Free Groups

Joachim Breitner

March 17, 2025

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

theory *Cancelation*
imports
HOL−Proofs−Lambda.Commutation
begin

This theory defines cancelation via relations. The one-step relation *cancels-to-1 a b* describes that *b* is obtained from *a* by removing exactly one pair of generators, while *cancels-to* 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*:
$\llbracket$ *wfP* $(R\hat{}{-}{-}1)$; $\bigwedge a\ b\ c.\ R\ a\ b \Longrightarrow R\ a\ c \Longrightarrow \exists\, d.\ R\hat{}{*}{*}\ b\ d \wedge R\hat{}{*}{*}\ c\ d$ $\rrbracket \Longrightarrow$ *confluent R*
⟨*proof*⟩

lemma *confluentD*:
$\llbracket$ *confluent R*; $R\hat{}{*}{*}\ a\ b$; $R\hat{}{*}{*}\ a\ c$ $\rrbracket \Longrightarrow \exists\, d.\ R\hat{}{*}{*}\ b\ d \wedge R\hat{}{*}{*}\ c\ d$
⟨*proof*⟩

lemma *tranclp-DomainP*: $R\hat{}{+}{+}\ a\ b \Longrightarrow$ *Domainp R a*
⟨*proof*⟩

lemma *confluent-unique-normal-form*:
$[\![$ *confluent R*; R^** *a b*; R^** *a c*; ¬ *Domainp R b*; ¬ *Domainp R c* $]\!] \Longrightarrow b = c$
⟨*proof*⟩

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 symmetic.

lemma *cancel-cancel*: $[\![$ *canceling a b*; *canceling b c* $]\!] \Longrightarrow a = c$
⟨*proof*⟩

lemma *cancel-sym*: *canceling a b* ⟹ *canceling b a*
⟨*proof*⟩

lemma *cancel-sym-neg*: ¬*canceling a b* ⟹ ¬*canceling b a*
⟨*proof*⟩

1.3 Definition of the *cancels-to* relation

First, we define the function that removes the ith 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* < *length l* ⟹ *length* (*cancel-at i l*) = *length l* − *2*
⟨*proof*⟩

lemma *cancel-at-nth1*[*simp*]:
$[\![$ *n* < *i*; *1*+*i* < *length l* $]\!] \Longrightarrow$ (*cancel-at i l*) ! *n* = *l* ! *n*
⟨*proof*⟩

lemma *cancel-at-nth2*[*simp*]:
assumes *n* ≥ *i* **and** *n* < *length l* − *2*
shows (*cancel-at i l*) ! *n* = *l* ! (*n* + *2*)
⟨*proof*⟩

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

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

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

definition *cancels-to-1-at* :: *nat* ⇒ *'a word-g-i* ⇒ *'a word-g-i* ⇒ *bool*
where *cancels-to-1-at i l1 l2* = (*0*≤*i* ∧ (*1*+*i*) < *length l1*
∧ *canceling* (*l1* ! *i*) (*l1* ! (*1*+*i*))
∧ (*l2* = *cancel-at i l1*))

definition *cancels-to-1* :: *'a word-g-i* ⇒ *'a word-g-i* ⇒ *bool*
where *cancels-to-1 l1 l2* = (∃ *i*. *cancels-to-1-at i l1 l2*)

definition *cancels-to* :: *'a word-g-i* ⇒ *'a word-g-i* ⇒ *bool*
where *cancels-to* = *cancels-to-1*^**

lemma *cancels-to-trans* [*trans*]:
⟦ *cancels-to a b*; *cancels-to b c* ⟧ ⟹ *cancels-to a c*
⟨*proof*⟩

definition *canceled* :: *'a word-g-i* ⇒ *bool*
where *canceled l* = (¬ *Domainp cancels-to-1 l*)

lemma *cancels-to-1-unfold*:
assumes *cancels-to-1 x y*
obtains *xs1 x1 x2 xs2*
where *x* = *xs1* @ *x1* # *x2* # *xs2*
and *y* = *xs1* @ *xs2*
and *canceling x1 x2*
⟨*proof*⟩

lemma *cancels-to-1-fold*:
canceling x1 x2 ⟹ *cancels-to-1* (*xs1* @ *x1* # *x2* # *xs2*) (*xs1* @ *xs2*)
⟨*proof*⟩

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* (*cancels-to-1*^−−*1*)
⟨*proof*⟩

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* (*Suc i*) *l b*
shows *a* = *b*

⟨*proof*⟩

lemma *canceling-indep*:
assumes *cancels-to-1-at i l a* **and** *cancels-to-1-at j l b* **and** $j > Suc\ i$
obtains *c* **where** *cancels-to-1-at* $(j - 2)$ *a c* **and** *cancels-to-1-at i b c*
⟨*proof*⟩

This is the confluence lemma

lemma *confluent-cancels-to-1*: *confluent cancels-to-1*
⟨*proof*⟩

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

lemma *norm-form-uniq*:
assumes *cancels-to a b*
and *cancels-to a c*
and *canceled b*
and *canceled c*
shows $b = c$
⟨*proof*⟩

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*⟩

lemma *cancel-to-append*:
assumes *cancels-to a b*
shows *cancels-to* $(l@a@l')$ $(l@b@l')$
⟨*proof*⟩

lemma *cancels-to-append2*:
assumes *cancels-to* $a\ a'$
and *cancels-to* $b\ b'$
shows *cancels-to* $(a@b)$ $(a'@b')$
⟨*proof*⟩

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

lemma *empty-canceled*[*simp*]: *canceled* []
⟨*proof*⟩

lemma *singleton-canceled*[*simp*]: *canceled* [*a*]
⟨*proof*⟩

lemma *cons-canceled*:

assumes *canceled* $(a\#x)$
shows *canceled* x
⟨*proof*⟩

lemma *cancels-to-self*[*simp*]: *cancels-to* l l
⟨*proof*⟩

1.4 Definition of normalization

Using the THE construct, we can define the normalization function *normalize* as the unique fully cancled 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*⟩

lemma *normalize-discover*:
assumes *canceled* l'
and *cancels-to* l l'
shows *normalize* $l = l'$
⟨*proof*⟩

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

lemma *normalize-canceled*[*simp*]:
assumes *cancels-to* l l'
shows *normalize* l = *normalize* l'
⟨*proof*⟩

Normalization is idempotent.

lemma *normalize-idemp*[*simp*]:
assumes *canceled* l
shows *normalize* $l = l$
⟨*proof*⟩

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*⟩

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 *cancels-to-1-preserves-generators*:
  assumes *cancels-to-1 l l′*
    and $l \in$ *lists* (*UNIV* × *gens*)
  shows $l' \in$ *lists* (*UNIV* × *gens*)
⟨*proof*⟩

lemma *cancels-to-preserves-generators*:
  assumes *cancels-to l l′*
    and $l \in$ *lists* (*UNIV* × *gens*)
  shows $l' \in$ *lists* (*UNIV* × *gens*)
⟨*proof*⟩

lemma *normalize-preserves-generators*:
  assumes $l \in$ *lists* (*UNIV* × *gens*)
   shows *normalize l* ∈ *lists* (*UNIV* × *gens*)
⟨*proof*⟩

Two simplification lemmas about lists.

lemma *empty-in-lists*[*simp*]:
  [] ∈ *lists A* ⟨*proof*⟩

lemma *lists-empty*[*simp*]: *lists* {} = {[]}
  ⟨*proof*⟩

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*: *cancel-at i* (*map f l*) = *map f* (*cancel-at i l*)
⟨*proof*⟩

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*⟩

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′*)

⟨*proof*⟩

lemma *rename-gens-canceled*:
  assumes *inj-on g* (*snd*‘*set l*)
    and *canceled l*
  shows *canceled* (*map* (*map-prod f g*) *l*)
⟨*proof*⟩

lemma *rename-gens-normalize*:
  assumes *inj f*
  and *inj-on g* (*snd* ‘ *set l*)
  shows *normalize* (*map* (*map-prod f g*) *l*) = *map* (*map-prod f g*) (*normalize l*)
⟨*proof*⟩

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* ⇒ *'a set* ⇒ *'a set* (‹⟨-⟩ı›)
  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*: $⟦ x \in \langle gens\rangle_G; y \in \langle gens\rangle_G ⟧ \Longrightarrow x \otimes_G y \in \langle gens\rangle_G$

lemma (**in** *group*) *gen-span-closed*:
  assumes *gens* ⊆ *carrier G*
  shows $\langle gens\rangle_G \subseteq$ *carrier G*
⟨*proof*⟩

lemma (**in** *group*) *gen-subgroup-is-subgroup*:
$\quad gens \subseteq carrier\ G \Longrightarrow \langle gens \rangle_G \leq G$
$\langle proof \rangle$

lemma (**in** *group*) *gen-subgroup-is-smallest-containing*:
assumes $gens \subseteq carrier\ G$
shows $\bigcap \{H.\ H \leq G \wedge gens \subseteq H\} = \langle gens \rangle_G$
$\langle proof \rangle$

2.2 Generators and homomorphisms

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

lemma *hom-unique-on-span*:
assumes *group G*
and *group H*
and $gens \subseteq carrier\ G$
and $h \in hom\ G\ H$
and $h' \in hom\ G\ H$
and $\forall g \in gens.\ h\ g = h'\ g$
shows $\forall x \in \langle gens \rangle_G.\ h\ x = h'\ x$
$\langle proof \rangle$

2.3 Sets of generators

There is no definition for "*gens* is a generating set of *G*". This is easily expressed by $\langle gens \rangle = carrier\ G$.

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 *gens*: $\langle gens \rangle_G = carrier\ G$
and $h \in hom\ G\ H$
and $h' \in hom\ G\ H$
and $\forall g \in gens.\ h\ g = h'\ g$
shows $\forall x \in carrier\ G.\ h\ x = h'\ x$
$\langle proof \rangle$

lemma (**in** *group-hom*) *hom-span*:
assumes $gens \subseteq carrier\ G$
shows $h\ `\ (\langle gens \rangle_G) = \langle h\ `\ gens \rangle_H$
$\langle proof \rangle$

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-concat*
  where *m-concat* $l \equiv$ *foldr* $(\otimes)$ l $\mathbf{1}$

lemma (**in** *monoid*) *m-concat-closed*[*simp*]:
 set $l \subseteq$ *carrier* $G \Longrightarrow$ *m-concat* $l \in$ *carrier* G
  ⟨*proof*⟩

lemma (**in** *monoid*) *m-concat-append*[*simp*]:
  assumes *set* $a \subseteq$ *carrier* G
    and *set* $b \subseteq$ *carrier* G
  shows *m-concat* $(a@b) =$ *m-concat* $a \otimes$ *m-concat* b
⟨*proof*⟩

lemma (**in** *monoid*) *m-concat-cons*[*simp*]:
 $[\![$ $x \in$ *carrier* G ; *set* $xs \subseteq$ *carrier* G $]\!] \Longrightarrow$ *m-concat* $(x\#xs) = x \otimes$ *m-concat* xs
⟨*proof*⟩

lemma (**in** *monoid*) *nat-pow-mult1l*:
  assumes x: $x \in$ *carrier* G
  shows $x \otimes x$ [^] $n = x$ [^] *Suc* n
⟨*proof*⟩

lemma (**in** *monoid*) *m-concat-power*[*simp*]: $x \in$ *carrier* $G \Longrightarrow$ *m-concat* (*replicate* n x) $= x$ [^] n
⟨*proof*⟩

2.5 Isomorphisms

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

lemma *group-homI*[*intro*]:
  assumes *range*: h ‘ (*carrier g1*) $\subseteq$ *carrier g2*
    and *hom*: $\forall x \in$*carrier g1*. $\forall y \in$*carrier g1*. h $(x \otimes_{g1} y) = h$ $x \otimes_{g2} h$ y
  shows $h \in$ *hom g1 g2*
⟨*proof*⟩

lemma (**in** *group-hom*) *hom-injI*:
  assumes $\forall x \in$*carrier* G. h $x = \mathbf{1}_H \longrightarrow x = \mathbf{1}_G$
  shows *inj-on* h (*carrier* G)
⟨*proof*⟩

lemma (**in** *group-hom*) *group-hom-isoI*:
  assumes *inj1*: $\forall x \in$*carrier* G. h $x = \mathbf{1}_H \longrightarrow x = \mathbf{1}_G$
    and *surj*: h ‘ (*carrier* G) $=$ *carrier* H
  shows $h \in$ *iso* G H
⟨*proof*⟩

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

assumes *G*: *group G*
and *H*: *group H*
and *inj1*: $\forall x \in carrier\ G.\ h\ x = \mathbf{1}_H \longrightarrow x = \mathbf{1}_G$
and *surj*: *h ‘ (carrier G) = carrier H*
and *hom*: $\forall x \in carrier\ G.\ \forall y \in carrier\ G.\ h\ (x \otimes_G y) = h\ x \otimes_H h\ y$
shows $h \in iso\ G\ H$
⟨*proof*⟩
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* ⇒ *'a g-i*
where *inv1* = *apfst Not*

lemma *inv1-inv1*: *inv1* ∘ *inv1* = *id*
⟨*proof*⟩

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

lemma *snd-inv1*: *snd* ∘ *inv1* = *snd*
⟨*proof*⟩

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* ⇒ *'a word-g-i*
where *inv-fg l* = *rev* (*map inv1 l*)

lemma *cancelling-inf*[*simp*]: *canceling* (*inv1 a*) (*inv1 b*) = *canceling a b*
⟨*proof*⟩

lemma *inv-idemp*: *inv-fg* (*inv-fg l*) = *l*
⟨*proof*⟩

lemma *inv-fg-cancel*: *normalize* (l @ *inv-fg* l) = []
⟨*proof*⟩

lemma *inv-fg-cancel2*: *normalize* (*inv-fg* l @ l) = []
⟨*proof*⟩

lemma *canceled-rev*:
assumes *canceled* l
shows *canceled* (*rev* l)
⟨*proof*⟩

lemma *inv-fg-closure1*:
assumes *canceled* l
shows *canceled* (*inv-fg* l)
⟨*proof*⟩

lemma *inv-fg-closure2*:
$l \in$ *lists* (*UNIV* $\times$ *gens*) $\Longrightarrow$ *inv-fg* $l \in$ *lists* (*UNIV* $\times$ *gens*)
⟨*proof*⟩

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* (‹$\mathcal{F}$ı›)
where
$\mathcal{F}_{gens}$ ≡ (|
carrier = {l∈*lists* (*UNIV* $\times$ *gens*). *canceled* l },
mult = λ x y. *normalize* (x @ y),
one = []
|)

lemma *occuring-gens-in-element*:
$x \in$ *carrier* $\mathcal{F}_{gens} \Longrightarrow x \in$ *lists* (*UNIV* $\times$ *gens*)
⟨*proof*⟩

theorem *free-group-is-group*: *group* $\mathcal{F}_{gens}$
⟨*proof*⟩

lemma *inv-is-inv-fg*[*simp*]:
$x \in$ *carrier* $\mathcal{F}_{gens} \Longrightarrow inv_{\mathcal{F}_{gens}}\ x =$ *inv-fg* x
⟨*proof*⟩

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›)
  where ι *g* = [(*False*, *g*)]

lemma *insert-closed*:
  $g \in gens \Longrightarrow \iota\ g \in carrier\ \mathcal{F}_{gens}$
  ⟨*proof*⟩

definition (**in** *group*) *lift-gi*
  where *lift-gi f gi* = (*if fst gi then inv* (*f* (*snd gi*)) *else f* (*snd gi*))

lemma (**in** *group*) *lift-gi-closed*:
  assumes *cl*: $f \in gens \to carrier\ G$
    and $snd\ gi \in gens$
  shows $lift\text{-}gi\ f\ gi \in carrier\ G$
⟨*proof*⟩

definition (**in** *group*) *lift*
  where *lift f w* = *m-concat* (*map* (*lift-gi f*) *w*)

lemma (**in** *group*) *lift-nil*[*simp*]: *lift f* [] = **1**
⟨*proof*⟩

lemma (**in** *group*) *lift-closed*[*simp*]:
  assumes *cl*: $f \in gens \to carrier\ G$
    and $x \in lists\ (UNIV \times gens)$
  shows $lift\ f\ x \in carrier\ G$
⟨*proof*⟩

lemma (**in** *group*) *lift-append*[*simp*]:
  assumes *cl*: $f \in gens \to carrier\ G$
    and $x \in lists\ (UNIV \times gens)$
    and $y \in lists\ (UNIV \times gens)$
  shows $lift\ f\ (x\ @\ y) = lift\ f\ x \otimes lift\ f\ y$
⟨*proof*⟩

lemma (**in** *group*) *lift-cancels-to*:
  assumes *cancels-to x y*
    and $x \in lists\ (UNIV \times gens)$
    and *cl*: $f \in gens \to carrier\ G$
  shows *lift f x* = *lift f y*
⟨*proof*⟩

lemma (**in** *group*) *lift-is-hom*:
  assumes *cl*: $f \in gens \to carrier\ G$
  shows $lift\ f \in hom\ \mathcal{F}_{gens}\ G$
⟨*proof*⟩

lemma *gens-span-free-group*:

shows $\langle\iota$ ‘ *gens*$\rangle_{\mathcal{F}_{gens}}$ = *carrier* $\mathcal{F}_{gens}$
⟨*proof*⟩

lemma (**in** *group*) *lift-is-unique*:
assumes *group G*
and *cl*: $f \in gens \rightarrow carrier\ G$
and $h \in hom\ \mathcal{F}_{gens}\ G$
and $\forall\ g \in gens.\ h\ (\iota\ g) = f\ g$
shows $\forall x \in carrier\ \mathcal{F}_{gens}.\ h\ x = lift\ f\ x$
⟨*proof*⟩

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*
⟨*proof*⟩

theorem (**in** *group*) *unit-group-unique*:
assumes *card* (*carrier G*) = *1*
shows ∃ *h*. *h* ∈ *iso G unit-group*
⟨*proof*⟩

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 $C2 = (\!|\ carrier = UNIV,\ mult = (=),\ one = True\ |\!)$

lemma [*simp*]: $(\otimes_{C2}) = (=)$
⟨*proof*⟩

lemma [*simp*]: $\mathbf{1}_{C2} = True$
⟨*proof*⟩

lemma [*simp*]: *carrier C2* = *UNIV*
⟨*proof*⟩

lemma *C2-is-group*: *group C2*
⟨*proof*⟩

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*: $\exists h.\ h \in iso\ \mathcal{F}_{\{\}}\ unit\text{-}group$
⟨*proof*⟩

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 $int\text{-}group \equiv (\!|\ carrier = carrier\ \mathcal{Z},\ monoid.mult = (+),\ one = 0::int\ |\!)$

lemma *replicate-set-eq*[*simp*]: $\forall x \in set\ xs.\ x = y \Longrightarrow xs = replicate\ (length\ xs)\ y$
⟨*proof*⟩

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

lemma *free-group-over-one-gen*: $\exists h.\ h \in iso\ \mathcal{F}_{\{()\}}\ int\text{-}group$
⟨*proof*⟩

6.3 Free Groups over isomorphic sets of generators

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

definition *lift-generator-function* :: $('a \Rightarrow 'b) \Rightarrow (bool \times 'a)\ list \Rightarrow (bool \times 'b)\ list$
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* $\in$ *iso* $\mathcal{F}_{gens1}\ \mathcal{F}_{gens2}$
⟨*proof*⟩

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

lemma (**in** *group-hom*) *restrict-hom*[*intro*!]:
  shows *restrict h* (*carrier G*) $\in$ *homr G H*
  ⟨*proof*⟩

lemma *hom-F-C2-Powerset*:
  $\exists\ f.\ bij\text{-}betw\ f\ (Pow\ X)\ (homr\ (\mathcal{F}_X)\ C2)$
⟨*proof*⟩

lemma *group-iso-betw-hom*:
  assumes *group G1* **and** *group G2*
    and *iso*: $i \in iso\ G1\ G2$
  shows $\exists\ f\ .\ bij\text{-}betw\ f\ (homr\ G2\ H)\ (homr\ G1\ H)$
⟨*proof*⟩

lemma *isomorphic-free-groups-bases-finite*:
  assumes *iso*: $i \in iso\ \mathcal{F}_X\ \mathcal{F}_Y$
    and *finite*: *finite X*
  shows $\exists f.\ bij\text{-}betw\ f\ X\ Y$
⟨*proof*⟩

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 $\neg$ *finite X*
shows $|X| =o |carrier\ \mathcal{F}_X|$
⟨*proof*⟩

theorem *isomorphic-free-groups-bases*:
assumes *iso*: $i \in iso\ \mathcal{F}_X\ \mathcal{F}_Y$
shows $\exists f.\ bij\text{-}betw\ f\ X\ Y$
⟨*proof*⟩

end

7 The Ping Pong lemma

theory *PingPongLemma*
imports
HOL−Algebra.Bij
FreeGroups
begin

The Ping Pong Lemma is a way to recognice 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 $x \in X$
shows $\mathbf{1}_{BijGroup\ X}\ x = x$
⟨*proof*⟩

lemma *other-member*:
assumes $I \neq \{\}$ **and** $i \in I$ **and** $card\ I \neq 1$
obtains j **where** $j \in I$ **and** $j \neq i$
⟨*proof*⟩

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 $act \in hom\ G\ (BijGroup\ X)$
and $g \in (I \rightarrow carrier\ G)$
and $\langle g\ `\ I\rangle_G = carrier\ G$
and *sub1*: $\forall i \in I.\ Xout\ i \subseteq X$
and *sub2*: $\forall i \in I.\ Xin\ i \subseteq X$

and *disj1*: $\forall\, i{\in}I.\ \forall\, j{\in}I.\ i \neq j \longrightarrow \mathit{Xout}\ i \cap \mathit{Xout}\ j = \{\}$
and *disj2*: $\forall\, i{\in}I.\ \forall\, j{\in}I.\ i \neq j \longrightarrow \mathit{Xin}\ i \cap \mathit{Xin}\ j = \{\}$
and *disj3*: $\forall\, i{\in}I.\ \forall\, j{\in}I.\ \mathit{Xin}\ i \cap \mathit{Xout}\ j = \{\}$
and $x \in X$
and *gencount*: $\forall\ \ i\ .\ I = \{i\} \longrightarrow (x \notin \mathit{Xout}\ i \wedge x \notin \mathit{Xin}\ i)$
and *ping*: $\forall\, i{\in}I.\ \mathit{act}\ (g\ i)\ \text{‘}\ (X - \mathit{Xout}\ i) \subseteq \mathit{Xin}\ i$
and *pong*: $\forall\, i{\in}I.\ \mathit{act}\ (\mathit{inv}_G\ (g\ i))\ \text{‘}\ (X - \mathit{Xin}\ i) \subseteq \mathit{Xout}\ i$
shows *group.lift* $G\ g \in \mathit{iso}\ (\mathcal{F}_I)\ G$
⟨*proof*⟩

end