An Example of a Cofinitary Group in Isabelle/HOL

Bart Kastermans

August 3, 2009

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

theory *CofGroups*
imports *Main*
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 andGrąbczewski [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 $\mathbb{N}$ (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 $\mathbb{Z}$ (in Isabelle *int* generated by the function $upOne : \mathbb{Z} \to \mathbb{Z}$ 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 $\mathbb{Z} \to \mathbb{N}$ 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 define a bijection *ni-bij* from the natural numbers to the integers and show some of its general properties. We also show there 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 functino *CONJ* that is conjugation by *ni-bij*, show that is 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).

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

Note here that $\textit{bij}\,f$ is the predicate that f is a bijection. This is common notation in Isabelle, a predicate applied to an object. Related to thsi $\textit{inj}\,f$ means f is injective, and $\textit{surj}\ f$ means f is surjective.

The same notation is used for functionn application. Next we define a function *Fix*, applying it to an object is also written by juxtaposition.

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

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

We then define a locale *CofinitaryGroup* that represents the notion of a cofinitary group. An interpretation is given by giving a set of functions $\textit{nat} \rightarrow \textit{nat}$ 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).

locale *CofinitaryGroup* =
  fixes
    $\textit{dom} :: (\textit{nat} \Rightarrow \textit{nat})\ \textit{set}$
  assumes
    $\textit{type-dom} : \textit{dom} \subseteq \textit{S-inf}$ **and**
    $\textit{id-com} : \textit{id} \in \textit{dom}$ **and**
    $\textit{mult-closed} : f \in \textit{dom} \wedge g \in \textit{dom} \Longrightarrow f \circ g \in \textit{dom}$ **and**
    $\textit{inv-closed} : f \in \textit{dom} \Longrightarrow \textit{inv}\,f \in \textit{dom}$ **and**
    $\textit{cofinitary} : f \in \textit{dom} \wedge f \neq \textit{id} \Longrightarrow \textit{finite}\ (\textit{Fix}\,f)$

3 The Function *upOne*

Here we define the function, *upOne*, translation up by 1 and proof some of its basic properties.

definition *upOne* :: *int* ⇒ *int*
where
upOne n = *n* + *1*

declare *upOne-def* [*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.

lemma *inj-upOne*: *inj upOne*
by (*rule Fun.injI*, *simp*)

lemma *surj-upOne*: *surj upOne*
proof (*unfold Fun.surj-def*, *rule*)
  fix *k*::*int*
  show ∃ *m*. *k* = *upOne m*
    by (*rule exI*[*of* λ*l*. *k* = *upOne l k* − *1*], *simp*)
qed

theorem *bij-upOne*: *bij upOne*
by (*unfold bij-def*, *rule conjI* [*OF inj-upOne surj-upOne*])

Now we show that the set of fixed points of *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.

lemma *no-fix-upOne*: *upOne n* ≠ *n*
proof (*rule notI*)
  assume *upOne n* = *n*
  with *upOne-def* **have** *n*+*1* = *n* **by** *simp*
  thus *False* **by** *auto*
qed

theorem *Fix upOne* = {}
proof −
  from *Fix-def*[*of upOne*]
  have *Fix upOne* = {*n* . *upOne n* = *n*} **by** *auto*
  with *no-fix-upOne* **have** *Fix upOne* = {*n* . *False*} **by** *auto*
  with *Set.empty-def* **show** *Fix upOne* = {} **by** *auto*
qed

Finally we derive the equation for the inverse of *upOne*. The rule we use references $Hilbert - Choice$ since the *inv* operator, the operator that gives an inverse of a function, is defined using Hilbert's choice operator.;

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

We can also show this quickly using Hilbert_Choice.inv_f_eq properly instantiated : $upOne\ (n - 1) = n \Longrightarrow inv\ upOne\ n = n - 1$.

lemma $(inv\ upOne)\ n = n - 1$
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 *upOne* and study some of its properties, note that this is the group generated by *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-inf*

inductive-set $Ex1 :: (int \Rightarrow int)\ set$ **where**
base-func: $upOne \in Ex1$ |
comp-func: $f \in Ex1 \Longrightarrow (upOne \circ f) \in Ex1$ |
comp-inv : $f \in Ex1 \Longrightarrow ((inv\ upOne) \circ f) \in Ex1$

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

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

```
lemma Ex1-Normal-form-part2:
  (∀f. ((∀n. f n = n + k) ⟶ f ∈ Ex1)) (is ?P k)
proof (rule int-induct [of ?P 1])
  show ∀f. (∀n. f n = n + 1) ⟶ f ∈ Ex1
  proof
    fix f:: int ⇒ int
    show (∀n. f n = n + 1) ⟶ f ∈ Ex1
    proof
      assume ∀n. f n = n + 1
      hence ∀n. f n = upOne n by auto
      with expand-fun-eq[of f upOne,THEN sym]
        have f = upOne by auto
      with Ex1.base-func show f ∈ Ex1 by auto
    qed
  qed
next
  fix i::int
  assume 1 ≤ i
  assume induct-hyp: ∀f. (∀n. f n = n + i) ⟶ f ∈ Ex1
  show ∀f. (∀n. f n = n + (i + 1)) ⟶ f ∈ Ex1
  proof
    fix f:: int ⇒ int
    show (∀n. f n = n + (i + 1)) ⟶ f ∈ Ex1
    proof
      assume f-eq: ∀n. f n = n + (i + 1)
      let ?h = λn. n + i
      from induct-hyp have h-Ex1: ?h ∈ Ex1 by auto
      from f-eq have ∀n. f n = upOne (?h n) by (unfold upOne-def,auto)
      hence ∀n. f n = (upOne ∘ ?h) n by auto
      with expand-fun-eq[THEN sym, of f upOne ∘ ?h]
        have f = upOne ∘ ?h by auto
      with h-Ex1 and Ex1.comp-func[of ?h] show f ∈ Ex1 by auto
    qed
  qed
next
  fix i::int
  assume i ≤ 1
  assume induct-hyp: ∀f. (∀n. f n = n + i) ⟶ f ∈ Ex1
```

```
  show ∀f. (∀n. f n = n + (i − 1)) ⟶ f ∈ Ex1
  proof
    fix f:: int ⇒ int
    show (∀n. f n = n + (i − 1)) ⟶ f ∈ Ex1
    proof
      assume f-eq: ∀n. f n = n + (i − 1)
      let ?h = λn. n + i
      from induct-hyp have h-Ex1: ?h ∈ Ex1 by auto
      from inv-upOne-eq and f-eq
        have ∀n. f n = (inv upOne) (?h n) by auto
      hence ∀n. f n = (inv upOne ∘ ?h) n by auto
      with expand-fun-eq[THEN sym, of f inv upOne ∘ ?h]
         have f = inv upOne ∘ ?h by auto
      with h-Ex1 and Ex1.comp-inv[of ?h] show f ∈ Ex1 by auto
    qed
  qed
qed
```

Combining the two directions we get the normal form theorem.

```
theorem Ex1-Normal-form: (f ∈ Ex1) = (∃k. ∀n. f(n) = n + k)
proof
  assume f ∈ Ex1
  with Ex1-Normal-form-part1 [of f]
    show (∃k. ∀n. f(n) = n + k) by auto
next
  assume ∃k. ∀n. f(n) = n + k
  with Ex1-Normal-form-part2
    show f ∈ Ex1 by auto
qed
```

5 All Elements Cofinitary Bijections.

We now show all elements in *CofGroups.Ex1* are bijections, Theorem *all-bij*, and have no fixed points, Theorem *no-fixed-pt*.

```
theorem all-bij: f ∈ Ex1 ⟹ bij f
proof (unfold bij-def)
  assume f ∈ Ex1
  with Ex1-Normal-form
    obtain k where f-eq:∀n. f n = n + k by auto

  show inj f ∧ surj f
  proof (rule conjI)
    show INJ: inj f
    proof (rule injI)
      fix n m
      assume f n = f m
      with f-eq have n + k = m + k by auto
      thus n = m by auto
```

```
    qed
  next

    show SURJ: surj f
    proof (unfold Fun.surj-def, rule allI)
      fix n
      from f-eq have n = f (n − k) by auto
      thus ∃ m. n = f m by (rule exI)
    qed
  qed
qed

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 expand-fun-eq[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
  then obtain k where k-prop: ∀ n. f(n) = n + k ..
  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 Ex1.comp-func [of fa ∘ g]
    and Fun.o-assoc [of upOne fa g]
    show upOne ∘ fa ∘ g ∈ Ex1 by auto
next
  fix fa
  assume fa ∘ g ∈ Ex1
  with Ex1.comp-inv [of fa ∘ g]
    and Fun.o-assoc [of inv upOne fa g]
    show (inv upOne) ∘ fa ∘ g ∈ Ex1 by auto
qed
```

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

```
theorem closed-inv: f ∈ Ex1 ⟹ inv f ∈ Ex1
proof (rule Ex1.induct [of f], blast)
  assume f ∈ Ex1
  show inv upOne ∈ Ex1 (is ?right ∈ Ex1)
  proof −
    let ?left = inv upOne ∘ (inv upOne ∘ upOne)
    {
      from Ex1.comp-inv and Ex1.base-func have ?left ∈ Ex1 by auto
    }
    moreover
    {
      from bij-upOne and bij-is-inj have inj upOne by auto
      hence inv upOne ∘ upOne = id by auto
      hence ?left = ?right by auto
    }
    ultimately
    show ?thesis by auto
  qed
next
  fix f
  assume f-Ex1: f ∈ Ex1
  from f-Ex1 and Ex1-Normal-form
  obtain k where f-eq: ∀ n. f n = n + k by auto

  show inv (upOne ∘ f) ∈ Ex1
  proof −
    let ?ic = inv (upOne ∘ f)
    let ?ci = inv f ∘ inv upOne
    {
      — first we get an expression for inv f ∘ inv 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 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 $\forall n.\ (inv\ f \circ upOne)\ n = n + (1 - k)$ **by** *arith*
moreover
from *Ex1-Normal-form-part2*[*of* $1 - k$]
have $(\forall f.\ ((\forall n.\ f\ n = n + (1 - k)) \longrightarrow f \in Ex1))$ **by** *auto*
ultimately
have $?c \in Ex1$ **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 $inv\ f \circ inv\ (inv\ upOne) = inv\ (inv\ upOne \circ f)$ **by** *auto*
moreover
from *bij-upOne* **and** *inv-inv-eq*
have $inv\ (inv\ upOne) = upOne$ **by** *auto*
ultimately
have $?c = ?ic$ **by** *auto*
}
ultimately
show *?thesis* **by** *auto*
qed
qed

7 Move onto the Natural Numbers

We define a bijection from the natural numbers to the integers. This will be used to coerce the functions above to be on the natural numbers.

definition *ni-bij*:: $nat \Rightarrow int$
where
$ni\text{-}bij\ n = (if\ ((n\ mod\ (2)) = 0)$
$then\ int\ (n\ div\ 2)$
$else\ -int\ (n\ div\ 2) - 1)$

declare *ni-bij-def* [*simp*] — automated tools can use the definition

Under this bijection the even natural numbers map to the positive integers, e.g. *ni-bij 0* is 0, *ni-bij 4* is 2. The odd natural numbers map to the negative integers, e.g. *ni-bij 1* is −1, and *ni-bij 3* is −3.

We prove a couple of simple facts on modular arithmetic that we'll use to prove properties of *ni-bij*.

lemma *mod-cases*: $(n{::}nat)\ mod\ 2 = 1 \lor n\ mod\ 2 = 0$ **by** *arith*

lemma *mod-neg*: $n\ mod\ 2 = 1 \Longrightarrow ni\text{-}bij\ n < 0$

proof −
  assume *n mod 2 = 1*
  with *ni-bij-def*
    have *eq*: *ni-bij n = −int (n div 2) − 1* **by** *auto*
  moreover
  have *−int (n div 2) − 1 < 0* **by** *arith*
  ultimately
  show *ni-bij n < 0* **by** *auto*
qed

lemma *mod-pos*: *n mod 2 = 0 ⟹ ni-bij n ≥ 0*
proof −
  assume *n mod 2 = 0*
  with *ni-bij-def*
    have *ni-bij n = int(n div 2)* **by** *auto*
  moreover
  have *int(n div 2) ≥ 0* **by** *auto*
  ultimately show *ni-bij n ≥ 0* **by** *auto*
qed

lemma *im-neg-mod*: *ni-bij n < 0 ⟹ n mod 2 = 1*
proof −
  assume *output-neg*: *ni-bij n < 0*
  have *n mod 2 ≠ 0*
  proof (*rule contrapos-nn* [*of ni-bij n ≥ 0*])
    from *mod-pos* **and** *output-neg* **show** ¬(*0 ≤ ni-bij n*) **by** *arith*
  next
    from *mod-pos* **show** *n mod 2 = 0 ⟹ ni-bij n ≥ 0* **.**
  qed
  with *mod-cases* **show** *n mod 2 = 1* **by** *auto*
qed

lemma *im-notneg-mod*: *ni-bij n ≥ 0 ⟹ n mod 2 = 0*
proof −
  assume *output-notneg*: *ni-bij n ≥ 0*
  have *n mod 2 ≠ 1*
  proof (*rule contrapos-nn* [*of ni-bij n < 0*])
    from *mod-neg* **and** *output-notneg* **show** ¬(*ni-bij n < 0*) **by** *arith*
  next
    from *mod-neg* **show** *n mod 2 = 1 ⟹ ni-bij n < 0* **.**
  qed
  with *mod-cases* **show** *n mod 2 = 0* **by** *auto*
qed

lemma *mod-rule-needed*: (*k::nat*) *mod 2 = 0 ∧ k > 0 ⟹* (*k − 1*) *mod 2 = 1*
proof −
  assume (*k::nat*) *mod 2 = 0 ∧ k > 0*
  thus (*k − 1*) *mod 2 = 1* **by** *arith*
qed

With these facts we can show *ni-bij* is a bijetion. The proof is really just a matter of (un)folding definitions, and some computatons.

```
theorem ni-bij-bij: bij ni-bij
proof (unfold bij-def, rule conjI)

  show INJ: inj ni-bij
  proof (rule injI)
    fix x::nat and y::nat
    assume eq-ass: ni-bij x = ni-bij y
    show x = y
    proof cases
      assume ni-bij x < 0
      with im-neg-mod have x-mod: x mod 2 = 1 .
      hence x-eq: ni-bij x = −int(x div 2) − 1 by simp
      moreover
      with eq-ass have ni-bij y < 0 by auto
      with im-neg-mod have y-mod: y mod 2 = 1 .
      hence ni-bij y = −int(y div 2) − 1 by simp
      ultimately
      have x div 2 = y div 2 using eq-ass by auto
      moreover
      from x-mod and y-mod have x mod 2 = y mod 2 by auto
      ultimately show x = y by arith
    next
      assume ¬(ni-bij x < 0)
      hence im-x-notneg: ni-bij x ≥ 0 by auto
      with eq-ass have ni-bij y ≥ 0 by auto
      with im-notneg-mod have y-mod: (y mod 2) = 0 .
      from im-notneg-mod and im-x-notneg have x-mod: x mod 2 = 0 .
      hence ni-bij-x-ex: ni-bij x = int(x div 2) by auto
      from y-mod
        have ni-bij y = int(y div 2) by auto
      with eq-ass and ni-bij-x-ex
        have x div 2 = y div 2 by auto
      moreover
      from x-mod and y-mod have x mod 2 = y mod 2 by auto
      ultimately show x = y by arith
    qed
  qed

next

  show SURJ: surj ni-bij
  proof (unfold Fun.surj-def, rule allI)
    fix y::int
    show ∃ x. y = ni-bij x
    proof (cases)
      assume y-pos: y ≥ 0
      let ?x = 2∗nat(y)
```

have *?x mod 2* = *0* **by** *auto*
hence *int* (*2* ∗ *nat y div 2*) = *ni-bij ?x* **by** *auto*
with *y-pos* **have** *y* = *ni-bij ?x* **by** *arith*
thus ∃ *x*. *y* = *ni-bij x* **by** (*rule exI*[*of* - *?x*])
next
assume ¬(*0* ≤ *y*)
hence *ne-y*: *y* < *0* **by** *auto*
let *?x* = (*2*∗*nat*(−*y*))− *1*
have *pos-x*: *?x* > *0*
proof −
from *ne-y* **have** −*y* > *0* **by** *auto*
hence *nat*(−*y*) > *0* **by** *auto*
hence *2*∗*nat*(−*y*) > *1* **by** *auto*
thus *?x* > *0* **by** *auto*
qed
have (*2*∗ *nat*(−*y*)) *mod* (*2*::*nat*) = (*0*::*nat*) **by** *auto*
with *mod-rule-needed* **and** *pos-x*
have (*2*∗*nat*(−*y*) − *1*) *mod* (*2*::*nat*) = (*1*::*nat*) **by** *auto*
hence *y* = *ni-bij ?x* **by** *auto*
thus ∃ *x*. *y* = *ni-bij x* **by** (*rule exI*)
qed
qed
qed

The following lemma turned out easier to prove than to find.

lemma *bij-f-o-inf-f*: *bij f* ⟹ *f* ∘ *inv f* = *id*
proof −
assume *bij-f*: *bij f*
with *bij-imp-bij-inv* **have** *bij-inv-f*: *bij* (*inv f*) **by** *auto*
with *bij-def* **have** *inj* (*inv f*) **by** *auto*
hence *iif-if-id*: *inv* (*inv f*) ∘ *inv f* = *id* **by** *auto*
from *bij-f* **and** *inv-inv-eq* **have** *inv* (*inv f*) = *f* **by** *auto*
with *iif-if-id* **show** *f* ∘ *inv f* = *id* **by** *auto*
qed

The following theorem is a key theorem is 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*: ⋀*f*::(′*a* ⇒ ′*b*). ⋀*g*::(′*b* ⇒ ′*b*). (*bij f*)
⟹ ((*inv f*)‘(*Fix g*)) = *Fix* ((*inv f*) ∘ *g* ∘ *f*)
proof −
fix *f*::′*a* ⇒ ′*b*
assume *bij-f*: *bij f*
with *bij-def* **have** *inj-f*: *inj f* **by** *auto*
fix *g*::′*b*⇒′*b*
show ((*inv f*)‘(*Fix g*)) = *Fix* ((*inv f*) ∘ *g* ∘ *f*)
thm *set-eq-subset*[*of* (*inv f*)‘(*Fix g*) *Fix*((*inv f*) ∘ *g* ∘ *f*)]
proof
show (*inv f*)‘(*Fix g*) ⊆ *Fix* ((*inv f*) ∘ *g* ∘ *f*)

proof
fix x
assume $x \in (inv\ f)`(Fix\ g)$
with *image-def* **have** $\exists\, y \in Fix\ g.\ x = (inv\ f)\ y$ **by** *auto*
from *this* **obtain** y **where** *y-prop*: $y \in Fix\ g \wedge x = (inv\ f)\ y$ **by** *auto*
hence $x = (inv\ f)\ y$ **..**
hence $f\,x = (f \circ inv\ f)\ y$ **by** *auto*
with *bij-f* **and** *bij-f-o-inf-f*[*of f*] **have** *f-x-y*: $f\,x = y$ **by** *auto*
from *y-prop* **have** $y \in Fix\ g$ **..**
with *Fix-def*[*of g*] **have** $g\ y = y$ **by** *auto*
with *f-x-y* **have** $g\ (f\,x) = f\,x$ **by** *auto*
hence $(inv\ f)\ (g\ (f\,x)) = inv\ f\ (f\,x)$ **by** *auto*
with *inv-f-f* **and** *inj-f* **have** $(inv\ f)\ (g\ (f\,x)) = x$ **by** *auto*
hence $((inv\ f) \circ g \circ f)\ x = x$ **by** *auto*
with *Fix-def*[*of inv f* $\circ$ *g* $\circ$ *f*]
show $x \in Fix\ ((inv\ f) \circ g \circ f)$ **by** *auto*
qed
next
show $Fix\ (inv\ f \circ g \circ f) \subseteq (inv\ f)`(Fix\ g)$
proof
fix x
assume $x \in Fix\ (inv\ f \circ g \circ f)$
with *Fix-def*[*of inv f* $\circ$ *g* $\circ$ *f*]
have *x-fix*: $(inv\ f \circ g \circ f)\ x = x$ **by** *auto*
hence $(inv\ f)\ (g(f(x))) = x$ **by** *auto*
hence $\exists\, y.\ (inv\ f)\ y = x$ **by** *auto*
from *this* **obtain** y **where** *x-inf-f-y*: $x = (inv\ f)\ y$ **by** *auto*
with *x-fix* **have** $(inv\ f \circ g \circ f)((inv\ f)\ y)) = (inv\ f)\ y$ **by** *auto*
hence $(f \circ inv\ f \circ g \circ f \circ inv\ f)\ (y) = (f \circ inv\ f)(y)$ **by** *auto*
with *o-assoc*
have $((f \circ inv\ f) \circ g \circ (f \circ inv\ f))\ y = (f \circ inv\ f)y$ **by** *auto*
with *bij-f* **and** *bij-f-o-inf-f*[*of f*]
have $g\ y = y$ **by** *auto*
with *Fix-def*[*of g*] **have** $y \in Fix\ g$ **by** *auto*
with *x-inf-f-y* **show** $x \in (inv\ f)`(Fix\ g)$ **by** *auto*
qed
qed
qed

8 Bijections on $\mathbb{N}$

In this section we define the subset *Ex2* of *S-inf* that is the conjugate of *CofGroups.Ex1* bij *ni-bij*, and show its basic properties.

First we prove a simple lemma that again was easier to prove than to find.

lemma *comp-bij*: $(bij\ (g{::}'a \Rightarrow 'b) \wedge bij\ (h{::}'b \Rightarrow 'c)) \Longrightarrow bij\ (h \circ g)$
proof $-$
assume $bij\ g \wedge bij\ h$
hence $bij\ g$ **and** $bij\ h$ **by** *auto*

with *bij-is-inj* **and** *bij-is-surj*
  have *inj-g*: *inj g* **and** *surj-g*: *surj g* **and** *inj-h*: *inj h*
    and *surj-h*: *surj h* **by** *auto*
show *bij* $(h \circ g)$
proof (*rule bijI*)
  show *inj* $(h \circ g)$
  proof (*rule injI*)
    fix $x\ y$
    assume $(h \circ g)\ x = (h \circ g)\ y$
    hence $h(g(x)) = h(g(y))$ **by** *auto*
    with *inj-h* **and** *inj-eq*[*of h*] **have** $g(x) = g(y)$ **by** *auto*
    with *inj-g* **and** *inj-eq*[*of g*] **show** $x = y$ **by** *auto*
  qed

  from *surj-h* **and** *surj-g* **and** *comp-surj* **show** *surj* $(h \circ g)$ **by** *auto*
qed
qed

CONJ is the function that will conjugate *CofGroups.Ex1* to *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 *ni-bij-bij* **and** *comp-bij*
    have *bij-f-nibij*: *bij* (*f* ∘ *ni-bij*) **by** *auto*
  with *ni-bij-bij* **and** *bij-imp-bij-inv* **have** *bij* (*inv ni-bij*) **by** *auto*
  with *bij-f-nibij* **and** *comp-bij*[*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 *ni-bij-bij* **and** *bij-imp-bij-inv*
    have *inv-ni-bij-bij*: *bij* (*inv ni-bij*) **by** *auto*

with *bij-f* **and** *comp-bij* **have** *bij* $(inv\ ni\text{-}bij \circ f)$ **by** *auto*
with *o-inv-distrib*[*of inv ni-bij* $\circ$ *f ni-bij*] **and** *ni-bij-bij*
have $inv\ ((inv\ ni\text{-}bij) \circ f \circ\ ni\text{-}bij) =$
$(inv\ ni\text{-}bij) \circ (inv\ ((inv\ ni\text{-}bij) \circ f))$ **by** *auto*
with *st1* **have** *st2*: $?left =$
$(inv\ ni\text{-}bij) \circ (inv\ ((inv\ ni\text{-}bij) \circ f))$ **by** *auto*
from *inv-ni-bij-bij* **and** ‹*bij f*› **and** *o-inv-distrib*
have *h1*: $inv\ (inv\ ni\text{-}bij \circ f) = inv\ f \circ inv\ (inv\ (ni\text{-}bij))$ **by** *auto*
from *ni-bij-bij* **and** *inv-inv-eq*[*of ni-bij*]
have $inv\ (inv\ ni\text{-}bij) = ni\text{-}bij$ **by** *auto*
with *st2* **and** *h1* **have** $?left = (inv\ ni\text{-}bij \circ (inv\ f \circ (\ ni\text{-}bij)))$ **by** *auto*
with *o-assoc* **have** $?left = inv\ ni\text{-}bij \circ inv\ f \circ ni\text{-}bij$ **by** *auto*
with *CONJ-def*[*of inv f*] **show** *?thesis* **by** *auto*
qed

lemma *comp-CONJ*:
$CONJ\ (f \circ g) = (CONJ\ f) \circ (CONJ\ g)$ (**is** $?left = ?right$)
proof −
from *ni-bij-bij* **have** *surj ni-bij* **using** *bij-def* **by** *auto*
with *surj-iff* **have** $ni\text{-}bij \circ (inv\ ni\text{-}bij) = id$ **by** *auto*
moreover
have $?left = (inv\ ni\text{-}bij) \circ (f \circ g) \circ ni\text{-}bij$ **by** *simp*
hence $?left = (inv\ ni\text{-}bij) \circ ((f \circ id) \circ g) \circ ni\text{-}bij$ **by** *simp*
ultimately
have $?left =$
$(inv\ ni\text{-}bij) \circ ((f \circ (ni\text{-}bij \circ (inv\ ni\text{-}bij))) \circ g) \circ ni\text{-}bij$
by *auto*
— a simple computation using only associativity
— completes the proof
thus $?left = ?right$ **by** (*auto simp add*: *o-assoc*)
qed

lemma *id-CONJ*: $CONJ\ id = id$
proof (*unfold CONJ-def*)
from *ni-bij-bij* **have** *inj ni-bij* **using** *bij-def* **by** *auto*
hence $inv\ ni\text{-}bij \circ ni\text{-}bij = id$ **by** *auto*
thus $(inv\ ni\text{-}bij \circ id) \circ ni\text{-}bij = id$ **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 *Ex2* :: $(nat \Rightarrow nat)\ set$
where
Ex2 = *CONJ‘Ex1*

theorem *mem-Ex2-rule*: $f \in Ex2 = (\exists\, g.\ (g \in Ex1 \wedge f = CONJ\ g))$
proof
assume $f \in Ex2$
hence $f \in CONJ`Ex1$ **using** *Ex2-def* **by** *auto*

```
  from this obtain g where g ∈ Ex1 ∧ f = CONJ g by blast
  thus ∃ g. (g ∈ Ex1 ∧ f = CONJ g) by auto
next
  assume ∃ g. (g ∈ Ex1 ∧ f = CONJ g)
  with Ex2-def show f ∈ Ex2 by auto
qed

theorem Ex2-cofinitary:
  assumes f-Ex2: f ∈ Ex2
  and f-nid: f ≠ id
  shows Fix f = {}
proof −
  from f-Ex2 and mem-Ex2-rule
  obtain g where g-Ex1: g ∈ Ex1 and f-cg: f = CONJ g by auto
  with id-CONJ and f-nid have g ≠ id by auto
  with g-Ex1 and no-fixed-pt[of g] have fg-empty: Fix g = {} by auto
  from conj-fix-pt[of ni-bij g] and ni-bij-bij
  have (inv ni-bij)‘(Fix g) = Fix(CONJ g) by auto
  with fg-empty have {} = Fix (CONJ g) by auto
  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*: $g\text{-}1 \in Ex1$ **and** $g = CONJ\ g\text{-}1$
using *mem-Ex2-rule* **by** *auto*
ultimately
have $f \circ g = (CONJ\ f\text{-}1) \circ (CONJ\ g\text{-}1)$ **by** *auto*
hence $f \circ g = CONJ\ (f\text{-}1 \circ g\text{-}1)$ **using** *comp-CONJ* **by** *auto*
moreover
have $f\text{-}1 \circ g\text{-}1 \in Ex1$ **using** *closed-comp* **and** *f-1-Ex1* **and** *g-1-Ex1* **by** *auto*
ultimately
show $f \circ g \in Ex2$ **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 $Ex2 \subseteq S\text{-}inf$
proof
fix *f*
assume $f \in Ex2$
with *mem-Ex2-rule* **obtain** *g* **where** $g \in Ex1$ **and** $f = CONJ\ g$ **by** *auto*
with *type-CONJ* **show** $f \in S\text{-}inf$ **by** *auto*
qed
next
from *id-Ex2* **show** $id \in Ex2$ **.**
next
fix *f g*
assume $f \in Ex2 \wedge g \in Ex2$
with *comp-Ex2* **show** $f \circ g \in Ex2$ **by** *auto*
next
fix *f*
assume $f \in Ex2$
with *inv-Ex2* **show** $inv\ f \in Ex2$ **by** *auto*
next
fix *f*
assume $f \in Ex2 \wedge f \neq id$
with *Ex2-cofinitary* **have** $Fix\ f = \{\}$ **by** *auto*
thus $finite\ (Fix\ f)$ **using** *finite-def* **by** *auto*
qed

end

References

[1] S. A. Adeleke. Embeddings of infinite permutation groups in sharp, highly transitive, and homogeneous groups. *Proc. Edinburgh Math. Soc. (2)*, 31(2):169–178, 1988.

[2] J. Brendle, O. Spinas, and Y. Zhang. Uniformity of the meager ideal and maximal cofinitary groups. *J. Algebra*, 232(1):209–225, 2000.

[3] P. J. Cameron. Cofinitary permutation groups. *Bull. London Math. Soc.*, 28(2):113–140, 1996.

[4] S. Gao and Y. Zhang. Definable sets of generators in maximal cofinitary groups. *Adv. Math.*, 217(2):814–832, 2008.

[5] M. Hrušák, J. Steprans, and Y. Zhang. Cofinitary groups, almost disjoint and dominating families. *J. Symbolic Logic*, 66(3):1259–1276, 2001.

[6] B. Kastermans. Isomorphism types of maximal cofinitary groups. to appear in the Bulletin of Symbolic Logic.

[7] B. Kastermans. Questions on cofinitary groups. in preparation.

[8] B. Kastermans. The complexity of maximal cofinitary groups. *Proceeding American Mathematical Society*, 137(1):307–316, 2009.

[9] B. Kastermans and Y. Zhang. Cardinal invariants related to permutation groups. *Ann. Pure Appl. Logic*, 143:139–146i, 2006.

[10] S. Koppelberg. Groups of permutations with few fixed points. *Algebra Universalis*, 17(1):50–64, 1983.

[11] L. C. Paulson and K. Grąbczewski. Mechanizing set theory. Cardinal arithmetic and the axiom of choice. *J. Automat. Reason.*, 17(3):291–323, 1996.

[12] J. K. Truss. Embeddings of infinite permutation groups. In *Proceedings of groups—St. Andrews 1985*, volume 121 of *London Math. Soc. Lecture Note Ser.*, pages 335–351, Cambridge, 1986. Cambridge Univ. Press.

[13] J. K. Truss. Joint embeddings of infinite permutation groups. In *Advances in algebra and model theory (Essen, 1994; Dresden, 1995)*, volume 9 of *Algebra Logic Appl.*, pages 121–134. Gordon and Breach, Amsterdam, 1997.

[14] Y. Zhang. Constructing a maximal cofinitary group. *Lobachevskii J. Math.*, 12:73–81 (electronic), 2003.