algebra.IntRing
  FreeGroups
  C2
  HOL-Cardinals.Cardinal-Order-Relation
begin

```


6.1 The Free Group over the empty set

The Free Group over an empty set of generators is isomorphic to the trivial group.

lemma *free-group-over-empty-set*: $\exists h. h \in \text{iso } \mathcal{F}_{\{\}} \text{ unit-group}$

proof(*rule group.unit-group-unique*)

show *group* $\mathcal{F}_{\{\}}$ **by** (*rule free-group-is-group*)

next

have *carrier* $\mathcal{F}_{\{\}::'a \text{ set}}$ = $\{\emptyset\}$

by (*auto simp add:free-group-def*)

thus *card* (*carrier* $\mathcal{F}_{\{\}::'a \text{ set}}$) = 1

by *simp*

qed

6.2 The Free Group over one generator

The Free Group over one generator is isomorphic to the free abelian group over one element, also known as the integers.

abbreviation *int-group*

where *int-group* \equiv (\langle *carrier* = *carrier* \mathcal{Z} , *monoid.mult* = (+), *one* = 0::*int* \rangle)

lemma *replicate-set-eq*[*simp*]: $\forall x \in \text{set } xs. x = y \implies xs = \text{replicate } (\text{length } xs) y$

by(*induct xs*)*auto*

lemma *int-group-gen-by-one*: $\langle\{1\}\rangle_{\text{int-group}} = \text{carrier } \text{int-group}$

proof

show $\langle\{1\}\rangle_{\text{int-group}} \subseteq \text{carrier } \text{int-group}$

by *auto*

show *carrier* *int-group* $\subseteq \langle\{1\}\rangle_{\text{int-group}}$

proof

interpret *int*: *group* *int-group*

using *int.a-group* **by** *auto*

fix *x*

have *plus1*: $1 \in \langle\{1\}\rangle_{\text{int-group}}$

by (*auto intro:gen-span.gen-gens*)

hence *inv*_{*int-group*} 1 $\in \langle\{1\}\rangle_{\text{int-group}}$

by (*auto intro:gen-span.gen-inv*)

moreover

have $-1 = \text{inv}_{\text{int-group}} 1$

by (*rule sym, rule int.inv-equality*) *simp-all*

ultimately

have *minus1*: $-1 \in \langle\{1\}\rangle_{\text{int-group}}$

by (*simp*)

show $x \in \langle\{1::\text{int}\}\rangle_{\text{int-group}}$

proof(*induct x rule:int-induct[of - 0::int]*)

case *base*

have $1_{\text{int-group}} \in \langle\{1::\text{int}\}\rangle_{\text{int-group}}$

```

    by (rule gen-span.gen-one)
  thus  $0 \in \langle \{1\} \rangle_{int\text{-group}}$ 
    by simp
next
case (step1 i)
  from  $\langle i \in \langle \{1\} \rangle_{int\text{-group}} \rangle$  and plus1
  have  $i \otimes_{int\text{-group}} 1 \in \langle \{1\} \rangle_{int\text{-group}}$ 
    by (rule gen-span.gen-mult)
  thus  $i + 1 \in \langle \{1\} \rangle_{int\text{-group}}$  by simp
next
case (step2 i)
  from  $\langle i \in \langle \{1\} \rangle_{int\text{-group}} \rangle$  and minus1
  have  $i \otimes_{int\text{-group}} -1 \in \langle \{1\} \rangle_{int\text{-group}}$ 
    by (rule gen-span.gen-mult)
  thus  $i - 1 \in \langle \{1\} \rangle_{int\text{-group}}$ 
    by simp
qed
qed
qed

lemma free-group-over-one-gen:  $\exists h. h \in iso \mathcal{F}_{\{()\}} int\text{-group}$ 
proof -
  interpret int: group int-group
  using int.a-group by auto
  define f :: unit  $\Rightarrow$  int where f x = 1 for x
  have f  $\in \{()\} \rightarrow carrier\ int\text{-group}$ 
    by auto
  hence int.lift f  $\in hom \mathcal{F}_{\{()\}} int\text{-group}$ 
    by (rule int.lift-is-hom)
  then
  interpret hom: group-hom  $\mathcal{F}_{\{()\}} int\text{-group} int.lift\ f$ 
  unfolding group-hom-def group-hom-axioms-def
  using int.a-group by (auto intro: free-group-is-group)

{
  fix x
  assume x  $\in carrier \mathcal{F}_{\{()\}}$ 
  hence canceled x by (auto simp add: free-group-def)
  assume int.lift f x = (0::int)
  have x = []
  proof (rule ccontr)
    assume x  $\neq []$ 
    then obtain a and xs where x = a # xs by (cases x, auto)
    hence length (takeWhile ( $\lambda y. y = a$ ) x) > 0 by auto
    then obtain i where i: length (takeWhile ( $\lambda y. y = a$ ) x) = Suc i
      by (cases length (takeWhile ( $\lambda y. y = a$ ) x), auto)
    have Suc i  $\geq$  length x
    proof (rule ccontr)
      assume  $\neg$  length x  $\leq$  Suc i

```

hence $\text{length } (\text{takeWhile } (\lambda y. y = a) x) < \text{length } x$ **using** i **by** *simp*
hence $\neg (\lambda y. y = a) (x ! \text{length } (\text{takeWhile } (\lambda y. y = a) x))$
by (*rule nth-length-takeWhile*)
hence $\neg (\lambda y. y = a) (x ! \text{Suc } i)$ **using** i **by** *simp*
hence $\text{fst } (x ! \text{Suc } i) \neq \text{fst } a$ **by** (*cases x ! Suc i, cases a, auto*)
moreover
{
have $\text{takeWhile } (\lambda y. y = a) x ! i = x ! i$
using i **by** (*auto intro: takeWhile-nth*)
moreover
have $(\text{takeWhile } (\lambda y. y = a) x) ! i \in \text{set } (\text{takeWhile } (\lambda y. y = a) x)$
using i **by** *auto*
ultimately
have $(\lambda y. y = a) (x ! i)$
by (*auto dest:set-takeWhileD*)
}
hence $\text{fst } (x ! i) = \text{fst } a$ **by** *auto*
moreover
have $\text{snd } (x ! i) = \text{snd } (x ! \text{Suc } i)$ **by** *simp*
ultimately
have $\text{canceling } (x ! i) (x ! \text{Suc } i)$ **unfolding** *canceling-def* **by** *auto*
hence $\text{cancels-to-1-at } i x$ (*cancel-at i x*)
using $\langle \neg \text{length } x \leq \text{Suc } i \rangle$ **unfolding** *cancels-to-1-at-def*
by (*auto simp add:length-takeWhile-le*)
hence $\text{cancels-to-1 } x$ (*cancel-at i x*) **unfolding** *cancels-to-1-def* **by** *auto*
hence $\neg \text{canceled } x$ **unfolding** *canceled-def* **by** *auto*
thus *False* **using** $\langle \text{canceled } x \rangle$ **by** *contradiction*
qed
hence $\text{length } (\text{takeWhile } (\lambda y. y = a) x) = \text{length } x$
using i [*THEN sym*] **by** (*auto dest:le-antisym simp add:length-takeWhile-le*)
hence $\text{takeWhile } (\lambda y. y = a) x = x$
by (*subst takeWhile-eq-take, simp*)
moreover
have $\forall y \in \text{set } (\text{takeWhile } (\lambda y. y = a) x). y = a$
by (*auto dest: set-takeWhileD*)
ultimately
have $\forall y \in \text{set } x. y = a$ **by** *auto*
hence $x = \text{replicate } (\text{length } x) a$ **by** *simp*
hence $\text{int.lift } f x = \text{int.lift } f (\text{replicate } (\text{length } x) a)$ **by** *simp*
also have $\dots = \text{pow int-group } (\text{int.lift-gi } f a) (\text{length } x)$
apply (*induct x*)
using *local.int.nat-pow-Suc local.int.nat-pow-0*
apply (*auto simp: int.lift-def [simplified]*)
done
also have $\dots = (\text{int.lift-gi } f a) * \text{int } (\text{length } x)$
apply (*induct x*)
using *local.int.nat-pow-Suc local.int.nat-pow-0*
by (*auto simp: int-distrib*)
finally have $\dots = 0$ **using** $\langle \text{int.lift } f x = 0 \rangle$ **by** *simp*

```

    hence nat (abs (group.lift-gi int-group f a * int (length x))) = 0 by simp
    hence nat (abs (group.lift-gi int-group f a)) * length x = 0 by simp
    hence nat (abs (group.lift-gi int-group f a)) = 0
      using ⟨x ≠ []⟩ by auto
    moreover
    have inv_int-group 1 = -1
      using int.inv-equality by auto
    hence abs (group.lift-gi int-group f a) = 1
      using int.is-group
      by(auto simp add: group.lift-gi-def f-def)
    ultimately
    show False by simp
  qed
}
hence ∀ x ∈ carrier  $\mathcal{F}_{\{()\}}$ . int.lift f x =  $\mathbf{1}_{int\text{-group}}$   $\longrightarrow$  x =  $\mathbf{1}_{\mathcal{F}_{\{()\}}}$ 
  by (auto simp add: free-group-def)
moreover
{
  have carrier  $\mathcal{F}_{\{()\}}$  = ⟨insert '{()}⟩ $\mathcal{F}_{\{()\}}$ 
    by (rule gens-span-free-group[THEN sym])
  moreover
  have carrier int-group = ⟨{1}⟩int-group
    by (rule int-group-gen-by-one[THEN sym])
  moreover
  have int.lift f 'insert '{()}' = {1}
    by (auto simp add: int.lift-def [simplified] insert-def f-def int.lift-gi-def
      [simplified])
  moreover
  have int.lift f '⟨insert '{()}⟩ $\mathcal{F}_{\{()\}}$  = ⟨int.lift f '(insert '{()}')⟩int-group
    by (rule hom.hom-span, auto intro: insert-closed)
  ultimately
  have int.lift f 'carrier  $\mathcal{F}_{\{()\}}$  = carrier int-group
    by simp
}
ultimately
have int.lift f ∈ iso  $\mathcal{F}_{\{()\}}$  int-group
  using ⟨int.lift f ∈ hom  $\mathcal{F}_{\{()\}}$  int-group⟩
  using hom.hom-mult int.is-group
  by (auto intro: group-isoI simp add: free-group-is-group)
thus ?thesis by auto
qed

```

6.3 Free Groups over isomorphic sets of generators

Free Groups are isomorphic if their set of generators are isomorphic.

definition lift-generator-function :: ('a ⇒ 'b) ⇒ (bool × 'a) list ⇒ (bool × 'b) list
where lift-generator-function f = map (map-prod id f)

```

theorem isomorphic-free-groups:
  assumes bij-betw f gens1 gens2
  shows lift-generator-function f ∈ iso Fgens1 Fgens2
unfolding lift-generator-function-def
proof(rule group-isoI)
  show  $\forall x \in \text{carrier } \mathcal{F}_{\text{gens1}}.$ 
     $\text{map } (\text{map-prod id } f) x = \mathbf{1}_{\mathcal{F}_{\text{gens2}}} \longrightarrow x = \mathbf{1}_{\mathcal{F}_{\text{gens1}}}$ 
    by(auto simp add:free-group-def)
next
from  $\langle \text{bij-betw } f \text{ gens1 gens2} \rangle$  have inj-on f gens1 by (auto simp:bij-betw-def)
show  $\text{map } (\text{map-prod id } f) ` \text{carrier } \mathcal{F}_{\text{gens1}} = \text{carrier } \mathcal{F}_{\text{gens2}}$ 
proof(rule Set.set-eqI,rule iffI)
from  $\langle \text{bij-betw } f \text{ gens1 gens2} \rangle$  have  $f ` \text{gens1} = \text{gens2}$  by (auto simp:bij-betw-def)
  fix  $x :: (\text{bool} \times 'b)$  list
  assume  $x \in \text{image } (\text{map } (\text{map-prod id } f)) (\text{carrier } \mathcal{F}_{\text{gens1}})$ 
  then obtain  $y :: (\text{bool} \times 'a)$  list where  $x = \text{map } (\text{map-prod id } f) y$ 
    and  $y \in \text{carrier } \mathcal{F}_{\text{gens1}}$  by auto
  from  $\langle y \in \text{carrier } \mathcal{F}_{\text{gens1}} \rangle$ 
have canceled y and  $y \in \text{lists}(UNIV \times \text{gens1})$  by (auto simp add:free-group-def)

  from  $\langle y \in \text{lists}(UNIV \times \text{gens1}) \rangle$ 
    and  $\langle x = \text{map } (\text{map-prod id } f) y \rangle$ 
    and  $\langle \text{image } f \text{ gens1} = \text{gens2} \rangle$ 
  have  $x \in \text{lists}(UNIV \times \text{gens2})$ 
    by (auto iff:lists-eq-set)
  moreover

  from  $\langle x = \text{map } (\text{map-prod id } f) y \rangle$ 
    and  $\langle y \in \text{lists}(UNIV \times \text{gens1}) \rangle$ 
    and  $\langle \text{canceled } y \rangle$ 
    and  $\langle \text{inj-on } f \text{ gens1} \rangle$ 
  have canceled x
    by (auto intro!:rename-gens-canceled subset-inj-on[OF \langle inj-on f gens1 \rangle])
iff:lists-eq-set)
  ultimately
  show  $x \in \text{carrier } \mathcal{F}_{\text{gens2}}$  by (simp add:free-group-def)
next
fix  $x$ 
assume  $x \in \text{carrier } \mathcal{F}_{\text{gens2}}$ 
hence canceled x and  $x \in \text{lists}(UNIV \times \text{gens2})$ 
  unfolding free-group-def by auto
define  $y$  where  $y = \text{map } (\text{map-prod id } (\text{the-inv-into } \text{gens1 } f)) x$ 
have  $\text{map } (\text{map-prod id } f) y =$ 
   $\text{map } (\text{map-prod id } f) (\text{map } (\text{map-prod id } (\text{the-inv-into } \text{gens1 } f)) x)$ 
  by (simp add:y-def)
also have  $\dots = \text{map } (\text{map-prod id } f \circ \text{map-prod id } (\text{the-inv-into } \text{gens1 } f)) x$ 
  by simp
also have  $\dots = \text{map } (\text{map-prod id } (f \circ \text{the-inv-into } \text{gens1 } f)) x$ 

```

```

    by auto
  also have ... = map id x
  proof(rule map-ext, rule impI)
    fix xa :: bool × 'b
    assume xa ∈ set x
    from ⟨x ∈ lists (UNIV × gens2)⟩
    have set (map snd x) ⊆ gens2 by auto
    hence snd ' set x ⊆ gens2 by (simp add: set-map)
    with ⟨xa ∈ set x⟩ have snd xa ∈ gens2 by auto
    with ⟨bij-betw f gens1 gens2⟩ have snd xa ∈ f'gens1
      by (auto simp add: bij-betw-def)

    have map-prod id (f ∘ the-inv-into gens1 f) xa
      = map-prod id (f ∘ the-inv-into gens1 f) (fst xa, snd xa) by simp
    also have ... = (fst xa, f (the-inv-into gens1 f (snd xa)))
      by (auto simp del:prod.collapse)
    also
    from ⟨snd xa ∈ image f gens1⟩ and ⟨inj-on f gens1⟩
    have ... = (fst xa, snd xa)
      by (auto elim:f-the-inv-into-f simp del:prod.collapse)
    also have ... = id xa by simp
    finally show map-prod id (f ∘ the-inv-into gens1 f) xa = id xa.
  qed
  also have ... = x unfolding id-def by auto
  finally have map (map-prod id f) y = x.
  moreover
  {
    from ⟨bij-betw f gens1 gens2⟩
    have bij-betw (the-inv-into gens1 f) gens2 gens1 by (rule bij-betw-the-inv-into)
    hence inj-on (the-inv-into gens1 f) gens2 by (rule bij-betw-imp-inj-on)

    with ⟨canceled x⟩
    and ⟨x ∈ lists (UNIV × gens2)⟩
    have canceled y
      by (auto intro!:rename-gens-canceled[OF subset-inj-on] simp add:y-def)
    moreover
    {
      from ⟨bij-betw (the-inv-into gens1 f) gens2 gens1⟩
      and ⟨x ∈ lists (UNIV × gens2)⟩
      have y ∈ lists (UNIV × gens1)
        unfolding y-def and bij-betw-def
        by (auto iff:lists-eq-set dest!:subsetD)
    }
    ultimately
    have y ∈ carrier  $\mathcal{F}_{gens1}$  by (simp add:free-group-def)
  }
  ultimately
  show x ∈ map (map-prod id f) ' carrier  $\mathcal{F}_{gens1}$  by auto
  qed

```

```

next
  from ⟨bij-betw f gens1 gens2⟩ have inj-on f gens1 by (auto simp:bij-betw-def)
  {
  fix x
  assume x ∈ carrier  $\mathcal{F}_{gens1}$ 
  fix y
  assume y ∈ carrier  $\mathcal{F}_{gens1}$ 

  from ⟨x ∈ carrier  $\mathcal{F}_{gens1}$ ⟩ and ⟨y ∈ carrier  $\mathcal{F}_{gens1}$ ⟩
  have x ∈ lists( $UNIV \times gens1$ ) and y ∈ lists( $UNIV \times gens1$ )
    by (auto simp add:occurring-gens-in-element)

  have map (map-prod id f) (x  $\otimes_{\mathcal{F}_{gens1}}$  y)
    = map (map-prod id f) (normalize (x@y)) by (simp add:free-group-def)
  also
  from ⟨x ∈ lists( $UNIV \times gens1$ )⟩ and ⟨y ∈ lists( $UNIV \times gens1$ )⟩
    and ⟨inj-on f gens1⟩
  have ... = normalize (map (map-prod id f) (x@y))
    by -(rule rename-gens-normalize[THEN sym],
        auto intro!: subset-inj-on[OF ⟨inj-on f gens1⟩] iff:lists-eq-set)
  also have ... = normalize (map (map-prod id f) x @ map (map-prod id f) y)
    by (auto)
  also have ... = map (map-prod id f) x  $\otimes_{\mathcal{F}_{gens2}}$  map (map-prod id f) y
    by (simp add:free-group-def)
  finally have map (map-prod id f) (x  $\otimes_{\mathcal{F}_{gens1}}$  y) =
    map (map-prod id f) x  $\otimes_{\mathcal{F}_{gens2}}$  map (map-prod id f) y.
  }
  thus  $\forall x \in \text{carrier } \mathcal{F}_{gens1}. \forall y \in \text{carrier } \mathcal{F}_{gens1}. \text{map (map-prod id f) (x } \otimes_{\mathcal{F}_{gens1}} \text{ y) =}$ 
    map (map-prod id f) x  $\otimes_{\mathcal{F}_{gens2}}$  map (map-prod id f) y
    by auto
qed (auto intro: free-group-is-group)

```

6.4 Bases of isomorphic free groups

Isomorphic free groups have bases of same cardinality. The proof is very different for infinite bases and for finite bases.

The proof for the finite case uses the set of homomorphisms from the free group to the group with two elements, as suggested by Christian Sievers. The definition of *hom* is not suitable for proofs about the cardinality of that set, as its definition does not require extensionality. This is amended by the following definition:

definition *homr*
 where $\text{homr } G H = \{h. h \in \text{hom } G H \wedge h \in \text{extensional (carrier } G)\}$

```

lemma (in group-hom) restrict-hom[intro!]:
  shows restrict h (carrier G) ∈ homr G H
  unfolding homr-def and hom-def
  by (auto)

lemma hom-F-C2-Powerset:
  ∃ f. bij-betw f (Pow X) (homr (FX) C2)
proof
  interpret F: group FX by (rule free-group-is-group)
  interpret C2: group C2 by (rule C2-is-group)
  let ?f = λS . restrict (C2.lift (λx. x ∈ S)) (carrier FX)
  let ?f' = λh . X ∩ Collect(h ∘ insert)
  show bij-betw ?f (Pow X) (homr (FX) C2)
  proof(induct rule: bij-betwI[of ?f - - ?f'])
  case 1 show ?case
    proof
    fix S assume S ∈ Pow X
    interpret h: group-hom FX C2 C2.lift (λx. x ∈ S)
      by unfold-locales (auto intro: C2.lift-is-hom)
    show ?f S ∈ homr FX C2
      by (rule h.restrict-hom)
    qed
  next
  case 2 show ?case by auto next
  case (3 S) show ?case
    proof (induct rule: Set.set-eqI)
    case (1 x) show ?case
      proof(cases x ∈ X)
      case True thus ?thesis using insert-closed[of x X]
        by (auto simp add:insert-def C2.lift-def C2.lift-gi-def)
      next case False thus ?thesis using 3 by auto
    qed
  qed
  next
  case (4 h)
    hence hom: h ∈ hom FX C2
      and extn: h ∈ extensional (carrier FX)
      unfolding homr-def by auto
    have ∀x ∈ carrier FX . h x = group.lift C2 (λz. z ∈ X & (h ∘ FreeGroups.insert)
z) x
      by (rule C2.lift-is-unique[OF C2-is-group - hom, of (λz. z ∈ X & (h ∘ Free-
Groups.insert) z)],
        auto)
    thus ?case
      by -(rule extensionalityI[OF restrict-extensional extn], auto)
    qed
  qed

```



```

lemma group-iso-betw-hom:
  assumes group G1 and group G2
    and iso: i ∈ iso G1 G2
  shows ∃ f . bij-betw f (homr G2 H) (homr G1 H)
proof -
  interpret G2: group G2 by (rule ‹group G2›)
  let ?i' = restrict (inv-into (carrier G1) i) (carrier G2)
  have inv-into (carrier G1) i ∈ iso G2 G1
    by (simp add: ‹group G1› group.iso-set-sym iso)
  hence iso': ?i' ∈ iso G2 G1
    by (auto simp add: Group.iso-def hom-def G2.m-closed)
  show ?thesis
  proof(rule, induct rule: bij-betwI[of (λh. compose (carrier G1) h i) - - (λh.
compose (carrier G2) h ?i')])
    case 1
      show ?case
      proof
        fix h assume h ∈ homr G2 H
        hence compose (carrier G1) h i ∈ hom G1 H
          using iso
        by (auto intro: group.hom-compose[OF ‹group G1›, of - G2] simp add: Group.iso-def
homr-def)
        thus compose (carrier G1) h i ∈ homr G1 H
          unfolding homr-def by simp
        qed
      next
      case 2
        show ?case
        proof
          fix h assume h ∈ homr G1 H
          hence compose (carrier G2) h ?i' ∈ hom G2 H
            using iso'
          by (auto intro: group.hom-compose[OF ‹group G2›, of - G1] simp add: Group.iso-def
homr-def)
          thus compose (carrier G2) h ?i' ∈ homr G2 H
            unfolding homr-def by simp
          qed
        next
        case (3 x)
          hence compose (carrier G2) (compose (carrier G1) x i) ?i'
            = compose (carrier G2) x (compose (carrier G2) i ?i')
            using iso iso'
          by (auto intro: compose-assoc[THEN sym] simp add: Group.iso-def hom-def
homr-def)
          also have ... = compose (carrier G2) x (λy∈carrier G2. y)
            using iso
          by (subst compose-id-inv-into, auto simp add: Group.iso-def hom-def bij-betw-def)
          also have ... = x
            using 3

```

```

    by (auto intro: compose-Id simp add: homr-def)
  finally
  show ?case .
next
case (4 y)
  hence compose (carrier G1) (compose (carrier G2) y ?i') i
    = compose (carrier G1) y (compose (carrier G1) ?i' i)
  using iso iso'
  by (auto intro: compose-assoc[THEN sym] simp add: Group.iso-def hom-def
homr-def)
  also have ... = compose (carrier G1) y ( $\lambda x \in \text{carrier } G1. x$ )
  using iso
  by (subst compose-inv-into-id, auto simp add: Group.iso-def hom-def bij-betw-def)
  also have ... = y
  using 4
  by (auto intro: compose-Id simp add: homr-def)
  finally
  show ?case .
qed
qed

```

lemma *isomorphic-free-groups-bases-finite:*

```

  assumes iso:  $i \in \text{iso } \mathcal{F}_X \mathcal{F}_Y$ 
    and finite: finite X
  shows  $\exists f. \text{bij-betw } f X Y$ 
proof -
  obtain f
    where bij-betw f (homr  $\mathcal{F}_Y C2$ ) (homr  $\mathcal{F}_X C2$ )
    using group-iso-betw-hom[OF free-group-is-group free-group-is-group iso]
    by auto
  moreover
  obtain g'
    where bij-betw g' (Pow X) (homr ( $\mathcal{F}_X$ ) C2)
    using hom-F-C2-Powerset by auto
  then obtain g
    where bij-betw g (homr ( $\mathcal{F}_X$ ) C2) (Pow X)
    by (auto intro: bij-betw-inv-into)
  moreover
  obtain h
    where bij-betw h (Pow Y) (homr ( $\mathcal{F}_Y$ ) C2)
    using hom-F-C2-Powerset by auto
  ultimately
  have bij-betw (g  $\circ$  f  $\circ$  h) (Pow Y) (Pow X)
    by (auto intro: bij-betw-trans)
  hence eq-card: card (Pow Y) = card (Pow X)
    by (rule bij-betw-same-card)
  with finite
  have finite (Pow Y)
    by -(rule card-ge-0-finite, auto simp add: card-Pow)

```

hence *finite'*: *finite Y* **by** *simp*
with *eq-card finite*
have $\text{card } X = \text{card } Y$
by (*auto simp add:card-Pow*)
with *finite finite'*
show *?thesis*
by (*rule finite-same-card-bij*)
qed

The proof for the infinite case is trivial once the fact that the free group over an infinite set has the same cardinality is established.

lemma *free-group-card-infinite*:

assumes $\neg \text{finite } X$
shows $|X| =_o |\text{carrier } \mathcal{F}_X|$
proof –
have *inj-on insert X*
by (*rule inj-onI*) (*auto simp add: insert-def*)
moreover **have** *insert ' X \subseteq carrier \mathcal{F}_X*
by (*auto intro: insert-closed*)
ultimately **have** $\exists f. \text{inj-on } f X \wedge f ' X \subseteq \text{carrier } \mathcal{F}_X$
by *auto*
then **have** $|X| \leq_o |\text{carrier } \mathcal{F}_X|$
by (*simp add: card-of-ordLeq*)
moreover
have $|\text{carrier } \mathcal{F}_X| \leq_o |\text{lists } ((\text{UNIV}::\text{bool set}) \times X)|$
by (*auto intro!: card-of-mono1 simp add: free-group-def*)
moreover
have $|\text{lists } ((\text{UNIV}::\text{bool set}) \times X)| =_o |(\text{UNIV}::\text{bool set}) \times X|$
using $\langle \neg \text{finite } X \rangle$
by (*auto intro: card-of-lists-infinite dest!: finite-cartesian-productD2*)
moreover
have $|(\text{UNIV}::\text{bool set}) \times X| =_o |X|$
using $\langle \neg \text{finite } X \rangle$
by (*auto intro: card-of-Times-infinite[OF - - ordLess-imp-ordLeq[OF finite-ordLess-infinite2], THEN conjunct2]*)
ultimately
show $|X| =_o |\text{carrier } \mathcal{F}_X|$
by (*subst ordIso-iff-ordLeq, auto intro: ord-trans*)
qed

theorem *isomorphic-free-groups-bases*:

assumes *iso: i \in iso $\mathcal{F}_X \mathcal{F}_Y$*
shows $\exists f. \text{bij-betw } f X Y$
proof(*cases finite X*)
case *True*
thus *?thesis* **using** *iso* **by** –(*rule isomorphic-free-groups-bases-finite*)
next
case *False* **show** *?thesis*

```

proof(cases finite Y)
case True
from iso obtain i' where i' ∈ iso  $\mathcal{F}_Y \mathcal{F}_X$ 
  using free-group-is-group group.iso-set-sym by blast
with ⟨finite Y⟩
have ∃f. bij-betw f Y X by -(rule isomorphic-free-groups-bases-finite)
thus ∃f. bij-betw f X Y by (auto intro: bij-betw-the-inv-into) next
case False
from ⟨¬ finite X⟩ have |X| =o |carrier  $\mathcal{F}_X$ |
  by (rule free-group-card-infinite)
moreover
from ⟨¬ finite Y⟩ have |Y| =o |carrier  $\mathcal{F}_Y$ |
  by (rule free-group-card-infinite)
moreover
from iso have |carrier  $\mathcal{F}_X$ | =o |carrier  $\mathcal{F}_Y$ |
  by (auto simp add: Group.iso-def iff:card-of-ordIso[THEN sym])
ultimately
have |X| =o |Y| by (auto intro: ordIso-equivalence)
thus ?thesis by (subst card-of-ordIso)
qed
qed

end

```

7 The Ping Pong lemma

```

theory PingPongLemma

```

```

imports

```

```

  HOL-Algebra.Bij

```

```

  FreeGroups

```

```

begin

```

The Ping Pong Lemma is a way to recognize a Free Group by its action on a set (often a topological space or a graph). The name stems from the way that elements of the set are passed forth and back between the subsets given there.

We start with two auxiliary lemmas, one about the identity of the group of bijections, and one about sets of cardinality larger than one.

```

lemma Bij-one[simp]:

```

```

  assumes x ∈ X

```

```

  shows 1BijGroup X x = x

```

```

using assms by (auto simp add: BijGroup-def)

```

```

lemma other-member:

```

```

  assumes I ≠ {} and i ∈ I and card I ≠ 1

```

```

  obtains j where j ∈ I and j ≠ i

```

```

proof(cases finite I)

```

```

  case True

```

hence $I - \{i\} \neq \{\}$ **using** $\langle \text{card } I \neq 1 \rangle$ **and** $\langle i \in I \rangle$ **by** $(\text{metis } \text{Suc-eq-plus1-left}$
 $\text{card-Diff-subset-Int } \text{card-Suc-Diff1 } \text{diff-add-inverse2 } \text{diff-self-eq-0 } \text{empty-Diff } \text{finite.emptyI}$
 $\text{inf-bot-left } \text{minus-nat.diff-0})$
thus $?thesis$ **using** $that$ **by** $auto$
next
case $False$
hence $I - \{i\} \neq \{\}$ **by** $(\text{metis } \text{Diff-empty } \text{finite.emptyI } \text{finite-Diff-insert})$
thus $?thesis$ **using** $that$ **by** $auto$
qed

And now we can attempt the lemma. The gencount condition is a weaker variant of “x has to lie outside all subsets” that is only required if the set of generators is one. Otherwise, we will be able to find a suitable x to start with in the proof.

lemma *ping-pong-lemma*:

assumes $group\ G$
and $act \in \text{hom } G\ (\text{BijGroup } X)$
and $g \in (I \rightarrow \text{carrier } G)$
and $\langle g \text{ ' } I \rangle_G = \text{carrier } G$
and $sub1: \forall i \in I. Xout\ i \subseteq X$
and $sub2: \forall i \in I. Xin\ i \subseteq X$
and $disj1: \forall i \in I. \forall j \in I. i \neq j \longrightarrow Xout\ i \cap Xout\ j = \{\}$
and $disj2: \forall i \in I. \forall j \in I. i \neq j \longrightarrow Xin\ i \cap Xin\ j = \{\}$
and $disj3: \forall i \in I. \forall j \in I. Xin\ i \cap Xout\ j = \{\}$
and $x \in X$
and $gencount: \forall i. I = \{i\} \longrightarrow (x \notin Xout\ i \wedge x \notin Xin\ i)$
and $ping: \forall i \in I. act\ (g\ i) \text{ ' } (X - Xout\ i) \subseteq Xin\ i$
and $pong: \forall i \in I. act\ (\text{inv}_G\ (g\ i)) \text{ ' } (X - Xin\ i) \subseteq Xout\ i$
shows $group.lift\ G\ g \in iso\ (\mathcal{F}_I)\ G$

proof –

interpret $F: group\ \mathcal{F}_I$
using $assms$ **by** $(\text{auto } \text{simp } \text{add: } \text{free-group-is-group})$
interpret $G: group\ G$ **by** $fact$
interpret $B: group\ \text{BijGroup } X$ **using** $group\text{-BijGroup}$ **by** $auto$
interpret $act: group\text{-hom } G\ \text{BijGroup } X$ **act** **by** $(\text{unfold-locales})\ fact$
interpret $h: group\text{-hom } \mathcal{F}_I\ G\ G.lift\ g$
using $F.is-group\ G.is-group\ G.lift-is-hom\ assms$
by $(\text{auto } \text{intro!: } \text{group-hom.intro } \text{group-hom-axioms.intro})$

show $?thesis$

proof $(rule\ h.group-hom-isoI)$

Injectivity is the hard part of the proof.

show $\forall x \in \text{carrier } \mathcal{F}_I. G.lift\ g\ x = \mathbf{1}_G \longrightarrow x = \mathbf{1}_{\mathcal{F}_I}$

proof $(rule+)$

We lift the Xout and Xin sets to generators and their inverses, and create variants of the disj-conditions:

define $Xout'$ **where** $Xout' = (\lambda(b,i::'d). \text{if } b \text{ then } Xin\ i \text{ else } Xout\ i)$

```

define  $Xin'$  where  $Xin' = (\lambda(b,i::'d). \text{if } b \text{ then } Xout \ i \ \text{else } Xin \ i)$ 

have  $disj1'$ :  $\forall i \in (UNIV \times I). \forall j \in (UNIV \times I). i \neq j \longrightarrow Xout' \ i \cap Xout'$ 
 $j = \{\}$ 
  using  $disj1$ [rule-format]  $disj2$ [rule-format]  $disj3$ [rule-format]
  by (auto simp add:Xout'-def Xin'-def split:if-splits, blast+)
have  $disj2'$ :  $\forall i \in (UNIV \times I). \forall j \in (UNIV \times I). i \neq j \longrightarrow Xin' \ i \cap Xin' \ j$ 
 $= \{\}$ 
  using  $disj1$ [rule-format]  $disj2$ [rule-format]  $disj3$ [rule-format]
  by (auto simp add:Xout'-def Xin'-def split:if-splits, blast+)
have  $disj3'$ :  $\forall i \in (UNIV \times I). \forall j \in (UNIV \times I). \neg \text{canceling } i \ j \longrightarrow Xin' \ i$ 
 $\cap Xout' \ j = \{\}$ 
  using  $disj1$ [rule-format]  $disj2$ [rule-format]  $disj3$ [rule-format]
  by (auto simp add:canceling-def Xout'-def Xin'-def split:if-splits, blast)

```

We need to pick a suitable element of the set to play ping pong with. In particular, it needs to be outside of the Xout-set of the last generator in the list, and outside the in-set of the first element. This part of the proof is surprisingly tedious, because there are several cases, some similar but not the same.

```

fix  $w$ 
assume  $w$ :  $w \in \text{carrier } \mathcal{F}_I$ 

obtain  $x$  where  $x \in X$ 
  and  $x1$ :  $w = [] \vee x \notin Xout' \ (\text{last } w)$ 
  and  $x2$ :  $w = [] \vee x \notin Xin' \ (\text{hd } w)$ 
proof–
  { assume  $I = \{\}$ 
    hence  $w = []$  using  $w$  by (auto simp add:free-group-def)
    hence ?thesis using that  $\langle x \in X \rangle$  by auto
  }
  moreover
  { assume  $\text{card } I = 1$ 
    then obtain  $i$  where  $I = \{i\}$  by (auto dest: card-eq-SucD)
    assume  $w \neq []$ 
    hence  $\text{snd } (\text{hd } w) = i$  and  $\text{snd } (\text{last } w) = i$ 
      using  $w \langle I = \{i\} \rangle$ 
      apply (cases  $w$ , auto simp add:free-group-def)
      apply (cases  $w$  rule:rev-exhaust, auto simp add:free-group-def)
      done
    hence ?thesis using gencount[rule-format, OF  $\langle I = \{i\} \rangle$ ] that[OF  $\langle x \in X \rangle$ ]
  }
   $\langle w \neq [] \rangle$ 
  by (cases last  $w$ , cases hd  $w$ , auto simp add:Xout'-def Xin'-def split:if-splits)
  }
  moreover
  { assume  $I \neq \{\}$  and  $\text{card } I \neq 1$  and  $w \neq []$ 

    from  $\langle w \neq [] \rangle$  and  $w$ 
    obtain  $b \ i$  where  $\text{hd } w = (b, i)$  and  $i \in I$ 
      by (cases  $w$ , auto simp add:free-group-def)
    from  $\langle w \neq [] \rangle$  and  $w$ 

```

obtain $b' i'$ **where** $last: last w = (b', i')$ **and** $i' \in I$
by (*cases w rule: rev-exhaust, auto simp add: free-group-def*)

What follows are two very similar cases, but the correct choice of variables depends on where we find x .

```

{
obtain  $b'' i''$  where
  ( $b'', i'' \neq (b, i)$ ) and
  ( $b'', i'' \neq (b', i')$ ) and
   $\neg canceling (b'', i'') (b', i')$  and
   $i'' \in I$ 
proof(cases i=i')
  case True
    obtain  $j$  where  $j \in I$  and  $j \neq i$  using  $\langle card I \neq 1 \rangle$  and  $\langle i \in I \rangle$ 
    by  $\neg(rule other-member, auto)$ 
    with True show ?thesis using that by (auto simp add: canceling-def)
  next
    case False thus ?thesis using that  $\langle i \in I \rangle \langle i' \in I \rangle$ 
    by (simp add: canceling-def, metis)
qed
let  $?g = (b'', i'')$ 

assume  $x \in Xout'$  (last w)
hence  $x \notin Xout' ?g$ 
  using disj1 [rule-format, OF - -  $\langle ?g \neq (b', i') \rangle$ ]
     $\langle i \in I \rangle \langle i' \in I \rangle \langle i'' \in I \rangle$  hd last
  by auto
hence act ( $G.lift-gi g ?g$ )  $x \in Xin' ?g$  (is  $?x \in -$ ) using  $\langle i'' \in I \rangle \langle x \in$ 
X
  ping[rule-format, OF  $\langle i'' \in I \rangle$ , THEN subsetD]
  pong[rule-format, OF  $\langle i'' \in I \rangle$ , THEN subsetD]
  by (auto simp add: G.lift-def G.lift-gi-def Xout'-def Xin'-def)
hence  $?x \notin Xout' (last w) \wedge ?x \notin Xin' (hd w)$ 
  using
    disj3 [rule-format, OF - -  $\langle \neg canceling (b'', i'') (b', i') \rangle$ ]
    disj2 [rule-format, OF - -  $\langle ?g \neq (b, i) \rangle$ ]
     $\langle i \in I \rangle \langle i' \in I \rangle \langle i'' \in I \rangle$  hd last
  by (auto simp add: canceling-def)
moreover
note  $\langle i'' \in I \rangle$ 
hence  $g i'' \in carrier G$  using  $\langle g \in (I \rightarrow carrier G) \rangle$  by auto
hence  $G.lift-gi g ?g \in carrier G$ 
  by (auto simp add: G.lift-gi-def inv1-def)
hence act ( $G.lift-gi g ?g$ )  $\in carrier (BijGroup X)$ 
  using  $\langle act \in hom G (BijGroup X) \rangle$  by auto
hence  $?x \in X$  using  $\langle x \in X \rangle$ 
  by (auto simp add: BijGroup-def Bij-def bij-betw-def)
ultimately have ?thesis using that[of ?x] by auto
}
moreover

```

```

{
obtain  $b'' i''$  where
   $\neg$  canceling  $(b'', i'')$   $(b, i)$  and
   $\neg$  canceling  $(b'', i'')$   $(b', i')$  and
   $(b, i) \neq (b'', i'')$  and
   $i'' \in I$ 
proof (cases  $i=i'$ )
  case True
    obtain  $j$  where  $j \in I$  and  $j \neq i$  using  $\langle \text{card } I \neq 1 \rangle$  and  $\langle i \in I \rangle$ 
    by  $\neg$ (rule other-member, auto)
    with True show ?thesis using that by (auto simp add:canceling-def)
  next
    case False thus ?thesis using that  $\langle i \in I \rangle \langle i' \in I \rangle$ 
    by (simp add:canceling-def, metis)
qed
let ?g =  $(b'', i'')$ 
note cancel-sym-neg[OF  $\langle \neg$  canceling  $(b'', i'')$   $(b, i) \rangle$ ]
note cancel-sym-neg[OF  $\langle \neg$  canceling  $(b'', i'')$   $(b', i') \rangle$ ]

assume  $x \in \text{Xin}'$  (hd w)
hence  $x \notin \text{Xout}'$  ?g
  using disj3'[rule-format, OF - -  $\langle \neg$  canceling  $(b, i)$  ?g]
   $\langle i \in I \rangle \langle i' \in I \rangle \langle i'' \in I \rangle$  hd last
  by auto
hence act  $(G.\text{lift-gi } g ?g)$   $x \in \text{Xin}'$  ?g (is ?x  $\in$  -) using  $\langle i'' \in I \rangle \langle x \in$ 
X>
  ping[rule-format, OF  $\langle i'' \in I \rangle$ , THEN subsetD]
  pong[rule-format, OF  $\langle i'' \in I \rangle$ , THEN subsetD]
  by (auto simp add:G.lift-def G.lift-gi-def Xout'-def Xin'-def)
hence ?x  $\notin \text{Xout}'$  (last w)  $\wedge$  ?x  $\notin \text{Xin}'$  (hd w)
  using
    disj3'[rule-format, OF - -  $\langle \neg$  canceling ?g  $(b', i')$ ]
    disj2'[rule-format, OF - -  $\langle (b, i) \neq ?g \rangle$ ]
     $\langle i \in I \rangle \langle i' \in I \rangle \langle i'' \in I \rangle$  hd last
  by (auto simp add: canceling-def)
moreover
note  $\langle i'' \in I \rangle$ 
hence  $g i'' \in \text{carrier } G$  using  $\langle g \in (I \rightarrow \text{carrier } G) \rangle$  by auto
hence  $G.\text{lift-gi } g ?g \in \text{carrier } G$ 
  by (auto simp add:G.lift-gi-def)
hence act  $(G.\text{lift-gi } g ?g) \in \text{carrier } (\text{BijGroup } X)$ 
  using  $\langle \text{act} \in \text{hom } G (\text{BijGroup } X) \rangle$  by auto
hence ?x  $\in X$  using  $\langle x \in X \rangle$ 
  by (auto simp add:BijGroup-def Bij-def bij-betw-def)
ultimately have ?thesis using that[of ?x] by auto
}
moreover note calculation
}
ultimately show ?thesis using  $\langle x \in X \rangle$  that by auto

```


qed

The proof works by induction over the length of the word. Each inductive step is one ping as in ping pong. At the end, we land in one of the subsets of X , so the word cannot be the identity.

```

from  $x1$  and  $w$ 
have  $w = [] \vee \text{act } (G.\text{lift } g \ w) \ x \in \text{Xin}' \ (\text{hd } w)$ 
proof(induct w)
  case Nil show ?case by simp
next case (Cons w ws)
  note  $C = \text{Cons}$ 

```

The following lemmas establish all “obvious” element relations that will be required during the proof.

```

note  $\text{calculation} = \text{Cons}(3)$ 
moreover have  $x \in X$  by fact
moreover have  $\text{snd } w \in I$  using  $\text{calculation}$  by (auto simp add:free-group-def)

```

```

moreover have  $g \in (I \rightarrow \text{carrier } G)$  by fact
moreover have  $g \ (\text{snd } w) \in \text{carrier } G$  using  $\text{calculation}$  by auto
moreover have  $ws \in \text{carrier } \mathcal{F}_I$ 
  using  $\text{calculation}$  by (auto intro:cons-canceled simp add:free-group-def)
moreover have  $G.\text{lift } g \ ws \in \text{carrier } G$  and  $G.\text{lift } g \ [w] \in \text{carrier } G$ 
  using  $\text{calculation}$  by (auto simp add: free-group-def)
moreover have  $\text{act } (G.\text{lift } g \ ws) \in \text{carrier } (\text{BijGroup } X)$ 
  and  $\text{act } (G.\text{lift } g \ [w]) \in \text{carrier } (\text{BijGroup } X)$ 
  and  $\text{act } (G.\text{lift } g \ (w\#ws)) \in \text{carrier } (\text{BijGroup } X)$ 
  and  $\text{act } (g \ (\text{snd } w)) \in \text{carrier } (\text{BijGroup } X)$ 
  using  $\text{calculation}$  by auto
moreover have  $\text{act } (g \ (\text{snd } w)) \in \text{Bij } X$ 
  using  $\text{calculation}$  by (auto simp add:BijGroup-def)
moreover have  $\text{act } (G.\text{lift } g \ ws) \ x \in X$  (is  $?x2 \in X$ )
  using  $\text{calculation}$  by (auto simp add:BijGroup-def Bij-def bij-betw-def)
moreover have  $\text{act } (G.\text{lift } g \ [w]) \ ?x2 \in X$ 
  using  $\text{calculation}$  by (auto simp add:BijGroup-def Bij-def bij-betw-def)
moreover have  $\text{act } (G.\text{lift } g \ (w\#ws)) \ x \in X$ 
  using  $\text{calculation}$  by (auto simp add:BijGroup-def Bij-def bij-betw-def)
moreover note  $\text{mems} = \text{calculation}$ 

```

```

have  $\text{act } (G.\text{lift } g \ ws) \ x \notin \text{Xout}' \ w$ 
proof(cases ws)
  case Nil
    moreover have  $x \notin \text{Xout}' \ w$  using  $\text{Cons}(2)$   $\text{Nil}$ 
      unfolding  $\text{Xout}'\text{-def}$  using  $\text{mems}$ 
      by (auto split:if-splits)
    ultimately show  $\text{act } (G.\text{lift } g \ ws) \ x \notin \text{Xout}' \ w$ 
      using  $\text{mems}$  by auto
  next case (Cons ww wws)
    hence  $\text{act } (G.\text{lift } g \ ws) \ x \in \text{Xin}' \ (\text{hd } ws)$ 
      using  $C$   $\text{mems}$  by simp

```

```

moreover have  $Xin' (hd\ ws) \cap Xout' w = \{\}$ 
proof-
  have  $\neg\ canceling\ (hd\ ws)\ w$ 
  proof
    assume  $canceling\ (hd\ ws)\ w$ 
    hence  $cancel\text{-}to\text{-}1\ (w\#\ ws)\ wws$  using  $Cons$ 
    by( $auto\ simp\ add:\ cancel\text{-}sym\ cancel\text{-}to\text{-}1\text{-}def\ cancel\text{-}to\text{-}1\text{-}at\text{-}def$ 
 $cancel\text{-}at\text{-}def$ )
    thus  $False$  using  $\langle w\#\ ws \in carrier\ \mathcal{F} \rangle$ 
    by( $auto\ simp\ add:\ free\text{-}group\text{-}def\ canceled\text{-}def$ )
  qed

  have  $w \in UNIV \times I\ hd\ ws \in UNIV \times I$ 
  using  $\langle snd\ w \in I \rangle\ mems\ Cons$ 
  by ( $cases\ w,\ auto,\ cases\ hd\ ws,\ auto\ simp\ add:\ free\text{-}group\text{-}def$ )
  thus  $?thesis$ 
  by- ( $rule\ disj3\ [rule\text{-}format,\ OF\ -\ -\ \langle \neg\ canceling\ (hd\ ws)\ w \rangle],\ auto$ )
qed
ultimately show  $act\ (G.lift\ g\ ws)\ x \notin Xout' w$  using  $Cons$  by  $auto$ 
qed
show  $?case$ 
proof-
  have  $act\ (G.lift\ g\ (w\ \#\ ws))\ x = act\ (G.lift\ g\ ([w]\ @\ ws))\ x$  by  $simp$ 
  also have  $\dots = act\ (G.lift\ g\ [w]\ \otimes_G\ G.lift\ g\ ws)\ x$ 
  using  $mems$  by ( $subst\ G.lift\text{-}append,\ auto\ simp\ add:\ free\text{-}group\text{-}def$ )
  also have  $\dots = (act\ (G.lift\ g\ [w])\ \otimes_{BijGroup\ X}\ act\ (G.lift\ g\ ws))\ x$ 
  using  $mems$  by ( $auto\ simp\ add:\ act.\ hom\text{-}mult\ free\text{-}group\text{-}def\ in\text{-}tro!\ G.lift\text{-}closed$ )
  also have  $\dots = act\ (G.lift\ g\ [w])\ (act\ (G.lift\ g\ ws)\ x)$ 
  using  $mems$  by ( $auto\ simp\ add:\ BijGroup\text{-}def\ compose\text{-}def$ )
  also have  $\dots \notin act\ (G.lift\ g\ [w])\ \langle Xout' w \rangle$ 
  apply( $rule\ ccontr$ )
  apply  $simp$ 
  apply ( $erule\ imageE$ )
  apply ( $subst\ (asm)\ inj\text{-}on\text{-}eq\text{-}iff\ [of\ act\ (G.lift\ g\ [w])\ X]$ )
  using  $mems\ \langle act\ (G.lift\ g\ ws)\ x \notin Xout' w \rangle\ \langle \forall i \in I.\ Xout\ i \subseteq X \rangle$ 
 $\langle \forall i \in I.\ Xin\ i \subseteq X \rangle$ 
  apply ( $auto\ simp\ add:\ BijGroup\text{-}def\ Bij\text{-}def\ bij\text{-}betw\text{-}def\ free\text{-}group\text{-}def$ 
 $Xout'\text{-}def\ split:\ if\text{-}splits$ )
  apply  $blast+$ 
  done
finally
have  $act\ (G.lift\ g\ (w\ \#\ ws))\ x \in Xin' w$ 
proof-
  assume  $act\ (G.lift\ g\ (w\ \#\ ws))\ x \notin act\ (G.lift\ g\ [w])\ \langle Xout' w \rangle$ 
  hence  $act\ (G.lift\ g\ (w\ \#\ ws))\ x \in (X - act\ (G.lift\ g\ [w])\ \langle Xout' w \rangle)$ 
  using  $mems$  by  $auto$ 
  also have  $\dots \subseteq act\ (G.lift\ g\ [w])\ \langle X - act\ (G.lift\ g\ [w])\ \langle Xout' w \rangle \rangle$ 
  using  $\langle act\ (G.lift\ g\ [w]) \in carrier\ (BijGroup\ X) \rangle$ 

```

```

      by (auto simp add:BijGroup-def Bij-def bij-betw-def)
    also have ...  $\subseteq$  act (G.lift g [w])  $\langle X - Xout' w \rangle$ 
      by (rule image-diff-subset)
    also have ...  $\subseteq$  Xin' w
    proof(cases fst w)
      assume  $\neg$  fst w
      thus ?thesis
        using mems
          by (auto intro!: ping[rule-format, THEN subsetD] simp add:
Xout'-def Xin'-def G.lift-def G.lift-gi-def free-group-def)
      next assume fst w
      thus ?thesis
        using mems
          by (auto intro!: pong[rule-format, THEN subsetD] simp add:
restrict-def inv-BijGroup Xout'-def Xin'-def G.lift-def G.lift-gi-def free-group-def)
    qed
    finally show ?thesis .
  qed
  thus ?thesis by simp
qed
qed
moreover assume G.lift g w =  $\mathbf{1}_G$ 
ultimately show w =  $\mathbf{1}_{\mathcal{F}_I}$ 
  using  $\langle x \in X \rangle$  Cons(1) x2  $\langle w \in \text{carrier } \mathcal{F}_I \rangle$ 
  by (cases w, auto simp add:free-group-def Xin'-def split-if-splits)
qed
next

Surjectivity is relatively simple, and often not even mentioned in human proofs.

have G.lift g  $\langle \text{carrier } \mathcal{F}_I =$ 
  G.lift g  $\langle \iota \langle I \rangle_{\mathcal{F}_I}$ 
  by (metis gens-span-free-group)
also have ... =  $\langle G.lift g \langle \iota \langle I \rangle \rangle_G$ 
  by (auto intro!:h.hom-span simp add: insert-closed)
also have ... =  $\langle g \langle I \rangle \rangle_G$ 
proof-
  have  $\forall i \in I. G.lift g (\iota i) = g i$ 
    using  $\langle g \in (I \rightarrow \text{carrier } G) \rangle$ 
    by (auto simp add:insert-def G.lift-def G.lift-gi-def intro:G.r-one)
  then have G.lift g  $\langle \iota \langle I \rangle = g \langle I$ 
    by (auto intro!: image-cong simp add: image-comp [symmetric, THEN
sym])
  thus ?thesis by simp
qed
also have ... = carrier G using assms by simp
finally show G.lift g  $\langle \text{carrier } \mathcal{F}_I = \text{carrier } G.$ 
qed
qed

```

end