An Example of a Cofinitary Group in Isabelle/HOL

Bart Kastermans

September 19, 2015

Abstract

We formalize the usual proof that the group generated by the function \( k \mapsto k + 1 \) on the integers gives rise to a cofinitary group.

Contents

1 Introduction 1
2 The Main Notions 3
3 The Function \( upOne \) 4
4 The Set of Functions and Normal Forms 5
5 All Elements Cofinitary Bijections. 7
6 Closed under Composition and Inverse 8
7 Conjugation with a Bijection 11
8 Bijections on \( \mathbb{N} \) 12
9 The Conclusion 15

theory CofGroups
imports Main ~~/src/HOL/Library/Nat-Bijection
begin

1 Introduction

Cofinitary groups have received a lot of attention in Set Theory. I will start by giving some references, that together give a nice view of the area. See also Kastermans [7] for my view of where the study of these groups (other
than formalization) is headed. Starting work was done by Adeleke [1], Truss [12] and [13], and Koppelberg [10]. Cameron [3] is a very nice survey. There is also work on cardinal invariants related to these groups and other almost disjoint families, see e.g. Brendle, Spinas, and Zhang [2], Hrušák, Steprans, and Zhang [5], and Kastermans and Zhang [9]. Then there is also work on constructions and descriptive complexity of these groups, see e.g. Zhang [14], Gao and Zhang [4], and Kastermans [6] and [8].

In this note we work through formalizing a basic example of a cofinitary group. We want to achieve two things by working through this example. First how to formalize some proofs from basic set-theoretic algebra, and secondly, to do some first steps in the study of formalization of this area of set theory. This is related to the work of Paulson and Grabczewski [11] on formalizing set theory, our preference however is towards using Isar resulting in a development more readable for “normal” mathematicians.

A **cofinitary group** is a subgroup $G$ of the symmetric group on $N$ (in Isabelle $nat$) such that all non-identity elements $g \in G$ have finitely many fixed points. A simple example of a cofinitary group is obtained by considering the group $G'$ a subgroup of the symmetric group on $Z$ (in Isabelle $int$ generated by the function $upOne : Z \to Z$ defined by $k \mapsto k + 1$. No element in this group other than the identity has a fixed point. Conjugating this group by any bijection $Z \to N$ gives a cofinitary group.

We will develop a workable definition of a cofinitary group (Section 2) and show that the group as described in the previous paragraph is indeed cofinitary (this takes the whole paper, but is all pulled together in Section 9). Note: formalizing the previous paragraph is all that is completed in this note.

Since this note is also written to be read by the proverbial “normal” mathematician we will sometimes remark on notations as used in Isabelle as they related to common notation. We do expect this proverbial mathematician to be somewhat flexible though. He or she will need to be flexible in reading, this is just like reading any other article; part of reading is reconstructing.

We end this introduction with a quick overview of the paper. In Section 2 we define the notion of cofinitary group. In Section 3 we define the function $upOne$ and give some of its basic properties. In Section 4 we define the set $Ex1$ which is the underlying set of the group generated by $upOne$, there we also derive a normal form theorem for the elements of this set. In Section 5 we show all elements in $Ex1$ are cofinitary bijections (cofinitary here is used in the general meaning of having finitely many fixed points). In Section 6 we show this set is closed under composition and inverse, in effect showing that it is a “cofinitary group” (cofinitary group here is in quotes, since we only define it for sets of permutations on the natural numbers). In Section 7 we show the general theorem that conjugating a permutation by a bijection
does the expected thing to the set of fixed points. In Section 8 we define the function $CONJ$ that is conjugation by $ni$-$bij$ (a bijection from $nat$ to $int$), show that it acts well with respect to the group operations, use it to define $Ex2$ which is the underlying set of the cofinitary group we are construction, and show the basic properties of $Ex2$. Finally in Section 9 we quickly show that all the work in the section before it combines to show that $Ex2$ is a cofinitary group.

2 The Main Notions

First we define the two main notions. We write $S$-$inf$ for the symmetric group on the natural numbers (we do not define this as a group, only as the set of bijections).

\textbf{definition} $S$-$inf$ :: ($nat$ $\Rightarrow$ $nat$) set
\textbf{where}
$S$-$inf$ = \{ $f$ ::($nat$ $\Rightarrow$ $nat$). $bij$ $f$ \}

Note here that $bij$ $f$ is the predicate that $f$ is a bijection. This is common notation in Isabelle, a predicate applied to an object. Related to this $inj$ $f$ means $f$ is injective, and $surj$ $f$ means $f$ is surjective. The same notation is used for function application. Next we define a function $Fix$, applying it to an object is also written by juxtaposition.

Given any function $f$ we define $Fix$ $f$ to be the set of fixed points for this function.

\textbf{definition} $Fix$ :: ($'a$ $\Rightarrow$ $'a$) $\Rightarrow$ ($'a$ set)
\textbf{where}
$Fix$ $f$ = \{ $n$ . $f$($n$) = $n$ \}

We then define a locale \textit{CofinitaryGroup} that represents the notion of a cofinitary group. An interpretation is given by giving a set of functions $nat$ $\rightarrow$ $nat$ and showing that it satisfies the identities the locale assumes. A locale is a way to collect together some information that can then later be used in a flexible way (we will not make a lot of use of that here).

\textbf{locale} \textit{CofinitaryGroup} =
\textbf{fixes}
$\mathit{dom}$ :: ($nat$ $\Rightarrow$ $nat$) set
\textbf{assumes}
\textit{type-dom} : $\mathit{dom}$ $\subseteq$ $S$-$inf$ \textbf{and}
\textit{id-com} : $\mathit{id}$ $\in$ $\mathit{dom}$ \textbf{and}
\textit{mult-closed} : $f$ $\in$ $\mathit{dom}$ $\land$ $g$ $\in$ $\mathit{dom}$ $\implies$ $f$ $\circ$ $g$ $\in$ $\mathit{dom}$ \textbf{and}
\textit{inv-closed} : $f$ $\in$ $\mathit{dom}$ $\implies$ $\mathit{inv}$ $f$ $\in$ $\mathit{dom}$ \textbf{and}
\textit{cofinitary} : $f$ $\in$ $\mathit{dom}$ $\land$ $f$ $\neq$ $\mathit{id}$ $\implies$ finite ($\mathit{Fix}$ $f$)
3 The Function \textit{upOne}

Here we define the function, \textit{upOne}, translation up by 1 and proof some of its basic properties.

\textbf{definition} \textit{upOne} :: int $\Rightarrow$ int
\textbf{where}
\textit{upOne} \ n = \ n + 1

\textbf{declare} \textit{upOne-def} [\texttt{simp}] — automated tools can use the definition

First we show that this function is a bijection. This is done in the usual two parts; we show it is injective by showing from the assumption that outputs on two numbers are equal that these two numbers are equal. Then we show it is surjective by finding the number that maps to a given number.

\textbf{lemma} \textit{inj-upOne}: \textit{inj} \ \textit{upOne}
\textbf{by} \ (\texttt{rule Fun.injI, simp})

\textbf{lemma} \textit{surj-upOne}: \textit{surj} \ \textit{upOne}
\textbf{proof} \ (\texttt{unfold Fun.surj-def, rule})
\textbf{fix} \ k::\texttt{int}
\textbf{show} $\exists \ m. \ k = \textit{upOne} \ m$
\textbf{by} \ (\texttt{rule exI[of $\lambda l. \ k = \textit{upOne} \ l$ $k - 1$], simp})
\textbf{qed}

\textbf{theorem} \textit{bij-upOne}: \textit{bij} \ \textit{upOne}
\textbf{by} \ (\texttt{unfold bij-def, rule conjI[OF inj-upOne surj-upOne]})

Now we show that the set of fixed points of \textit{upOne} is empty. We show this in two steps, first we show that no number is a fixed point, and then derive from this that the set of fixed points is empty.

\textbf{lemma} \textit{no-fix-upOne}: \textit{upOne} \ n $\neq$ \ n
\textbf{proof} \ (\texttt{rule notI})
\textbf{assume} \ \textit{upOne} \ n = \ n
\textbf{with} \ \textit{upOne-def} \ \textbf{have} \ n+1 = \ n \ \textbf{by} \ \texttt{simp}
\textbf{thus} \ False \ \textbf{by} \ \texttt{auto}
\textbf{qed}

\textbf{theorem} \textit{Fix \ textit{upOne}} = \ {\}
\textbf{proof} —
\textbf{from} \ \textit{Fix-def[of \ textit{upOne}]}
\textbf{have} \ \textit{Fix \ textit{upOne}} = \ {\ n . \ \textit{upOne} \ n = \ n} \ \textbf{by} \ \texttt{auto}
\textbf{with} \ \textit{no-fix-upOne} \ \textbf{have} \ \textit{Fix \ textit{upOne}} = \ {\ n . \ False} \ \textbf{by} \ \texttt{auto}
\textbf{with} \ \textit{Set.empty-def} \ \textbf{show} \ \textit{Fix \ textit{upOne}} = \ {} \ \textbf{by} \ \texttt{auto}
\textbf{qed}

Finally we derive the equation for the inverse of \textit{upOne}. The rule we use references \textit{Hilbert-Choice} since the \textit{inv} operator, the operator that gives an inverse of a function, is defined using Hilbert’s choice operator.
lemma inv-upOne-eq: \((\text{inv upOne}) (n::\text{int}) = n - 1\)

proof
  fix \(n :: \text{int}\)
  have \(((\text{inv upOne}) \circ \text{upOne}) (n - 1) = (\text{inv upOne}) n\) by simp
  with inj-upOne and Hilbert-Choice.inv-o-cancel
  show \((\text{inv upOne}) n = n - 1\) by auto
qed

We can also show this quickly using Hilbert Choice.inv_eq properly instantiated: \(\text{upOne} (n - 1) = n \Rightarrow \text{inv upOne} n = n - 1\).

lemma \((\text{inv upOne}) n = n - 1\)
by (rule Hilbert-Choice.inv-f-eq[of upOne n - 1 n, OF inj-upOne, simp])

4 The Set of Functions and Normal Forms

We define the set \(Ex1\) of all powers of \(\text{upOne}\) and study some of its properties, note that this is the group generated by \(\text{upOne}\) (in Section 6 we prove it closed under composition and inverse). In Section 5 we show that all its elements are cofinitary and bijections (bijections with finitely many fixed points). Note that this is not a cofinitary group, since our definition requires the group to be a subset of \(S_{\text{inf}}\)

inductive-set Ex1 :: (int ⇒ int) set where
base-func: \(\text{upOne} \in \text{Ex1}\) |
comp-func: \(f \in \text{Ex1} \Longrightarrow (\text{upOne} \circ f) \in \text{Ex1}\) |
comp-inv: \(f \in \text{Ex1} \Longrightarrow ((\text{inv upOne}) \circ f) \in \text{Ex1}\)

We start by showing a normal form for elements in this set.

lemma Ex1-Normal-form-part1: \(f \in \text{Ex1} \Longrightarrow \exists k. \forall n. f(n) = n + k\)
proof (rule Ex1.induct[of f], blast)
  — blast takes care of the first goal which is formal noise
  assume \(f \in \text{Ex1}\)
  have \(\forall n. \text{upOne} n = n + 1\) by simp
  with HOL.eqI show \(\exists k. \forall n. \text{upOne} n = n + k\) by auto
next
  fix \(fa:: \text{int} \Rightarrow \text{int}\)
  assume fa-k: \(\exists k. \forall n. fa n = n + k\)
  thus \(\exists k. \forall n. (\text{upOne} \circ fa) n = n + k\) by auto
next
  fix \(fa:: \text{int} \Rightarrow \text{int}\)
  assume fa-k: \(\exists k. \forall n. fa n = n + k\)
  from inv-upOne-eq have \(\forall n. (\text{inv upOne}) n = n - 1\) by auto
  with fa-k show \(\exists k. \forall n. (\text{inv upOne} \circ fa) n = n + k\) by auto
qed

Now we'll show the other direction. Then we apply rule int-induct which allows us to do the induction by first showing it true for \(k = 1\), then showing
that if true for \( k = i \) it is also true for \( k = i + 1 \) and finally showing that if true for \( k = i \) then it is also true for \( k = i - 1 \).

All proofs are fairly straightforward and use extensionality for functions. In the base case we are just dealing with \( \text{upOne} \). In the other cases we define the function \( ?h \) which satisfies the induction hypothesis. Then \( f \) is obtained from this by adding or subtracting one pointwise.

In this proof we use some pattern matching to save on writing. In the statement of the theorem, we match the theorem against \( ?P_k \) thereby defining the predicate \( ?P \).

**Lemma Ex1-Normal-form-part2:**

\[
(\forall f. (\forall n. f n = n + k) \rightarrow f \in \text{Ex1}) \text{ (is } ?P \text{)}
\]

**proof** (rule int-induct [of ?P 1])

show \( \forall f. (\forall n. f n = n + 1) \rightarrow f \in \text{Ex1} \)

proof

fix \( f :: \text{int} \rightarrow \text{int} \)

show \( (\forall n. f n = n + 1) \rightarrow f \in \text{Ex1} \)

proof

assume \( \forall n. f n = n + 1 \)

hence \( \forall n. f n = \text{upOne} n \) by auto

with fun-eq-iff[of f upOne, THEN sym]

have \( f = \text{upOne} \) by auto

with Ex1.base-func show \( f \in \text{Ex1} \) by auto

qed

next

fix \( i :: \text{int} \)

assume \( 1 \leq i \)

assume induct-hyp: \( \forall f. (\forall n. f n = n + i) \rightarrow f \in \text{Ex1} \)

show \( \forall f. (\forall n. f n = n + (i + 1)) \rightarrow f \in \text{Ex1} \)

proof

fix \( f :: \text{int} \rightarrow \text{int} \)

show \( (\forall n. f n = n + (i + 1)) \rightarrow f \in \text{Ex1} \)

proof

assume f-eq: \( \forall n. f n = n + (i + 1) \)

let \( ?h = \lambda n. n + i \)

from induct-hyp have \( h-\text{Ex1}: ?h \in \text{Ex1} \) by auto

from f-eq have \( \forall n. f n = \text{upOne} (?h n) \) by (unfold upOne-def, auto)

hence \( \forall n. f n = (\text{upOne} \circ ?h) n \) by auto

with fun-eq-iff[THEN sym, of f upOne o ?h]

have \( f = \text{upOne} \circ ?h \) by auto

with h-Ex1 and Ex1.comp-func[of ?h] show \( f \in \text{Ex1} \) by auto

qed

next

fix \( i :: \text{int} \)

assume \( i \leq 1 \)

assume induct-hyp: \( \forall f. (\forall n. f n = n + i) \rightarrow f \in \text{Ex1} \)
show \( \forall f. (\forall n. f n = n + (i - 1)) \rightarrow f \in \text{Ex1} \)

proof

fix \( f :: \text{int} \rightarrow \text{int} \)

show \( (\forall n. f n = n + (i - 1)) \rightarrow f \in \text{Ex1} \)

proof

assume \( f\text{-eq: } \forall n. f n = n + (i - 1) \)

let \( h = \lambda n. n + i \)

from induct-hyp have \( \text{h-Ex1: } ?h \in \text{Ex1} \) by auto

from \( \text{inv-upOne-eq and f-eq} \)

have \( \forall n. f n = (\text{inv upOne} \circ ?h) n \) by auto

hence \( \forall n. f n = (\text{inv upOne} \circ ?h) n \) by auto

with \( \text{fun-eq-iff[THEN sgm, of f inv upOne o ?h]} \)

have \( f = \text{inv upOne} \circ ?h \) by auto

with \( \text{h-Ex1 and Ex1.comp-inv[of ?h]} \)

show \( f \in \text{Ex1} \) by auto

qed

qed

Combining the two directions we get the normal form theorem.

**theorem** \( \text{Ex1-Normal-form}: (f \in \text{Ex1}) = (\exists k. \forall n. f(n) = n + k) \)

proof

assume \( f \in \text{Ex1} \)

with \( \text{Ex1-Normal-form-part1 [of f]} \)

show \( (\exists k. \forall n. f(n) = n + k) \) by auto

next

assume \( \exists k. \forall n. f(n) = n + k \)

with \( \text{Ex1-Normal-form-part2} \)

show \( f \in \text{Ex1} \) by auto

qed

5 All Elements Cofinitary Bijections.

We now show all elements in \( \text{CofGroups.Ex1} \) are bijections, Theorem \( \text{all-bij} \), and have no fixed points, Theorem \( \text{no-fixed-pt} \).

**theorem** \( \text{all-bij: } f \in \text{Ex1} \implies \text{bij f} \)

proof (unfold \( \text{bij-def} \))

assume \( f \in \text{Ex1} \)

with \( \text{Ex1-Normal-form} \)

obtain \( k \) where \( f\text{-eq: } \forall n. f n = n + k \) by auto

show \( \text{inj f \land surj f} \)

proof (rule conjI)

show \( \text{INJ: inj f} \)

proof (rule injI)

fix \( n m \)

assume \( f n = f m \)

with \( f\text{-eq} \) have \( n + k = m + k \) by auto

thus \( n = m \) by auto
theorem no-fixed-pt:
assumes f-Ex1: f ∈ Ex1
and f-not-id: f ≠ id
shows Fix f = {}
proof −
— we start by proving an easy general fact
have f-eq-then-id: (∀ n. f(n) = n) ⇒ f = id
proof −
assume f-prop : ∀ n. f(n) = n
have (f x = id x) = (f x = x) by simp
hence (∀ x. (f x = id x)) = (∀ x. (f x = x)) by simp
with fun-eq-iff[THEN sym, of f id] and f-prop show f = id by auto
qed
from f-Ex1 and Ex1-Normal-form have ∃ k. ∀ n. f(n) = n + k by auto
hence k = 0 ⇒ ∀ n. f(n) = n by auto
with f-eq-then-id and f-not-id have k ≠ 0 by auto
with k-prop have ∀ n. f(n) ≠ n by auto
moreover
from Fix-def[of f] have Fix f = { n . f(n) = n} by auto
ultimately have Fix f = {n. False} by auto
with Set.empty-def show Fix f = {} by auto
qed

6 Closed under Composition and Inverse

We start by showing that this set is closed under composition. These facts can later be conjugated to easily obtain the corresponding results for the group on the natural numbers.

theorem closed-comp: f ∈ Ex1 ∧ g ∈ Ex1 ⇒ f ◦ g ∈ Ex1
proof (rule Ex1.induct[of f], blast)
assume f ∈ Ex1 ∧ g ∈ Ex1
with Ex1.comp-func[of g] show upOne ◦ g ∈ Ex1 by auto
next
fix fa
assume fa ◦ g ∈ Ex1
with \textit{Ex1}\textunderscore\textit{comp-func} \ of \fa \circ g
and \textit{Fun}\textunderscore\textit{o-assoc} \ of \upOne \fa \circ g
show \upOne \circ \fa \circ g \in \textit{Ex1} \ by \textit{auto}

next
fix \fa
assume \fa \circ g \in \textit{Ex1}
with \textit{Ex1}\textunderscore\textit{comp-inv} \ of \fa \circ g
and \textit{Fun}\textunderscore\textit{o-assoc} \ of \inv \upOne \fa \circ g
show \left( \inv \upOne \right) \circ \fa \circ g \in \textit{Ex1} \ by \textit{auto}

qed

Now we show the set is closed under inverses. This is done by an induction
on the definition of \textit{CofGroups}\textit{.Ex1} only using the normal form theorem and
rewriting of expressions.

\begin{varquote}
\begin{verbatim}
theorem closed-inv: \( f \in \textit{Ex1} \implies \inv f \in \textit{Ex1} \)
proof (rule \textit{Ex1}\textunderscore\textit{induct} \[\of f\], blast)
assume \( f \in \textit{Ex1} \)
show \( \inv \upOne \in \textit{Ex1} \) (is \( \textit{?right} \in \textit{Ex1} \))
proof –
let \( \textit{?left} = \inv \upOne \circ \left( \inv \upOne \circ \upOne \right) \)
{ from \textit{Ex1}\textunderscore\textit{comp-inv} and \textit{Ex1}\textunderscore\textit{base-func} have \( \textit{?left} \in \textit{Ex1} \) by auto }
moreover
{ from \textit{bij-upOne} and \textit{bij-is-inj} have \( \textit{inj} \upOne \) by auto
hence \( \inv \upOne \circ \upOne = \id \) by auto
hence \( \textit{?left} = \textit{?right} \) by auto }
ultimately
show \( \textit{?thesis} \) by auto

qed

next
fix \( f \)
assume \( f\text{-Ex1}: f \in \textit{Ex1} \)
from \( f\text{-Ex1} \) and \textit{Ex1}\textunderscore\textit{Normal-form}
obtain \( k \) where \( f\text{-eq}: \forall n. f n = n + k \) by auto

show \( \inv (\upOne \circ f) \in \textit{Ex1} \)
proof –
let \( \textit{?ic} = \inv (\upOne \circ f) \)
let \( \textit{?ci} = \inv f \circ \inv \upOne \)
{ — first we get an expression for \( \inv f \circ \inv \upOne \)
{ from \textit{all-bij} and \( f\text{-Ex1} \) have \( \textit{bij} \ f \) by auto
with \( \textit{bij-is-inj} \) have \( \textit{inj} f: \inj f \) by auto
have \( \forall n. \inv f n = n - k \)
proof

\end{verbatim}
\end{varquote}
fix n
from f-eq have f (n − k) = n by auto
with inv-f-eq[of f n−k n] and inj-f
show inv f n = n−k by auto
qed
with inv-upOne-eq
have ∀n. ?ci n = n − k − 1 by auto
hence ∀n. ?ci n = n + (−1 − k) by arith
}
moreover
— then we check that this implies inv f ◦ inv upOne is
— a member of CofGroups.Ex1
{ from Ex1-Normal-form-part2[of −1 − k]
  have (∀f. ((∀n. f n = n + (−1 − k)) → f ∈ Ex1)) by auto
}
ultimately
have ?ci ∈ Ex1 by auto
}
moreover
{ from f-Ex1 all-bij have bij f by auto
  with bij-upOne and o-inv-distrib[THEN sym]
  have ?ci = ?ic by auto
}
ultimately show ?thesis by auto
qed
next
fix f
assume f-Ex1: f ∈ Ex1
with Ex1-Normal-form
obtain k where f-eq: ∀n. f n = n + k by auto

show inv (inv upOne ◦ f) ∈ Ex1
proof —
let ?ic = inv (inv upOne ◦ f)
let ?c = inv f ◦ upOne
{
from all-bij and f-Ex1 have bij f by auto
with bij-is-inj have inj-f: inj f by auto
have ∀n. inv f n = n − k
proof
  fix n
  from f-eq have f (n − k) = n by auto
  with inv-f-eq[of f n−k n] and inj-f
  show inv f n = n−k by auto
qed
with upOne-def
have ∀n. (inv f ◦ upOne) n = n − k + 1 by auto

hence \( \forall n. (\text{inv } f \circ \text{upOne})\ n = n + (1 - k) \) by arith

moreover

from Ex1-Normal-form-part2[of 1 - k]

have \((\forall f. ((\forall n. f\ n = n + (1 - k)) \rightarrow f \in \text{Ex1}))\) by auto

ultimately

have \(?c \in \text{Ex1}\) by auto

moreover

{ from f-Ex1 all-bij and bij-is-inj have bij f by auto

moreover from bij-upOne and bij-imp-bij-inv have bij (inv upOne) by auto

moreover note o-inv-distrib[THEN sym]

ultimately have \(\text{inv } f \circ \text{inv upOne} = \text{inv upOne} \circ f\) by auto

moreover from bij-upOne and inv-inv-eq

have \(\text{inv upOne} = \text{upOne}\) by auto

ultimately

have \(?c = ?ic\) by auto

}

ultimately show \(?\text{thesis}\) by auto

qed

qed

7 Conjugation with a Bijection

An abbreviation of the bijection from the natural numbers to the integers defined in the library. This will be used to coerce the functions above to be on the natural numbers.

abbreviation ni-bij == int-decode

lemma bij-f-o-inf-f: bij f \implies f \circ \text{inv } f = \text{id}

unfolding bij-def surj-iff by simp

The following theorem is a key theorem in showing that the group we are interested in is cofinitary. It states that when you conjugate a function with a bijection the fixed points get mapped over.

theorem conj-fix-pt: \(\forall f::(\text{'a} \Rightarrow \text{'b}). \forall g::(\text{'b} \Rightarrow \text{'b}). (\text{bij } f)\)

\implies ((\text{inv } f)\ (\text{Fix } g)) = \text{Fix } ((\text{inv } f) \circ g \circ f)\)

proof -

fix f::\text{'a} \Rightarrow \text{'b}

assume bij-f: bij f

with bij-def have inj-f: inj f by auto

fix g::\text{'b} \Rightarrow \text{'b}
show \((\text{inv } f)\)'(\text{Fix } g) = \text{Fix } ((\text{inv } f) \circ g \circ f) \\
thm \text{set-eq-subset[of } ((\text{inv } f)\)'(\text{Fix } g) \text{Fix}((\text{inv } f) \circ g \circ f)] \\
proof 
  show (\text{inv } f)\)'(\text{Fix } g) \subseteq \text{Fix } ((\text{inv } f) \circ g \circ f) \\
  proof 
    fix \(x\) 
    assume \(x \in (\text{inv } f)\)'(\text{Fix } g) 
    with image-def have \(\exists y \in \text{Fix } g. \ x = (\text{inv } f) \ y\) by auto 
    from this obtain \(y\) where \(y\)-prop: \(y \in \text{Fix } g \land x = (\text{inv } f) \ y\) by auto 
    hence \(x = (\text{inv } f) \ y\) . 
    hence \(f \ x = (f \circ \text{inv } f) \ y\) by auto 
    with bij-f and bij-f-o-inf-f[of f] have \(f\)-x-y: \(f \ x = y\) by auto 
    hence \((\text{inv } f) \circ g \circ f) \ x = x\) by auto 
    with Fix-def[of f \circ g \circ f] have show \(x \in \text{Fix } ((\text{inv } f) \circ g \circ f)\) by auto 
    qed 
next 
  show \text{Fix } ((\text{inv } f) \circ g \circ f) \subseteq (\text{inv } f)\)'(\text{Fix } g) 
  proof 
    fix \(x\) 
    assume \(x \in \text{Fix } ((\text{inv } f) \circ g \circ f)\) 
    with Fix-def[of inv f \circ g \circ f] have \(x\)-fix: \((\text{inv } f) \circ g \circ f) \ x = x\) by auto 
    hence \((\text{inv } f) \ (g (f x))) = x\) by auto 
    hence \(\exists y. (\text{inv } f) \ y = x\) by auto 
    from this obtain \(y\) where \(y\)-inf-f-y: \(x = (\text{inv } f) \ y\) by auto 
    with \(x\)-fix have \((\text{inv } f) \circ g \circ f) ((\text{inv } f) \ y) = (\text{inv } f) \ y\) by auto 
    hence \((f \circ \text{inv } f) \circ g \circ f \circ \text{inv } f) \ y = (f \circ \text{inv } f) \ y\) by auto 
    with o-assoc have \((f \circ \text{inv } f) \circ g \circ (f \circ \text{inv } f)) \ y = (f \circ \text{inv } f) \ y\) by auto 
    with bij-f and bij-f-o-inf-f[of f] have \(y \in \text{Fix } g\) by auto 
    with \(x\)-inf-f-y show \(x \in ((\text{inv } f)\)'(\text{Fix } g)\) by auto 
    qed 
  qed 
qed 

8 Bijections on \(\mathbb{N}\) 

In this section we define the subset \(\text{Ex2}\) of \(\text{S-inf}\) that is the conjugate of \(\text{CofGroups.Ex1 bij ni-bij}\), and show its basic properties. 

\(\text{CONJ}\) is the function that will conjugate \(\text{CofGroups.Ex1}\) to \(\text{Ex2}\).
definition CONJ :: (int ⇒ int) ⇒ (nat ⇒ nat)
where
CONJ f = (inv ni-bij) ∘ f ∘ ni-bij

declare CONJ-def [simp] — automated tools can use the definition

We quickly check that this function is of the right type, and then show three of its properties that are very useful in showing Ex2 is a group.

lemma type-CONJ: f ∈ Ex1 =⇒ (inv ni-bij) ∘ f ∘ ni-bij ∈ S-inf
proof –
assume f-Ex1: f ∈ Ex1
with all-bij have bij f by auto
with bij-int-decode and bij-comp have bij f-nibij by auto
with bij-int-decode and bij-imp-bij-inv have bij (inv ni-bij) by auto
with bij-f-nibij and bij-comp[of f ∘ ni-bij inv ni-bij]
and o-assoc[of inv ni-bij f ni-bij]
have bij ((inv ni-bij) ∘ f ∘ ni-bij) by auto
with S-inf-def show ((inv ni-bij) ∘ f ∘ ni-bij) ∈ S-inf by auto
qed

lemma inv-CONJ:
assumes bij-f: bij f
shows inv (CONJ f) = CONJ (inv f) (is ?left = ?right)
proof –
have st1: ?left = inv ((inv ni-bij) ∘ f ∘ ni-bij)
using CONJ-def by auto
from bij-int-decode and bij-imp-bij-inv
have inv-ni-bij-bij: bij (inv ni-bij) by auto
with bij-f and bij-comp have bij (inv ni-bij ∘ f) by auto
with o-inv-distrib[of inv ni-bij ∘ f ni-bij] and bij-int-decode
have inv ((inv ni-bij) ∘ f ∘ ni-bij) =
(inv ni-bij) ∘ (inv ((inv ni-bij) ∘ f)) by auto
with st1 have st2: ?left =
(inv ni-bij) ∘ (inv ((inv ni-bij) ∘ f)) by auto
from inv-ni-bij-bij and bij f and o-inv-distrib
have h1: inv (inv ni-bij ∘ f) = inv f ∘ inv (inv (ni-bij)) by auto
from bij-int-decode and inv-imp[of ni-bij]
have inv (inv ni-bij) = ni-bij by auto
with st2 and h1 have ?left = (inv ni-bij ∘ (inv f ∘ (ni-bij))) by auto
with o-assoc have ?left = inv ni-bij ∘ inv f ∘ ni-bij by auto
with CONJ-def[of inv f] show ?thesis by auto
qed

lemma comp-CONJ:
CONJ (f ∘ g) = (CONJ f) ∘ (CONJ g) (is ?left = ?right)
proof –
from bij-int-decode have surj ni-bij unfolding bij-def by auto
\[
\text{then have } \text{ni-bij} \circ (\text{inv ni-bij}) = \text{id} \text{ unfolding surj-iff by auto}
\]

moreover

\[
\text{have } \text{?left} = (\text{inv ni-bij}) \circ (f \circ g) \circ \text{ni-bij by simp}
\]

hence

\[
\text{?left} = (\text{inv ni-bij}) \circ ((f \circ \text{id}) \circ g) \circ \text{ni-bij by simp}
\]

ultimately

\[
\text{have } \text{?left} = (\text{inv ni-bij}) \circ ((f \circ (\text{ni-bij} \circ (\text{inv ni-bij})))) \circ g \circ \text{ni-bij}
\]

by auto

— a simple computation using only associativity

— completes the proof

thus

\[
\text{?left} = \text{?right by (auto simp add: o-assoc)}
\]

done

lemma id-CONJ: \(\text{CONJ id} = \text{id}\)

proof (unfold \text{CONJ-def})

from \text{bij-int-decode} have \text{inj ni-bij using bij-def by auto}

hence \text{inv ni-bij} \circ \text{ni-bij} = \text{id by auto}

thus \((\text{inv ni-bij} \circ \text{id}) \circ \text{ni-bij} = \text{id by auto)}

qed

We now define the group we are interested in, and show the basic facts that

together will show this is a cofinitary group.

definition \text{Ex2} :: \((\text{nat} \Rightarrow \text{nat})\) set

where

\text{Ex2} = \text{CONJ Ex1}

theorem \text{mem-Ex2-rule}: \(f \in \text{Ex2} = (\exists g. (g \in \text{Ex1} \land f = \text{CONJ g}))\)

proof

assume \(f \in \text{Ex2}\)

hence \(f \in \text{CONJ Ex1}\ using \text{Ex2-def by auto}\)

from \text{this} obtain \(g\ where g \in \text{Ex1} \land f = \text{CONJ g}\ by \text{blast}\)

thus \(\exists g. (g \in \text{Ex1} \land f = \text{CONJ g}) by auto\)

next

assume \(\exists g. (g \in \text{Ex1} \land f = \text{CONJ g})\)

with \text{Ex2-def show } f \in \text{Ex2 by auto}\n
qed

theorem \text{Ex2-cofinitary}:

assumes \text{f-Ex2}: \(f \in \text{Ex2}\)

and \text{f-nid}: \(f \neq \text{id}\)

shows \(\text{Fix } f = \{\}\)

proof

from \text{f-Ex2 and mem-Ex2-rule}

obtain \(g\ where g-\text{Ex1}: g \in \text{Ex1} and f-cg: f = \text{CONJ g by auto}\)

with \text{id-CONJ and f-nid have } g \neq \text{id by auto}\n
with g-\text{Ex1} and \text{no-fixed-pt[of g] have } \text{fg-empty: Fix } g = \{\} by auto\n
from \text{conj-fix-pt[of ni-bij g and bij-int-decode}

have \((\text{inv ni-bij}) \circ (\text{Fix } g) = \text{Fix(CONJ g) by auto}\)

with \text{fg-empty have } \{\} = \text{Fix (CONJ g) by auto}\n
14
with f-cg show Fix f = {} by auto
qed

lemma id-Ex2: id ∈ Ex2
proof –
  from Ex1-Normal-form-part2[of 0] have id ∈ Ex1 by auto
  with id-CONJ and Ex2-def and mem-Ex2-rule show thesis by auto
qed

lemma inv-Ex2: f ∈ Ex2 ⇒ (inv f) ∈ Ex2
proof –
  assume f ∈ Ex2
  with mem-Ex2-rule obtain g where g ∈ Ex1 and f = CONJ g by auto
  with closed-inv have inv g ∈ Ex1 by auto
  from ⟨f = CONJ g⟩ have if-iCg: inv f = inv (CONJ g) by auto
  from all-bij and ⟨g ∈ Ex1⟩ have bij g by auto
  with if-iCg and inv-CONJ have inv f = CONJ (inv g) by auto
  from ⟨g ∈ Ex1⟩ and closed-inv have inv g ∈ Ex1 by auto
  with ⟨inv f = CONJ (inv g)⟩ and mem-Ex2-rule show inv f ∈ Ex2 by auto
qed

lemma comp-Ex2:
  assumes f-Ex2: f ∈ Ex2 and
g-Ex2: g ∈ Ex2
  shows f ◦ g ∈ Ex2
proof –
  from f-Ex2 obtain f-1
    where f-1-Ex1: f-1 ∈ Ex1 and f = CONJ f-1
      using mem-Ex2-rule by auto
  moreover
  from g-Ex2 obtain g-1
    where g-1-Ex1: g-1 ∈ Ex1 and g = CONJ g-1
      using mem-Ex2-rule by auto
  ultimately
  have f ◦ g = (CONJ f-1) ◦ (CONJ g-1) by auto
  hence f ◦ g = CONJ (f-1 ◦ g-1) using comp-CONJ by auto
  moreover
  have f-1 ◦ g-1 ∈ Ex1 using closed-comp and f-1-Ex1 and g-1-Ex1 by auto
  ultimately
  show f ◦ g ∈ Ex2 using mem-Ex2-rule by auto
qed

9 The Conclusion

With all that we have shown we have already clearly shown Ex2 to be a cofinitary group. The formalization also shows this, we just have to refer to
the correct theorems proved above.

**interpretation** CofinitaryGroup Ex2

**proof**

show \( \text{Ex}2 \subseteq \text{S-inf} \)

**proof**

fix \( f \)

assume \( f \in \text{Ex}2 \)

with \( \text{mem-Ex2-rule} \) obtain \( g \in \text{Ex}1 \) where \( g = \text{CONJ} \ g \) by \( \text{auto} \)

with \( \text{type-CONJ} \) show \( f \in \text{S-inf} \) by \( \text{auto} \)

qed

next

from \( \text{id-Ex2} \) show \( \text{id} \in \text{Ex}2 \).

next

fix \( f, g \)

assume \( f \in \text{Ex}2 \land g \in \text{Ex}2 \)

with \( \text{comp-Ex2} \) show \( f \circ g \in \text{Ex}2 \) by \( \text{auto} \)

next

fix \( f \)

assume \( f \in \text{Ex}2 \)

with \( \text{inv-Ex2} \) show \( \text{inv} \ f \in \text{Ex}2 \) by \( \text{auto} \)

next

fix \( f \)

assume \( f \in \text{Ex}2 \land f \neq \text{id} \)

with \( \text{Ex2-cofinitary} \) have \( \text{Fix} \ f = \{ \} \) by \( \text{auto} \)

thus \( \text{finite} \ (\text{Fix} \ f) \) using \( \text{finite-def} \) by \( \text{auto} \)

qed

end

**References**


