The Twelvefold Way

Lukas Bulwahn

October 13, 2025

Abstract

This entry provides all cardinality theorems of the Twelvefold Way. The Twelvefold Way [1, 5, 6] systematically classifies twelve related combinatorial problems concerning two finite sets, which include counting permutations, combinations, multisets, set partitions and number partitions. This development builds upon the existing formal developments [2, 3, 4] with cardinality theorems for those structures. It provides twelve bijections from the various structures to different equivalence classes on finite functions, and hence, proves cardinality formulae for these equivalence classes on finite functions.

Contents

1 Preliminaries

theory *Preliminaries*
imports
Main
HOL−Library.Multiset
HOL−Library.FuncSet
HOL−Combinatorics.Permutations
HOL−ex.Birthday-Paradox
Card-Partitions.Card-Partitions
Bell-Numbers-Spivey.Bell-Numbers
Card-Multisets.Card-Multisets
Card-Number-Partitions.Card-Number-Partitions
begin

1.1 Additions to Finite Set Theory

```
lemma subset-with-given-card-exists:
  assumes n ≤ card A
  shows ∃B ⊆ A. card B = n
using assms proof (induct n)
  case 0
  then show ?case by auto
next
  case (Suc n)
  from this obtain B where B ⊆ A card B = n by auto
  from this ‹B ⊆ A› ‹card B = n› have card B < card A
    using Suc.prems by linarith
  from ‹Suc n ≤ card A› card.infinite have finite A by force
  from this ‹B ⊆ A› finite-subset have finite B by blast
  from ‹card B < card A› ‹B ⊆ A› obtain a where a ∈ A a ∉ B
    by (metis less-irrefl subsetI subset-antisym)
  have insert a B ⊆ A card (insert a B) = Suc n
    using ‹finite B› ‹a ∈ A› ‹a ∉ B› ‹B ⊆ A› ‹card B = n› by auto
  then show ?case by blast
qed
```

1.2 Additions to Equiv Relation Theory

```
lemmas univ-commute′ = univ-commute[unfolded Equiv-Relations.proj-def]
```

lemma *univ-predicate-impl-forall*:
  assumes *equiv A R*
  assumes *P respects R*
  assumes $X \in A$ // R
  assumes *univ P X*
  shows $\forall x \in X.\ P\ x$
proof −
  from *assms*(*1*,*3*) **obtain** x **where** $x \in X$
    by (*metis equiv-class-self quotientE*)
  from ‹$x \in X$› *assms*(*1*,*3*) **have** $X = R$ ‘‘ $\{x\}$
    by (*metis Image-singleton-iff equiv-class-eq quotientE*)
  from *assms*(*1*,*2*,*4*) *this* **show** *?thesis*
    using *equiv-class-eq-iff univ-commute′* **by** *fastforce*
qed

lemma *univ-preserves-predicate*:
  assumes *equiv A r*
  assumes *P respects r*
  shows $\{x \in A.\ P\ x\}$ // r = $\{X \in A$ // $r.$ *univ P X*$\}$
proof
  show $\{x \in A.\ P\ x\}$ // $r \subseteq \{X \in A$ // $r.$ *univ P X*$\}$
  proof
    fix X
    assume $X \in \{x \in A.\ P\ x\}$ // r
    from *this* **obtain** x **where** $x \in \{x \in A.\ P\ x\}$ **and** $X = r$ ‘‘ $\{x\}$
      using *quotientE* **by** *blast*
    have $X \in A$ // r
      using ‹$X = r$ ‘‘ $\{x\}$› ‹$x \in \{x \in A.\ P\ x\}$›
      by (*auto intro*: *quotientI*)
    moreover have *univ P X*
      using ‹$X = r$ ‘‘ $\{x\}$› ‹$x \in \{x \in A.\ P\ x\}$› *assms*
      by (*simp add*: *proj-def*[*symmetric*] *univ-commute*)
    ultimately show $X \in \{X \in A$ // $r.$ *univ P X*$\}$ **by** *auto*
  qed
next
  show $\{X \in A$ // $r.$ *univ P X*$\} \subseteq \{x \in A.\ P\ x\}$ // r
  proof
    fix X
    assume $X \in \{X \in A$ // $r.$ *univ P X*$\}$
    from *this* **have** $X \in A$ // r **and** *univ P X* **by** *auto*
    from ‹$X \in A$ // r› **obtain** x **where** $x \in A$ **and** $X = r$ ‘‘ $\{x\}$
      using *quotientE* **by** *blast*
    have $x \in \{x \in A.\ P\ x\}$
      using ‹$x \in A$› ‹$X = r$ ‘‘ $\{x\}$› ‹*univ P X*› *assms*
      by (*simp add*: *proj-def*[*symmetric*] *univ-commute*)
    from *this* **show** $X \in \{x \in A.\ P\ x\}$ // r
      using ‹$X = r$ ‘‘ $\{x\}$› **by** (*auto intro*: *quotientI*)
  qed

qed

lemma *Union-quotient-restricted*:
  assumes *equiv A r*
  assumes *P respects r*
  shows $\bigcup(\{x \in A.\ P\ x\}\ //\ r) = \{x \in A.\ P\ x\}$
proof
  show $\bigcup(\{x \in A.\ P\ x\}\ //\ r) \subseteq \{x \in A.\ P\ x\}$
  proof
    fix x
    assume $x \in \bigcup(\{x \in A.\ P\ x\}\ //\ r)$
    from *this* **obtain** X **where** $x \in X$ **and** $X \in \{x \in A.\ P\ x\}\ //\ r$ **by** *blast*
    from *this* **obtain** x' **where** $X = r$ `` $\{x'\}$ **and** $x' \in \{x \in A.\ P\ x\}$
      using *quotientE* **by** *blast*
    from *this* ‹$x \in X$› **have** $x \in A$
      using ‹*equiv A r*› **by** (*simp add*: *equiv-class-eq-iff*)
    moreover from ‹$X = r$ `` $\{x'\}$› ‹$x \in X$› ‹$x' \in \{x \in A.\ P\ x\}$› **have** $P\ x$
      using ‹*P respects r*› *congruentD* **by** *fastforce*
    ultimately show $x \in \{x \in A.\ P\ x\}$ **by** *auto*
  qed
next
  show $\{x \in A.\ P\ x\} \subseteq \bigcup(\{x \in A.\ P\ x\}\ //\ r)$
  proof
    fix x
    assume $x \in \{x \in A.\ P\ x\}$
    from *this* **have** $x \in r$ `` $\{x\}$
      using ‹*equiv A r*› *equiv-class-self* **by** *fastforce*
    from ‹$x \in \{x \in A.\ P\ x\}$› **have** r `` $\{x\} \in \{x \in A.\ P\ x\}\ //\ r$
      by (*auto intro*: *quotientI*)
    from *this* ‹$x \in r$ `` $\{x\}$› **show** $x \in \bigcup(\{x \in A.\ P\ x\}\ //\ r)$ **by** *auto*
  qed
qed

lemma *finite-equiv-implies-finite-carrier*:
  assumes *equiv A R*
  assumes *finite* $(A\ //\ R)$
  assumes $\forall X \in A\ //\ R.$ *finite X*
  shows *finite A*
proof −
  from ‹*equiv A R*› **have** $A = \bigcup(A\ //\ R)$
    by (*simp add*: *Union-quotient*)
  from *this* ‹*finite* $(A\ //\ R)$› ‹$\forall X \in A\ //\ R.$ *finite X*› **show** *finite A*
    using *finite-Union* **by** *fastforce*
qed

lemma *finite-quotient-iff*:
  assumes *equiv A R*
  shows *finite A* $\longleftrightarrow$ (*finite* $(A\ //\ R) \wedge (\forall X \in A\ //\ R.$ *finite X*))
using *assms* **by** (*meson equiv-type finite-equiv-class finite-equiv-implies-finite-carrier*

finite-quotient)

1.2.1 Counting Sets by Splitting into Equivalence Classes

lemma *card-equiv-class-restricted*:
  assumes *finite* $\{x \in A.\ P\ x\}$
  assumes *equiv A R*
  assumes *P respects R*
  shows *card* $\{x \in A.\ P\ x\}$ = *sum card* $(\{x \in A.\ P\ x\}\ //\ R)$
proof −
  have *card* $\{x \in A.\ P\ x\}$ = *card* $(\bigcup(\{x \in A.\ P\ x\}\ //\ R))$
    using ‹*equiv A R*› ‹*P respects R*› **by** (*simp add*: *Union-quotient-restricted*)
  also have *card* $(\bigcup(\{x \in A.\ P\ x\}\ //\ R)) = (\sum C{\in}\{x \in A.\ P\ x\}\ //\ R.\ card\ C)$
  proof −
    from ‹*finite* $\{x \in A.\ P\ x\}$› **have** *finite* $(\{x \in A.\ P\ x\}\ //\ R)$
      using ‹*equiv A R*› **by** (*metis finite-imageI proj-image*)
    moreover from ‹*finite* $\{x \in A.\ P\ x\}$› **have** $\forall C{\in}\{x \in A.\ P\ x\}\ //\ R.$ *finite C*
      using ‹*equiv A R*› ‹*P respects R*› *Union-quotient-restricted*
        Union-upper finite-subset **by** *fastforce*
    moreover have $\forall C1 \in \{x \in A.\ P\ x\}\ //\ R.\ \forall C2 \in \{x \in A.\ P\ x\}\ //\ R.\ C1 \neq C2 \longrightarrow C1 \cap C2 = \{\}$
      using ‹*equiv A R*› *quotient-disj*
      by (*metis* (*no-types*, *lifting*) *mem-Collect-eq quotientE quotientI*)
    ultimately show *?thesis*
      by (*subst card-Union-disjoint*) (*auto simp*: *pairwise-def disjnt-def*)
  qed
  finally show *?thesis* **.**
qed

lemma *card-equiv-class-restricted-same-size*:
  assumes *equiv A R*
  assumes *P respects R*
  assumes $\bigwedge F.\ F \in \{x \in A.\ P\ x\}\ //\ R \Longrightarrow card\ F = k$
  shows *card* $\{x \in A.\ P\ x\} = k * card\ (\{x \in A.\ P\ x\}\ //\ R)$
proof *cases*
  assume *finite* $\{x \in A.\ P\ x\}$
  have *card* $\{x \in A.\ P\ x\}$ = *sum card* $(\{x \in A.\ P\ x\}\ //\ R)$
    using ‹*finite* $\{x \in A.\ P\ x\}$› ‹*equiv A R*› ‹*P respects R*›
    by (*simp add*: *card-equiv-class-restricted*)
  also have *sum card* $(\{x \in A.\ P\ x\}\ //\ R) = k * card\ (\{x \in A.\ P\ x\}\ //\ R)$
    by (*simp add*: ‹$\bigwedge F.\ F \in \{x \in A.\ P\ x\}\ //\ R \Longrightarrow card\ F = k$›)
  finally show *?thesis* **.**
next
  assume *infinite* $\{x \in A.\ P\ x\}$
  from *this* **have** *infinite* $(\bigcup(\{a \in A.\ P\ a\}\ //\ R))$
    using ‹*equiv A R*› ‹*P respects R*› **by** (*simp add*: *Union-quotient-restricted*)
  from *this* **have** *infinite* $(\{x \in A.\ P\ x\}\ //\ R) \vee (\exists X \in \{x \in A.\ P\ x\}\ //\ R.$ *infinite X*)
    by *auto*

from *this* **show** *?thesis*
proof
assume *infinite* ({$x \in A$. *P x*} // *R*)
from *this* ‹*infinite* {$x \in A$. *P x*}› **show** *?thesis* **by** *simp*
next
assume $\exists X \in$ {$x \in A$. *P x*} // *R*. *infinite X*
from *this* ‹*infinite* {$x \in A$. *P x*}› **show** *?thesis*
using ‹$\bigwedge F$. $F \in$ {$x \in A$. *P x*} // $R \Longrightarrow$ *card* $F = k$› *card.infinite* **by** *auto*
qed
qed

lemma *card-equiv-class*:
assumes *finite A*
assumes *equiv A R*
shows *card* $A =$ *sum card* (*A* // *R*)
proof −
have (λx. *True*) *respects R* **by** (*simp add*: *congruentI*)
from ‹*finite A*› ‹*equiv A R*› *this* **show** *?thesis*
using *card-equiv-class-restricted*[**where** $P{=}\lambda x$. *True*] **by** *auto*
qed

lemma *card-equiv-class-same-size*:
assumes *equiv A R*
assumes $\bigwedge F$. $F \in A$ // $R \Longrightarrow$ *card* $F = k$
shows *card* $A = k *$ *card* (*A* // *R*)
proof −
have (λx. *True*) *respects R* **by** (*simp add*: *congruentI*)
from ‹*equiv A R*› ‹$\bigwedge F$. $F \in A$ // $R \Longrightarrow$ *card* $F = k$› *this* **show** *?thesis*
using *card-equiv-class-restricted-same-size*[**where** $P{=}\lambda x$. *True*] **by** *auto*
qed

1.3 Additions to FuncSet Theory

lemma *finite-same-card-bij-on-ext-funcset*:
assumes *finite A finite B card* $A =$ *card B*
shows $\exists f$. $f \in A \rightarrow_E B \wedge$ *bij-betw f A B*
proof −
from *assms* **obtain** f' **where** f': *bij-betw* f' *A B*
using *finite-same-card-bij* **by** *auto*
define *f* **where** $\bigwedge x$. $f\ x =$ (*if* $x \in A$ *then* $f'\ x$ *else undefined*)
have $f \in A \rightarrow_E B$
using f' **unfolding** *f-def* **by** (*auto simp add*: *bij-betwE*)
moreover have *bij-betw f A B*
proof −
have *bij-betw* f' *A B* $\longleftrightarrow$ *bij-betw f A B*
unfolding *f-def* **by** (*auto intro*!: *bij-betw-cong*)
from *this* ‹*bij-betw* f' *A B*› **show** *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*

qed

lemma *card-extensional-funcset*:
  assumes *finite A*
  shows *card* $(A \rightarrow_E B)$ = *card B* ^ *card A*
using *assms* **by** (*simp add*: *card-PiE prod-constant*)

lemma *bij-betw-implies-inj-on-and-card-eq*:
  assumes *finite B*
  assumes $f \in A \rightarrow_E B$
  shows *bij-betw f A B* ⟷ *inj-on f A* ∧ *card A* = *card B*
proof
  assume *bij-betw f A B*
  from *this* **show** *inj-on f A* ∧ *card A* = *card B*
    by (*simp add*: *bij-betw-imp-inj-on bij-betw-same-card*)
next
  assume *inj-on f A* ∧ *card A* = *card B*
  from *this* **have** *inj-on f A* **and** *card A* = *card B* **by** *auto*
  from ‹$f \in A \rightarrow_E B$› **have** *f* ‘ *A* ⊆ *B* **by** *auto*
  from ‹*inj-on f A*› **have** *card* (*f* ‘ *A*) = *card A* **by** (*simp add*: *card-image*)
  from ‹*f* ‘ *A* ⊆ *B*› ‹*card A* = *card B*› *this* **have** *f* ‘ *A* = *B*
    by (*simp add*: ‹*finite B*› *card-subset-eq*)
  from ‹*inj-on f A*› *this* **show** *bij-betw f A B* **by** (*rule bij-betw-imageI*)
qed

lemma *bij-betw-implies-surj-on-and-card-eq*:
  assumes *finite A*
  assumes $f \in A \rightarrow_E B$
  shows *bij-betw f A B* ⟷ *f* ‘ *A* = *B* ∧ *card A* = *card B*
proof
  assume *bij-betw f A B*
  show *f* ‘ *A* = *B* ∧ *card A* = *card B*
    using ‹*bij-betw f A B*› *bij-betw-imp-surj-on bij-betw-same-card* **by** *blast*
next
  assume *f* ‘ *A* = *B* ∧ *card A* = *card B*
  from *this* **have** *f* ‘ *A* = *B* **and** *card A* = *card B* **by** *auto*
  from *this* **have** *inj-on f A*
    by (*simp add*: ‹*finite A*› *inj-on-iff-eq-card*)
  from *this* ‹*f* ‘ *A* = *B*› **show** *bij-betw f A B* **by** (*rule bij-betw-imageI*)
qed

1.4 Additions to Permutations Theory

lemma
  assumes $f \in A \rightarrow_E B$ *f* ‘ *A* = *B*
  assumes *p permutes B* $(\forall x. f' x = p (f x))$
  shows $(\lambda b. \{x{\in}A. f x = b\})$ ‘ *B* = $(\lambda b. \{x{\in}A. f' x = b\})$ ‘ *B*
proof
  show $(\lambda b. \{x \in A. f x = b\})$ ‘ *B* ⊆ $(\lambda b. \{x \in A. f' x = b\})$ ‘ *B*

proof
  fix X
  assume $X \in (\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ B
  from *this* **obtain** b **where** *X-eq*: $X = \{x \in A.\ f\ x = b\}$ **and** $b \in B$ **by** *blast*
  from *assms*(*3*, *4*) **have** $\bigwedge x.\ f\ x = b \longleftrightarrow f'\ x = p\ b$ **by** (*metis permutes-def*)
  from ‹p *permutes* B› *X-eq this* **have** $X = \{x \in A.\ f'\ x = p\ b\}$
    using *Collect-cong* **by** *auto*
  moreover from ‹$b \in B$› ‹p *permutes* B› **have** $p\ b \in B$
    by (*simp add*: *permutes-in-image*)
  ultimately show $X \in (\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ B **by** *blast*
qed
next
show $(\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B \subseteq (\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ B
proof
  fix X
  assume $X \in (\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ B
  from *this* **obtain** b **where** *X-eq*: $X = \{x \in A.\ f'\ x = b\}$ **and** $b \in B$ **by** *blast*
  from *assms*(*3*, *4*) **have** $\bigwedge x.\ f'\ x = b \longleftrightarrow f\ x = inv\ p\ b$
    by (*auto simp add*: *permutes-inverses*(*1*, *2*))
  from ‹p *permutes* B› *X-eq this* **have** $X = \{x \in A.\ f\ x = inv\ p\ b\}$
    using *Collect-cong* **by** *auto*
  moreover from ‹$b \in B$› ‹p *permutes* B› **have** $inv\ p\ b \in B$
    by (*simp add*: *permutes-in-image permutes-inv*)
  ultimately show $X \in (\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ B **by** *blast*
qed
qed

1.5 Additions to List Theory

The theorem *card-lists-length-eq* contains the superfluous assumption *finite* A. Here, we derive that fact without that unnecessary assumption.

lemma *lists-length-eq-Suc-eq-image-Cons*:
  $\{xs.\ set\ xs \subseteq A \wedge length\ xs = Suc\ n\} = (\lambda(x, xs).\ x\#xs)$ ‘ $(A \times \{xs.\ set\ xs \subseteq A \wedge length\ xs = n\})$
  (**is** $?A = ?B$)
proof
  show $?A \subseteq ?B$
  proof
    fix xs
    assume $xs \in ?A$
    from *this* **show** $xs \in ?B$ **by** (*cases xs*) *auto*
  qed
next
  show $?B \subseteq ?A$ **by** *auto*
qed

lemma *lists-length-eq-Suc-eq-empty-iff*:
  $\{xs.\ set\ xs \subseteq A \wedge length\ xs = Suc\ n\} = \{\} \longleftrightarrow A = \{\}$
proof (*induct n*)

```
  case 0
  have {xs. set xs ⊆ A ∧ length xs = Suc 0} = {x#[] |x. x ∈ A}
  proof
    show {[x] |x. x ∈ A} ⊆ {xs. set xs ⊆ A ∧ length xs = Suc 0} by auto
  next
    show {xs. set xs ⊆ A ∧ length xs = Suc 0} ⊆ {[x] |x. x ∈ A}
    proof
      fix xs
      assume xs ∈ {xs. set xs ⊆ A ∧ length xs = Suc 0}
      from this have set xs ⊆ A ∧ length xs = Suc 0 by simp
      from this have ∃ x. xs = [x] ∧ x ∈ A
        by (metis Suc-length-conv insert-subset length-0-conv list.set(2))
      from this show xs ∈ {[x] |x. x ∈ A} by simp
    qed
  qed
  then show ?case by simp
next
  case (Suc n)
  from this show ?case by (auto simp only: lists-length-eq-Suc-eq-image-Cons)
qed

lemma lists-length-eq-eq-empty-iff:
  {xs. set xs ⊆ A ∧ length xs = n} = {} ⟷ (A = {} ∧ n > 0)
proof (cases n)
  case 0
  then show ?thesis by auto
next
  case (Suc n)
  then show ?thesis by (auto simp only: lists-length-eq-Suc-eq-empty-iff)
qed

lemma finite-lists-length-eq-iff:
  finite {xs. set xs ⊆ A ∧ length xs = n} ⟷ (finite A ∨ n = 0)
proof
  assume finite {xs. set xs ⊆ A ∧ length xs = n}
  from this show finite A ∨ n = 0
  proof (induct n)
    case 0
    then show ?case by simp
  next
    case (Suc n)
    have inj (λ(x, xs). x#xs)
      by (auto intro: inj-onI)
    from this Suc(2) have finite (A × {xs. set xs ⊆ A ∧ length xs = n})
     using finite-imageD inj-on-subset subset-UNIV lists-length-eq-Suc-eq-image-Cons[of
A n]
      by fastforce
    from this have finite A
      by (cases A = {})
```

```
    (auto simp only: lists-length-eq-eq-empty-iff dest: finite-cartesian-productD1)
  from this show ?case by auto
 qed
next
 assume finite A ∨ n = 0
 from this show finite {xs. set xs ⊆ A ∧ length xs = n}
  by (auto intro: finite-lists-length-eq)
qed

lemma card-lists-length-eq:
 shows card {xs. set xs ⊆ B ∧ length xs = n} = card B ^ n
proof cases
 assume finite B
 then show ?thesis by (rule card-lists-length-eq)
next
 assume infinite B
 then show ?thesis
 proof cases
  assume n = 0
  from this have {xs. set xs ⊆ B ∧ length xs = n} = {[]} by auto
  from this ‹n = 0› show ?thesis by simp
 next
  assume n ≠ 0
  from this ‹infinite B› have infinite {xs. set xs ⊆ B ∧ length xs = n}
   by (simp add: finite-lists-length-eq-iff)
  from this ‹infinite B› show ?thesis by auto
 qed
qed
```

1.6 Additions to Disjoint Set Theory

```
lemma bij-betw-congI:
 assumes bij-betw f A A′
 assumes ∀ a ∈ A. f a = g a
 shows bij-betw g A A′
using assms bij-betw-cong by fastforce

lemma disjoint-family-onI[intro]:
 assumes ⋀m n. m ∈ S ⟹ n ∈ S ⟹ m ≠ n ⟹ A m ∩ A n = {}
 shows disjoint-family-on A S
using assms unfolding disjoint-family-on-def by simp
```

The following lemma is not needed for this development, but is useful and could be moved to Disjoint Set theory or Equiv Relation theory if translated from set partitions to equivalence relations.

```
lemma infinite-partition-on:
 assumes infinite A
 shows infinite {P. partition-on A P}
proof −
```

from ‹*infinite A*› **obtain** *x* **where** $x \in A$
by (*meson finite.intros*(*1*) *finite-subset subsetI*)
from ‹*infinite A*› **have** *infinite* $(A - \{x\})$
by (*simp add*: *infinite-remove*)
define *singletons-except-one*
where *singletons-except-one* = ($\lambda a'$. (λa. *if* $a = a'$ *then* $\{a, x\}$ *else* $\{a\}$) ‘ $(A - \{x\})$)
have *infinite* (*singletons-except-one* ‘ $(A - \{x\})$)
proof −
have *inj-on singletons-except-one* $(A - \{x\})$
unfolding *singletons-except-one-def* **by** (*rule inj-onI*) *auto*
from ‹*infinite* $(A - \{x\})$› *this* **show** *?thesis*
using *finite-imageD* **by** *blast*
qed
moreover have *singletons-except-one* ‘ $(A - \{x\}) \subseteq \{P.$ *partition-on A P*$\}$
proof
fix *P*
assume $P \in$ *singletons-except-one* ‘ $(A - \{x\})$
from *this* **obtain** a' **where** $a' \in A - \{x\}$ **and** *P*: *P* = *singletons-except-one* a' **by** *blast*
have *partition-on A* ((λa. *if* $a = a'$ *then* $\{a, x\}$ *else* $\{a\}$) ‘ $(A - \{x\})$)
using ‹$x \in A$› ‹$a' \in A - \{x\}$› **by** (*auto intro*: *partition-onI*)
from *this* **have** *partition-on A P*
unfolding *P singletons-except-one-def* **.**
from *this* **show** $P \in \{P.$ *partition-on A P*$\}$ **..**
qed
ultimately show *?thesis* **by** (*simp add*: *infinite-super*)
qed

lemma *finitely-many-partition-on-iff*:
finite $\{P.$ *partition-on A P*$\}$ $\longleftrightarrow$ *finite A*
using *finitely-many-partition-on infinite-partition-on* **by** *blast*

1.7 Additions to Multiset Theory

lemma *mset-set-subseteq-mset-set*:
assumes *finite B* $A \subseteq B$
shows *mset-set A* ⊆# *mset-set B*
proof −
from ‹$A \subseteq B$› ‹*finite B*› **have** *finite A* **using** *finite-subset* **by** *blast*
{
fix *x*
have *count* (*mset-set A*) $x \leq$ *count* (*mset-set B*) *x*
using ‹*finite A*› ‹*finite B*› ‹$A \subseteq B$›
by (*metis count-mset-set*(*1*, *3*) *eq-iff subsetCE zero-le-one*)
}
from *this* **show** *mset-set A* ⊆# *mset-set B*
using *mset-subset-eqI* **by** *blast*
qed

lemma *mset-set-set-mset*:
  assumes *M* ⊆# *mset-set A*
  shows *mset-set* (*set-mset M*) = *M*
proof −
  {
    fix *x*
    from ‹*M* ⊆# *mset-set A*› **have** *count M x* ≤ *count* (*mset-set A*) *x*
      by (*simp add*: *mset-subset-eq-count*)
    from *this* **have** *count* (*mset-set* (*set-mset M*)) *x* = *count M x*
      by (*metis count-eq-zero-iff count-greater-eq-one-iff count-mset-set*
        dual-order.antisym dual-order.trans finite-set-mset)
  }
  from *this* **show** *?thesis* **by** (*simp add*: *multiset-eq-iff*)
qed

lemma *mset-set-set-mset'*:
  assumes ∀ *x*. *count M x* ≤ *1*
  shows *mset-set* (*set-mset M*) = *M*
proof −
  {
    fix *x*
    from *assms* **have** *count M x* = *0* ∨ *count M x* = *1* **by** (*auto elim*: *le-SucE*)
    from *this* **have** *count* (*mset-set* (*set-mset M*)) *x* = *count M x*
      by (*metis count-eq-zero-iff count-mset-set*(*1*,*3*) *finite-set-mset*)
  }
  from *this* **show** *?thesis* **by** (*simp add*: *multiset-eq-iff*)
qed

lemma *card-set-mset*:
  assumes *M* ⊆# *mset-set A*
  shows *card* (*set-mset M*) = *size M*
using *assms*
by (*metis mset-set-set-mset size-mset-set*)

lemma *card-set-mset'*:
  assumes ∀ *x*. *count M x* ≤ *1*
  shows *card* (*set-mset M*) = *size M*
using *assms*
by (*metis mset-set-set-mset' size-mset-set*)

lemma *count-mset-set-leq*:
  assumes *finite A*
  shows *count* (*mset-set A*) *x* ≤ *1*
using *assms* **by** (*metis count-mset-set*(*1*,*3*) *eq-iff zero-le-one*)

lemma *count-mset-set-leq'*:
  assumes *finite A*
  shows *count* (*mset-set A*) *x* ≤ *Suc 0*

using *assms count-mset-set-leq* **by** *fastforce*

lemma *msubset-mset-set-iff*:
assumes *finite A*
shows *set-mset M* ⊆ *A* ∧ (∀ *x*. *count M x* ≤ *1*) ⟷ (*M* ⊆# *mset-set A*)
proof
assume *set-mset M* ⊆ *A* ∧ (∀ *x*. *count M x* ≤ *1*)
from *this assms* **show** *M* ⊆# *mset-set A*
by (*metis count-inI count-mset-set*(*1*) *le0 mset-subset-eqI subsetCE*)
next
assume *M* ⊆# *mset-set A*
from *this assms* **have** *set-mset M* ⊆ *A*
using *mset-subset-eqD* **by** *fastforce*
moreover {
fix *x*
from ‹*M* ⊆# *mset-set A*› **have** *count M x* ≤ *count* (*mset-set A*) *x*
by (*simp add*: *mset-subset-eq-count*)
from *this* ‹*finite A*› **have** *count M x* ≤ *1*
by (*meson count-mset-set-leq le-trans*)
}
ultimately show *set-mset M* ⊆ *A* ∧ (∀ *x*. *count M x* ≤ *1*) **by** *simp*
qed

lemma *image-mset-fun-upd*:
assumes *x* ∉# *M*
shows *image-mset* (*f*(*x* := *y*)) *M* = *image-mset f M*
using *assms* **by** (*induct M*) *auto*

1.8 Additions to Number Partitions Theory

lemma *Partition-diag*:
shows *Partition n n* = *1*
by (*cases n*) (*auto simp only*: *Partition-diag Partition.simps*(*1*))

1.9 Cardinality Theorems with Iverson Function

definition *iverson* :: *bool* ⇒ *nat*
where
iverson b = (*if b then 1 else 0*)

lemma *card-partition-on-size1-eq-iverson*:
assumes *finite A*
shows *card* {*P*. *partition-on A P* ∧ *card P* ≤ *k* ∧ (∀ *X*∈*P*. *card X* = *1*)} = *iverson* (*card A* ≤ *k*)
proof (*cases card A* ≤ *k*)
case *True*
from *this* ‹*finite A*› **show** *?thesis*
unfolding *iverson-def*
using *card-partition-on-size1-eq-1* **by** *fastforce*
next

case *False*
from *this* ‹*finite A*› **show** *?thesis*
unfolding *iverson-def*
using *card-partition-on-size1-eq-0* **by** *fastforce*
qed

lemma *card-number-partitions-with-only-parts-1*:
card $\{N.\ (\forall n.\ n \in\# N \longrightarrow n = 1) \wedge \mathit{number\text{-}partition}\ n\ N \wedge \mathit{size}\ N \leq x\} = \mathit{iverson}\ (n \leq x)$
proof −
show *?thesis*
proof *cases*
assume $n \leq x$
from *this* **show** *?thesis*
using *card-number-partitions-with-only-parts-1-eq-1*
unfolding *iverson-def* **by** *auto*
next
assume $\neg\ n \leq x$
from *this* **show** *?thesis*
using *card-number-partitions-with-only-parts-1-eq-0*
unfolding *iverson-def* **by** *auto*
qed
qed

end

2 Main Observations on Operations and Permutations

theory *Twelvefold-Way-Core*
imports *Preliminaries*
begin

2.1 Range Multiset

2.1.1 Existence of a Suitable Finite Function

lemma *obtain-function*:
assumes *finite A*
assumes *size M* = *card A*
shows $\exists f.\ \mathit{image\text{-}mset}\ f\ (\mathit{mset\text{-}set}\ A) = M$
using *assms*
proof (*induct arbitrary*: *M rule*: *finite-induct*)
case *empty*
from *this* **show** *?case* **by** *simp*
next
case (*insert x A*)
from *insert*(*1*,*2*,*4*) **have** *size M* > *0*
by (*simp add*: *card-gt-0-iff*)

from *this* **obtain** *y* **where** $y \in\# M$
using *gr0-implies-Suc size-eq-Suc-imp-elem* **by** *blast*
from *insert(1,2,4) this* **have** *size* $(M - \{\#y\#\})$ = *card A*
by (*simp add*: *Diff-insert-absorb card-Diff-singleton-if insertI1 size-Diff-submset*)
from *insert.hyps this* **obtain** *f′* **where** *image-mset f′* (*mset-set A*) = $M - \{\#y\#\}$ **by** *blast*
from *this* **have** *image-mset* $(f'(x := y))$ (*mset-set* (*insert x A*)) = *M*
using ‹*finite A*› ‹$x \notin A$› ‹$y \in\# M$› **by** (*simp add*: *image-mset-fun-upd*)
from *this* **show** *?case* **by** *blast*
qed

lemma *obtain-function-on-ext-funcset*:
assumes *finite A*
assumes *size M* = *card A*
shows $\exists f \in A \rightarrow_E$ *set-mset M*. *image-mset f* (*mset-set A*) = *M*
proof −
obtain *f* **where** *range-eq-M*: *image-mset f* (*mset-set A*) = *M*
using *obtain-function* ‹*finite A*› ‹*size M* = *card A*› **by** *blast*
let *?f* = λx. *if* $x \in A$ *then f x else undefined*
have *?f* $\in A \rightarrow_E$ *set-mset M*
using *range-eq-M* ‹*finite A*› **by** *auto*
moreover have *image-mset ?f* (*mset-set A*) = *M*
using *range-eq-M* ‹*finite A*› **by** (*auto intro*: *multiset.map-cong0*)
ultimately show *?thesis* **by** *auto*
qed

2.1.2 Existence of Permutation

lemma *image-mset-eq-implies-bij-betw*:
fixes *f* :: *′a1* ⇒ *′b* **and** *f′* :: *′a2* ⇒ *′b*
assumes *finite A finite A′*
assumes *mset-eq*: *image-mset f* (*mset-set A*) = *image-mset f′* (*mset-set A′*)
obtains *bij* **where** *bij-betw bij A A′* **and** $\forall x \in A$. *f x* = *f′* (*bij x*)
proof −
from ‹*finite A*› **have** [*simp*]: *finite* $\{a \in A.\ f\ a = (b{::}'b)\}$ **for** *b* **by** *auto*
from ‹*finite A′*› **have** [*simp*]: *finite* $\{a \in A'.\ f'\ a = (b{::}'b)\}$ **for** *b* **by** *auto*
have *f* ‘ *A* = *f′* ‘ *A′*
proof −
have *f* ‘ *A* = *f* ‘ (*set-mset* (*mset-set A*)) **using** ‹*finite A*› **by** *simp*
also have . . . = *f′* ‘ (*set-mset* (*mset-set A′*))
by (*metis mset-eq multiset.set-map*)
also have . . . = *f′* ‘ *A′* **using** ‹*finite A′*› **by** *simp*
finally show *?thesis* **.**
qed
have $\forall b \in (f$ ‘ $A)$. $\exists$ *bij*. *bij-betw bij* $\{a \in A.\ f\ a = b\}$ $\{a \in A'.\ f'\ a = b\}$
proof
fix *b*
from *mset-eq* **have**
count (*image-mset f* (*mset-set A*)) *b* = *count* (*image-mset f′* (*mset-set A′*)) *b*

```
by simp
    from this have card {a ∈ A. f a = b} = card {a ∈ A'. f' a = b}
      using ‹finite A› ‹finite A'›
      by (simp add: count-image-mset-eq-card-vimage)
    from this show ∃ bij. bij-betw bij {a ∈ A. f a = b} {a ∈ A'. f' a = b}
      by (intro finite-same-card-bij) simp-all
  qed
  from bchoice [OF this]
  obtain bij where bij: ∀ b∈f ‘ A. bij-betw (bij b) {a ∈ A. f a = b} {a ∈ A'. f' a
= b}
    by auto
  define bij' where bij' = (λa. bij (f a) a)
  have bij-betw bij' A A'
  proof −
    have disjoint-family-on (λi. {a ∈ A'. f' a = i}) (f ‘ A)
      unfolding disjoint-family-on-def by auto
    moreover have bij-betw (λa. bij (f a) a) {a ∈ A. f a = b} {a ∈ A'. f' a = b}
if b: b ∈ f ‘ A for b
      using bij b by (subst bij-betw-cong[where g=bij b]) auto
    ultimately have bij-betw (λa. bij (f a) a) (⋃ b∈f ‘ A. {a ∈ A. f a = b}) (⋃ b∈f
‘ A. {a ∈ A'. f' a = b})
      by (rule bij-betw-UNION-disjoint)
    moreover have (⋃ b∈f ‘ A. {a ∈ A. f a = b}) = A by auto
    moreover have (⋃ b∈f ‘ A. {a ∈ A'. f' a = b}) = A' using ‹f ‘ A = f' ‘ A'›
by auto
    ultimately show bij-betw bij' A A'
      unfolding bij'-def by (subst bij-betw-cong[where g=(λa. bij (f a) a)]) auto
  qed
  moreover from bij have ∀ x∈A. f x = f' (bij' x)
    unfolding bij'-def using bij-betwE by fastforce
  ultimately show ?thesis by (rule that)
qed

lemma image-mset-eq-implies-permutes:
  fixes f :: 'a ⇒ 'b
  assumes finite A
  assumes mset-eq: image-mset f (mset-set A) = image-mset f' (mset-set A)
  obtains p where p permutes A and ∀ x∈A. f x = f' (p x)
proof −
  from assms obtain b where bij-betw b A A and ∀ x∈A. f x = f' (b x)
    using image-mset-eq-implies-bij-betw by blast
  define p where p = (λa. if a ∈ A then b a else a)
  have p permutes A
  proof (rule bij-imp-permutes)
    show bij-betw p A A
      unfolding p-def by (simp add: ‹bij-betw b A A› bij-betw-cong)
  next
    fix x
    assume x ∉ A
```

from *this* **show** $p\ x = x$
unfolding *p-def* **by** *simp*
qed
moreover from ‹$\forall x{\in}A.\ f\ x = f'\ (b\ x)$› **have** $\forall x{\in}A.\ f\ x = f'\ (p\ x)$
unfolding *p-def* **by** *simp*
ultimately show *?thesis* **by** (*rule that*)
qed

2.2 Domain Partition

2.2.1 Existence of a Suitable Finite Function

lemma *obtain-function-with-partition*:
assumes *finite A finite B*
assumes *partition-on A P*
assumes $card\ P \leq card\ B$
shows $\exists f \in A \rightarrow_E B.\ (\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\} = P$
proof –
obtain g' **where** *bij-betw* $g'\ P\ (g'\ `\ P)$ **and** $g'\ `\ P \subseteq B$
by (*meson assms card-le-inj finite-elements inj-on-imp-bij-betw*)
define f **where** $\bigwedge a.\ f\ a = (\mathit{if}\ a \in A\ \mathit{then}\ g'\ (\mathit{THE}\ X.\ a \in X \wedge X \in P)\ \mathit{else}\ \mathit{undefined})$
have $f \in A \rightarrow_E B$
unfolding *f-def*
using ‹$g'\ `\ P \subseteq B$› *assms(3) partition-on-the-part-mem* **by** *fastforce*
moreover have $(\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\} = P$
proof
show $(\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\} \subseteq P$
proof
fix X
assume $X{:}X \in (\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\}$
from *this* **obtain** b **where** $b \in B$ **and** $X = \{x' \in A.\ f\ x' = b\}$ **by** *auto*
from *this* X **obtain** a **where** $a \in A$ **and** $a \in X$ **and** $f\ a = b$ **by** *blast*
have $(\mathit{THE}\ X.\ a \in X \wedge X \in P) \in P$
using ‹$a \in A$› ‹*partition-on A P*› **by** (*simp add*: *partition-on-the-part-mem*)
from ‹$X = \{x' \in A.\ f\ x' = b\}$› **have** *X-eq1*: $X = \{x' \in A.\ g'\ (\mathit{THE}\ X.\ x' \in X \wedge X \in P) = b\}$
unfolding *f-def* **by** *auto*
also have $\ldots = \{x' \in A.\ (\mathit{THE}\ X.\ x' \in X \wedge X \in P) = \mathit{inv\text{-}into}\ P\ g'\ b\}$
proof –
{
fix x'
assume $x' \in A$
have $(\mathit{THE}\ X.\ x' \in X \wedge X \in P) \in P$
using ‹*partition-on A P*› ‹$x' \in A$› **by** (*simp add*: *partition-on-the-part-mem*)
from *X-eq1* ‹$a \in X$› **have** $g'\ (\mathit{THE}\ X.\ a \in X \wedge X \in P) = b$
unfolding *f-def* **by** *auto*
from *this* ‹$(\mathit{THE}\ X.\ a \in X \wedge X \in P) \in P$› **have** $b \in g'\ `\ P$ **by** *auto*
have $(g'\ (\mathit{THE}\ X.\ x' \in X \wedge X \in P) = b) \longleftrightarrow ((\mathit{THE}\ X.\ x' \in X \wedge X \in P) = \mathit{inv\text{-}into}\ P\ g'\ b)$

proof −
from ‹(*THE* X. $x' \in X \wedge X \in P$) $\in P$›
have (g' (*THE* X. $x' \in X \wedge X \in P$) $= b$) $\longleftrightarrow$ (*inv-into* P g' (g' (*THE* X. $x' \in X \wedge X \in P$)) $=$ *inv-into* P g' b)
using ‹$b \in g'$ ‘ P› **by** (*auto intro*: *inv-into-injective*)
moreover have *inv-into* P g' (g' (*THE* X. $x' \in X \wedge X \in P$)) $=$ (*THE* X. $x' \in X \wedge X \in P$)
using ‹*bij-betw* g' P (g' ‘ P)› ‹(*THE* X. $x' \in X \wedge X \in P$) $\in P$›
by (*simp add*: *bij-betw-inv-into-left*)
ultimately show *?thesis* **by** *simp*
qed
}
from *this* **show** *?thesis* **by** *auto*
qed
finally have *X-eq*: $X = \{x' \in A.$ (*THE* X. $x' \in X \wedge X \in P$) $=$ *inv-into* P g' $b\}$ **.**
moreover have *inv-into* P g' $b \in P$
proof −
from *X-eq* **have** *eq*: *inv-into* P g' $b =$ (*THE* X. $a \in X \wedge X \in P$)
using ‹$a \in X$› ‹$a \in A$› **by** *auto*
from *this* **show** *?thesis*
using ‹(*THE* X. $a \in X \wedge X \in P$) $\in P$› **by** *simp*
qed
ultimately have $X =$ *inv-into* P g' b
using *partition-on-all-in-part-eq-part*[*OF* ‹*partition-on* A P›] **by** *blast*
from *this* ‹*inv-into* P g' $b \in P$› **show** $X \in P$ **by** *blast*
qed
next
show $P \subseteq (\lambda b.\ \{x \in A.\ f\,x = b\})$ ‘ $B - \{\{\}\}$
proof
fix X
assume $X \in P$
from *assms*(*3*) *this* **have** $X \neq \{\}$
by (*auto elim*: *partition-onE*)
moreover have $X \in (\lambda b.\ \{x \in A.\ f\,x = b\})$ ‘ B
proof
show g' $X \in B$
using ‹$X \in P$› ‹g' ‘ $P \subseteq B$› **by** *blast*
show $X = \{x \in A.\ f\,x = g'\ X\}$
proof
show $X \subseteq \{x \in A.\ f\,x = g'\ X\}$
proof
fix x
assume $x \in X$
from *this* **have** $x \in A$
using ‹$X \in P$› *assms*(*3*) **by** (*fastforce elim*: *partition-onE*)
have (*THE* X. $x \in X \wedge X \in P$) $= X$
using ‹$X \in P$› ‹$x \in X$› *assms*(*3*) *partition-on-the-part-eq* **by** *fastforce*
from *this* ‹$x \in A$› **have** $f\,x = g'\ X$

unfolding *f-def* **by** *auto*
from *this* ‹$x \in A$› **show** $x \in \{x \in A.\ f\ x = g'\ X\}$ **by** *auto*
qed
next
show $\{x \in A.\ f\ x = g'\ X\} \subseteq X$
proof
fix x
assume $x \in \{x \in A.\ f\ x = g'\ X\}$
from *this* **have** $x \in A$ **and** *g-eq*: $g'\ (THE\ X.\ x \in X \wedge X \in P) = g'\ X$
unfolding *f-def* **by** *auto*
from ‹$x \in A$› **have** $(THE\ X.\ x \in X \wedge X \in P) \in P$
using *assms(3)* **by** (*simp add*: *partition-on-the-part-mem*)
from *this g-eq* **have** $(THE\ X.\ x \in X \wedge X \in P) = X$
using ‹$X \in P$› ‹*bij-betw* $g'\ P\ (g'$ ‘ $P)$›
by (*metis bij-betw-inv-into-left*)
from *this* ‹$x \in A$› *assms(3)* **show** $x \in X$
using *partition-on-in-the-unique-part* **by** *fastforce*
qed
qed
qed
ultimately show $X \in (\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\}$
by *auto*
qed
qed
ultimately show *?thesis* **by** *blast*
qed

2.2.2 Equality under Permutation Application

lemma *permutes-implies-inv-image-on-eq*:
assumes *p permutes B*
shows $(\lambda b.\ \{x \in A.\ p\ (f\ x) = b\})$ ‘ $B = (\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ B
proof −
have $\forall b \in B.\ \forall x \in A.\ p\ (f\ x) = b \longleftrightarrow f\ x = inv\ p\ b$
using ‹*p permutes B*› **by** (*auto simp add*: *permutes-inverses*)
from *this* **have** $(\lambda b.\ \{x \in A.\ p\ (f\ x) = b\})$ ‘ $B = (\lambda b.\ \{x \in A.\ f\ x = inv\ p\ b\})$ ‘ B
using *image-cong* **by** *blast*
also have $\ldots = (\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ *inv p* ‘ B
by (*auto simp add*: *image-comp*)
also have $\ldots = (\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ B
by (*simp add*: ‹*p permutes B*› *permutes-inv permutes-image*)
finally show *?thesis* **.**
qed

2.2.3 Existence of Permutation

lemma *the-elem*:
assumes $f \in A \rightarrow_E B\ f' \in A \rightarrow_E B$

assumes *partitions-eq*: $(\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\} = (\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\}$
assumes $x \in A$
shows *the-elem* $(f$ ‘ $\{xa \in A.\ f'\ xa = f'\ x\}) = f\ x$
proof −
from ‹$x \in A$› **have** x: $x \in \{x' \in A.\ f'\ x' = f'\ x\}$ **by** *blast*
have $f'\ x \in B$
using ‹$x \in A$› ‹$f' \in A \to_E B$› **by** *blast*
from *this* **have** $\{x' \in A.\ f'\ x' = f'\ x\} \in (\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\}$
using ‹$x \in A$› **by** *blast*
from *this* **have** $\{x' \in A.\ f'\ x' = f'\ x\} \in (\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\}$
using *partitions-eq* **by** *blast*
from *this* **obtain** b **where** *eq*: $\{x' \in A.\ f'\ x' = f'\ x\} = \{x' \in A.\ f\ x' = b\}$ **by** *blast*
also from x *this* **show** *the-elem* $(f$ ‘ $\{x' \in A.\ f'\ x' = f'\ x\}) = f\ x$
by (*metis* (*mono-tags*, *lifting*) *empty-iff mem-Collect-eq the-elem-image-unique*)
qed

lemma *the-elem-eq*:
assumes $f \in A \to_E B$
assumes $b \in f$ ‘ A
shows *the-elem* $(f$ ‘ $\{x' \in A.\ f\ x' = b\}) = b$
proof −
from ‹$b \in f$ ‘ A› **obtain** a **where** $a \in A$ **and** $b = f\ a$ **by** *blast*
from *this* **show** *the-elem* $(f$ ‘ $\{x' \in A.\ f\ x' = b\}) = b$
using *the-elem*[*OF* ‹$f \in A \to_E B$› ‹$f \in A \to_E B$›] **by** *simp*
qed

lemma *partitions-eq-implies*:
assumes $f \in A \to_E B\ f' \in A \to_E B$
assumes *partitions-eq*: $(\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\} = (\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\}$
assumes $x \in A\ x' \in A$
assumes $f\ x = f\ x'$
shows $f'\ x = f'\ x'$
proof −
have $f\ x \in B$ **and** $x \in \{a \in A.\ f\ a = f\ x\}$ **and** $x' \in \{a \in A.\ f\ a = f\ x\}$
using ‹$f \in A \to_E B$› ‹$x \in A$› ‹$x' \in A$› ‹$f\ x = f\ x'$› **by** *auto*
moreover have $\{a \in A.\ f\ a = f\ x\} \in (\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\}$
using ‹$f\ x \in B$› ‹$x \in \{a \in A.\ f\ a = f\ x\}$› **by** *auto*
ultimately obtain b **where** $x \in \{a \in A.\ f'\ a = b\}$ **and** $x' \in \{a \in A.\ f'\ a = b\}$
using *partitions-eq* **by** (*metis* (*no-types*, *lifting*) *Diff-iff imageE*)
from *this* **show** $f'\ x = f'\ x'$ **by** *auto*
qed

lemma *card-domain-partitions*:
assumes $f \in A \to_E B$
assumes *finite* B
shows *card* $((\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\}) =$ *card* $(f$ ‘ $A)$

proof −
  note [*simp*] = *the-elem-eq*[*OF* ‹$f \in A \to_E B$›]
  have *bij-betw* (λX. *the-elem* (f ‘ X)) ((λb. $\{x \in A.\ f\ x = b\}$) ‘ $B - \{\{\}\}$) (f ‘ A)
  proof (*rule bij-betw-imageI*)
    show *inj-on* (λX. *the-elem* (f ‘ X)) ((λb. $\{x \in A.\ f\ x = b\}$) ‘ $B - \{\{\}\}$)
    proof (*rule inj-onI*)
      fix X X'
      assume X: $X \in$ (λb. $\{x \in A.\ f\ x = b\}$) ‘ $B - \{\{\}\}$
      assume X': $X' \in$ (λb. $\{x \in A.\ f\ x = b\}$) ‘ $B - \{\{\}\}$
      assume *eq*: *the-elem* (f ‘ X) = *the-elem* (f ‘ X')
      from X **obtain** b **where** $b \in B$ **and** *X-eq*: $X = \{x \in A.\ f\ x = b\}$ **by** *blast*
      from X *this* **have** $b \in f$ ‘ A
        using *Collect-empty-eq Diff-iff image-iff insertCI* **by** *auto*
      from X' **obtain** b' **where** $b' \in B$ **and** X'*-eq*: $X' = \{x \in A.\ f\ x = b'\}$ **by** *blast*
      from X' *this* **have** $b' \in f$ ‘ A
        using *Collect-empty-eq Diff-iff image-iff insertCI* **by** *auto*
      from *X-eq* X'*-eq* *eq* ‹$\bigwedge b.\ b \in f$ ‘ $A \Longrightarrow$ *the-elem* (f ‘ $\{x' \in A.\ f\ x' = b\}$) $= b$› ‹$b \in f$ ‘ A› ‹$b' \in f$ ‘ A›
        have $b = b'$ **by** *auto*
      from *this* **show** $X = X'$
        using *X-eq* X'*-eq* **by** *simp*
    qed
    show (λX. *the-elem* (f ‘ X)) ‘ ((λb. $\{x \in A.\ f\ x = b\}$) ‘ $B - \{\{\}\}$) = f ‘ A
    proof
      show (λX. *the-elem* (f ‘ X)) ‘ ((λb. $\{x \in A.\ f\ x = b\}$) ‘ $B - \{\{\}\}$) $\subseteq f$ ‘ A
        using ‹$\bigwedge b.\ b \in f$ ‘ $A \Longrightarrow$ *the-elem* (f ‘ $\{x' \in A.\ f\ x' = b\}$) $= b$› **by** *auto*
    next
      show f ‘ $A \subseteq$ (λX. *the-elem* (f ‘ X)) ‘ ((λb. $\{x \in A.\ f\ x = b\}$) ‘ $B - \{\{\}\}$)
      proof
        fix b
        assume $b \in f$ ‘ A
        from *this* **have** $b =$ *the-elem* (f ‘ $\{x \in A.\ f\ x = b\}$)
          using ‹$\bigwedge b.\ b \in f$ ‘ $A \Longrightarrow$ *the-elem* (f ‘ $\{x' \in A.\ f\ x' = b\}$) $= b$› **by** *auto*
        moreover from ‹$b \in f$ ‘ A› **have** $\{x \in A.\ f\ x = b\} \in$ (λb. $\{x \in A.\ f\ x = b\}$) ‘ $B - \{\{\}\}$
          using ‹$f \in A \to_E B$› **by** *auto*
        ultimately show $b \in$ (λX. *the-elem* (f ‘ X)) ‘ ((λb. $\{x \in A.\ f\ x = b\}$) ‘ $B - \{\{\}\}$) **..**
      qed
    qed
  qed
  from *this* **show** *?thesis* **by** (*rule bij-betw-same-card*)
qed

lemma *partitions-eq-implies-permutes*:
  assumes $f \in A \to_E B$ $f' \in A \to_E B$
  assumes *finite* B
  assumes *partitions-eq*: (λb. $\{x \in A.\ f\ x = b\}$) ‘ $B - \{\{\}\}$ = (λb. $\{x \in A.\ f'\ x$

$= b\}$) ‘ $B - \{\{\}\}$
  shows $\exists p.$ *p permutes B* $\wedge$ $(\forall x{\in}A.\ f\ x = p\ (f'\ x))$
proof −
  have *card-eq*: *card* $(f'$ ‘ $A)$ = *card* $(f$ ‘ $A)$
    using *card-domain-partitions*[*OF* ‹$f \in A \rightarrow_E B$› ‹*finite B*›]
    using *card-domain-partitions*[*OF* ‹$f' \in A \rightarrow_E B$› ‹*finite B*›]
    using *partitions-eq* **by** *simp*
  have f' ‘ $A \subseteq B$ f ‘ $A \subseteq B$
    using ‹$f \in A \rightarrow_E B$› ‹$f' \in A \rightarrow_E B$› **by** *auto*
  from *this card-eq* **have** *card* $(B - f'$ ‘ $A)$ = *card* $(B - f$ ‘ $A)$
    using ‹*finite B*› **by** (*auto simp add*: *card-Diff-subset finite-subset*)
  from *this* **obtain** p' **where** *bij-betw* p' $(B - f'$ ‘ $A)$ $(B - f$ ‘ $A)$
    using ‹*finite B*› **by** (*metis finite-same-card-bij finite-Diff*)
  from *this* **have** p' ‘ $(B - f'$ ‘ $A) = (B - f$ ‘ $A)$
    by (*simp add*: *bij-betw-imp-surj-on*)
  define *p* **where** $\bigwedge b.$ $p\ b$ = (*if* $b \in B$ *then*
    (*if* $b \in f'$ ‘ A *then the-elem* $(f$ ‘ $\{x \in A.\ f'\ x = b\})$ *else* $p'\ b$) *else* b)
  have $\forall x{\in}A.\ f\ x = p\ (f'\ x)$
  proof
    fix *x*
    assume $x \in A$
    from *this partitions-eq* **have** *the-elem* $(f$ ‘ $\{xa \in A.\ f'\ xa = f'\ x\}) = f\ x$
      using *the-elem*[*OF* ‹$f \in A \rightarrow_E B$› ‹$f' \in A \rightarrow_E B$›] **by** *auto*
    from *this* **show** $f\ x = p\ (f'\ x)$
      using ‹$x \in A$› *p-def* ‹$f' \in A \rightarrow_E B$› **by** *auto*
  qed
  moreover have *p permutes B*
  proof (*rule bij-imp-permutes*)
    let *?invp* = $\lambda b.$ *if* $b \in f$ ‘ A *then the-elem* $(f'$ ‘ $\{x \in A.\ f\ x = b\})$ *else b*
    note [*simp*] = *the-elem*[*OF* ‹$f \in A \rightarrow_E B$› ‹$f' \in A \rightarrow_E B$› *partitions-eq*]
    show *bij-betw p B B*
    proof (*rule bij-betw-imageI*)
      show p ‘ $B = B$
      proof
        have ($\lambda b.$ *the-elem* $(f$ ‘ $\{x \in A.\ f'\ x = b\})$) ‘ $(f'$ ‘ $A) \subseteq B$
          using ‹$f \in A \rightarrow_E B$› **by** *auto*
        from ‹p' ‘ $(B - f'$ ‘ $A) = (B - f$ ‘ $A)$› *this* **show** p ‘ $B \subseteq B$
          unfolding *p-def* ‹$f \in A \rightarrow_E B$› **by** *force*
      next
        show $B \subseteq p$ ‘ B
        proof
          fix *b*
          assume $b \in B$
          show $b \in p$ ‘ B
          proof (*cases* $b \in f$ ‘ A)
            assume $b \notin f$ ‘ A
            note ‹p' ‘ $(B - f'$ ‘ $A) = (B - f$ ‘ $A)$›
            from *this* ‹$b \in B$› ‹$b \notin f$ ‘ A› **show** *?thesis*
              unfolding *p-def* **by** *auto*

next
assume $b \in f \text{`} A$
from *this* ‹$\forall x{\in}A.\ f\ x = p\ (f'\ x)$› ‹$b \in B$› **show** *?thesis*
using ‹$f' \in A \rightarrow_E B$› **by** *auto*
qed
qed
qed
next
show *inj-on p B*
proof (*rule inj-onI*)
fix $b\ b'$
assume $b \in B\ b' \in B\ p\ b = p\ b'$
have $b \in f' \text{`} A \longleftrightarrow b' \in f' \text{`} A$
proof $-$
have $b \in f' \text{`} A \longleftrightarrow p\ b \in f \text{`} A$
unfolding *p-def* **using** ‹$b \in B$› ‹$p' \text{`} (B - f' \text{`} A) = B - f \text{`} A$› **by** *auto*
also have $p\ b \in f \text{`} A \longleftrightarrow p\ b' \in f \text{`} A$
using ‹$p\ b = p\ b'$› **by** *simp*
also have $p\ b' \in f \text{`} A \longleftrightarrow b' \in f' \text{`} A$
unfolding *p-def* **using** ‹$b' \in B$› ‹$p' \text{`} (B - f' \text{`} A) = B - f \text{`} A$› **by** *auto*
finally show *?thesis* **.**
qed
from *this* **have** $(b \in f' \text{`} A \wedge b' \in f' \text{`} A) \vee (b \notin f' \text{`} A \wedge b' \notin f' \text{`} A)$ **by** *blast*
from *this* **show** $b = b'$
proof
assume $b \in f' \text{`} A \wedge b' \in f' \text{`} A$
from *this* **obtain** $a\ a'$ **where** $a \in A\ \ b = f'\ a$ **and** $a' \in A\ \ b' = f'\ a'$ **by** *auto*
from *this* ‹$b \in B$› ‹$b' \in B$› **have** $p\ b = f\ a\ p\ b' = f\ a'$
unfolding *p-def* **by** *auto*
from *this* ‹$p\ b = p\ b'$› **have** $f\ a = f\ a'$ **by** *simp*
from *this* **have** $f'\ a = f'\ a'$
using *partitions-eq-implies*[*OF* ‹$f \in A \rightarrow_E B$› ‹$f' \in A \rightarrow_E B$› *partitions-eq*]
using ‹$a \in A$› ‹$a' \in A$› **by** *blast*
from *this* **show** $b = b'$
using ‹$b' = f'\ a'$› ‹$b = f'\ a$› **by** *simp*
next
assume $b \notin f' \text{`} A \wedge b' \notin f' \text{`} A$
from *this* ‹$b \in B$› ‹$b' \in B$› **have** $p\ b' = p'\ b'\ p\ b = p'\ b$
unfolding *p-def* **by** *auto*
from *this* ‹$p\ b = p\ b'$› **have** $p'\ b = p'\ b'$ **by** *simp*
moreover have $b \in B - f' \text{`} A\ b' \in B - f' \text{`} A$
using ‹$b \in B$› ‹$b' \in B$› ‹$b \notin f' \text{`} A \wedge b' \notin f' \text{`} A$› **by** *auto*
ultimately show $b = b'$
using ‹*bij-betw* p' - -› **by** (*metis bij-betw-inv-into-left*)
qed
qed
qed

next
  fix *x*
  assume $x \notin B$
  from *this* **show** $p\ x = x$
    using ‹$f' \in A \rightarrow_E B$› *p-def* **by** *auto*
qed
ultimately show *?thesis* **by** *blast*
qed

2.3 Number Partition of Range

2.3.1 Existence of a Suitable Finite Function

lemma *obtain-partition*:
  assumes *finite A*
  assumes *number-partition* (*card A*) *N*
  shows ∃ *P*. *partition-on A P* ∧ *image-mset card* (*mset-set P*) = *N*
using *assms*
proof (*induct N arbitrary*: *A*)
  case *empty*
  from *this* **have** *A* = {}
    unfolding *number-partition-def* **by** *auto*
  from *this* **have** *partition-on A* {} **by** (*simp add*: *partition-on-empty*)
  moreover have *image-mset card* (*mset-set* {}) = {#} **by** *simp*
  ultimately show *?case* **by** *blast*
next
  case (*add x N*)
  from *add.prems*(*2*) **have** *0* ∉# *add-mset x N* **and** *sum-mset* (*add-mset x N*) = *card A*
    unfolding *number-partition-def* **by** *auto*
  from *this* **have** *x* ≤ *card A* **by** *auto*
  from *this* **obtain** *X* **where** *X* ⊆ *A* **and** *card X* = *x*
    using *subset-with-given-card-exists* **by** *auto*
  from *this* **have** *X* ≠ {}
    using ‹*0* ∉# *add-mset x N*› ‹*finite A*› **by** *auto*
  have *sum-mset N* = *card* (*A* − *X*)
    using ‹*sum-mset* (*add-mset x N*) = *card A*› ‹*card X* = *x*› ‹*X* ⊆ *A*›
      by (*metis add.commute add.prems*(*1*) *add-diff-cancel-right′ card-Diff-subset infinite-super sum-mset.add-mset*)
  from *this* ‹*0* ∉# *add-mset x N*› **have** *number-partition* (*card* (*A* − *X*)) *N*
    unfolding *number-partition-def* **by** *auto*
  from *this* **obtain** *P* **where** *partition-on* (*A* − *X*) *P* **and** *eq-N*: *image-mset card* (*mset-set P*) = *N*
    using *add.hyps* ‹*finite A*› **by** *auto*
  from ‹*partition-on* (*A* − *X*) *P*› **have** *finite P*
    using ‹*finite A*› *finite-elements* **by** *blast*
  from ‹*partition-on* (*A* − *X*) *P*› **have** *X* ∉ *P*
    using ‹*X* ≠ {}› *partition-onD1* **by** *fastforce*
  have *partition-on A* (*insert X P*)
    using ‹*partition-on* (*A* − *X*) *P*› ‹*X* ⊆ *A*› ‹*X* ≠ {}›

by (*rule partition-on-insert'*)
moreover have *image-mset card* (*mset-set* (*insert X P*)) = *add-mset x N*
using *eq-N* ‹*card X* = *x*› ‹*finite P*› ‹$X \notin P$› **by** *simp*
ultimately show *?case* **by** *blast*
qed

lemma *obtain-extensional-function-from-number-partition*:
assumes *finite A finite B*
assumes *number-partition* (*card A*) *N*
assumes *size* $N \leq$ *card B*
shows $\exists f \in A \rightarrow_E B$. *image-mset* ($\lambda X$. *card X*) (*mset-set* (((λb. $\{x \in A.\ f\ x = b\}$)) ‘ $B - \{\{\}\}$)) = *N*
proof −
obtain *P* **where** *partition-on A P* **and** *eq-N*: *image-mset card* (*mset-set P*) = *N*
using *assms obtain-partition* **by** *blast*
from *eq-N*[*symmetric*] ‹*size* $N \leq$ *card B*› **have** *card P* $\leq$ *card B* **by** *simp*
from ‹*partition-on A P*› *this* **obtain** *f* **where** $f \in A \rightarrow_E B$
and *eq-P*: (λb. $\{x \in A.\ f\ x = b\}$) ‘ $B - \{\{\}\} = P$
using *obtain-function-with-partition*[*OF* ‹*finite A*› ‹*finite B*›] **by** *blast*
have *image-mset* (λX. *card X*) (*mset-set* (((λb. $\{x \in A.\ f\ x = b\}$)) ‘ $B - \{\{\}\}$)) = *N*
using *eq-P eq-N* **by** *simp*
from *this* ‹$f \in A \rightarrow_E B$› **show** *?thesis* **by** *auto*
qed

2.3.2 Equality under Permutation Application

lemma *permutes-implies-multiset-of-partition-cards-eq*:
assumes p_A *permutes A* p_B *permutes B*
shows *image-mset card* (*mset-set* ((λb. $\{x \in A.\ p_B\ (f'\ (p_A\ x)) = b\}$) ‘ $B - \{\{\}\}$)) = *image-mset card* (*mset-set* ((λb. $\{x \in A.\ f'\ x = b\}$) ‘ $B - \{\{\}\}$))
proof −
have *inj-on* ((‘) (*inv* p_A)) ((λb. $\{x \in A.\ f'\ x = b\}$) ‘ $B - \{\{\}\}$)
by (*meson* ‹p_A *permutes A*› *inj-image-eq-iff inj-onI permutes-surj surj-imp-inj-inv*)
have *image-mset card* (*mset-set* ((λb. $\{x \in A.\ p_B\ (f'\ (p_A\ x)) = b\}$) ‘ $B - \{\{\}\}$)) =
image-mset card (*mset-set* ((λX. *inv* p_A ‘ *X*) ‘ ((λb. $\{x \in A.\ f'\ x = b\}$) ‘ $B - \{\{\}\}$)))
proof −
have (λb. $\{x \in A.\ p_B\ (f'\ (p_A\ x)) = b\}$) ‘ $B - \{\{\}\}$ = (λb. $\{x \in A.\ f'\ (p_A\ x) = b\}$) ‘ $B - \{\{\}\}$
using *permutes-implies-inv-image-on-eq*[*OF* ‹p_B *permutes B*›] **by** *metis*
also have ... = (λb. *inv* p_A ‘ $\{x \in A.\ f'\ x = b\}$) ‘ $B - \{\{\}\}$
proof −
have $\{x \in A.\ f'\ (p_A\ x) = b\}$ = *inv* p_A ‘ $\{x \in A.\ f'\ x = b\}$ **for** *b*
proof
show $\{x \in A.\ f'\ (p_A\ x) = b\} \subseteq$ *inv* p_A ‘ $\{x \in A.\ f'\ x = b\}$
proof

fix x
assume $x \in \{x \in A.\ f'\ (p_A\ x) = b\}$
from *this* **have** $x \in A\ f'\ (p_A\ x) = b$ **by** *auto*
moreover from *this* ‹p_A *permutes* A› **have** $p_A\ x \in A$ **by** (*simp add*: *permutes-in-image*)
moreover from ‹p_A *permutes* A› **have** $x = inv\ p_A\ (p_A\ x)$
using *permutes-inverses*(*2*) **by** *fastforce*
ultimately show $x \in inv\ p_A$ ‘ $\{x \in A.\ f'\ x = b\}$ **by** *auto*
qed
next
show $inv\ p_A$ ‘ $\{x \in A.\ f'\ x = b\} \subseteq \{x \in A.\ f'\ (p_A\ x) = b\}$
proof
fix x
assume $x \in inv\ p_A$ ‘ $\{x \in A.\ f'\ x = b\}$
from *this* **obtain** x' **where** x: $x = inv\ p_A\ x'\ x' \in A\ f'\ x' = b$ **by** *auto*
from *this* ‹p_A *permutes* A› **have** $x \in A$ **by** (*simp add*: *permutes-in-image permutes-inv*)
from ‹$x = inv\ p_A\ x'$› ‹$f'\ x' = b$› **have** $f'\ (p_A\ x) = b$
using ‹p_A *permutes* A› *permutes-inverses*(*1*) **by** *fastforce*
from *this* ‹$x \in A$› **show** $x \in \{x \in A.\ f'\ (p_A\ x) = b\}$ **by** *auto*
qed
qed
from *this* **show** *?thesis* **by** *blast*
qed
also have $\ldots = (\lambda X.\ inv\ p_A$ ‘ $X)$ ‘ $((\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\})$ **by** *auto*
finally show *?thesis* **by** *simp*
qed
also have $\ldots =$ *image-mset* $(\lambda X.\ card\ (inv\ p_A$ ‘ $X))$ (*mset-set* $((\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\}))$
using ‹*inj-on* ((‘) ($inv\ p_A$)) $((\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\})$›
by (*simp only*: *image-mset-mset-set*[*symmetric*] *image-mset.compositionality*) (*meson comp-apply*)
also have $\ldots =$ *image-mset card* (*mset-set* $((\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\}))$
using ‹p_A *permutes* A› **by** (*simp add*: *card-image inj-on-inv-into permutes-surj*)
finally show *?thesis* **.**
qed

2.3.3 Existence of Permutation

lemma *partition-implies-permutes*:
assumes *finite* A
assumes *partition-on* A P *partition-on* A P'
assumes *image-mset card* (*mset-set* P') = *image-mset card* (*mset-set* P)
obtains p **where** p *permutes* A $P' = (\lambda X.\ p$ ‘ $X)$ ‘ P
proof −
from ‹*partition-on* A P› ‹*partition-on* A P'› **have** *finite* P *finite* P'
using ‹*finite* A› *finite-elements* **by** *blast+*
from *this* ‹*image-mset card* (*mset-set* P') = *image-mset card* (*mset-set* P)›

```
  obtain bij where bij-betw bij P P′ and ∀ X∈P. card X = card (bij X)
    using image-mset-eq-implies-bij-betw by metis
  have ∀ X∈P. ∃ p′. bij-betw p′ X (bij X)
  proof
    fix X
    assume X ∈ P
    from this have X ⊆ A
      using ‹partition-on A P› partition-onD1 by fastforce
    from this have finite X
      using ‹finite A› rev-finite-subset by blast
    from ‹X ∈ P› have bij X ∈ P′
      using ‹bij-betw bij P P′› bij-betwE by blast
    from this have bij X ⊆ A
      using ‹partition-on A P′› partition-onD1 by fastforce
    from this have finite (bij X)
      using ‹finite A› rev-finite-subset by blast
    from ‹X ∈ P› have card X = card (bij X)
      using ‹∀ X∈P. card X = card (bij X)› by blast
    from this show ∃ p′. bij-betw p′ X (bij X)
      using ‹finite (bij X)› ‹finite X› finite-same-card-bij by blast
  qed
  from this have ∃ p′. ∀ X∈P. bij-betw (p′ X) X (bij X) by metis
  from this obtain p′ where p′: ∀ X∈P. bij-betw (p′ X) X (bij X) ..
  define p where ⋀a. p a = (if a ∈ A then p′ (THE X. a ∈ X ∧ X ∈ P) a else
a)
  have p permutes A
  proof −
    have bij-betw p A A
    proof −
      have disjoint-family-on bij P
      proof
        fix X X′
        assume XX′: X ∈ P X′ ∈ P X ≠ X′
        from this have bij X ∈ P′ bij X′ ∈ P′
          using ‹bij-betw bij P P′› bij-betwE by blast+
        moreover from XX′ have bij X ≠ bij X′
          using ‹bij-betw bij P P′› by (metis bij-betw-inv-into-left)
        ultimately show bij X ∩ bij X′ = {}
          using ‹partition-on A P′› by (meson partition-onE)
      qed
      moreover have bij-betw (λa. p′ (THE X. a ∈ X ∧ X ∈ P) a) X (bij X) if
X ∈ P for X
      proof −
        from ‹X ∈ P› have bij-betw (p′ X) X (bij X)
          using ‹∀ X∈P. bij-betw (p′ X) X (bij X)› by blast
        moreover from ‹X ∈ P› have ∀ a∈X. (THE X. a ∈ X ∧ X ∈ P) = X
          using ‹partition-on A P› partition-on-the-part-eq by fastforce
        ultimately show ?thesis by (auto intro: bij-betw-congI)
      qed
```

ultimately have *bij-betw* (λa. p' (*THE X*. $a \in X \wedge X \in P$) a) ($\bigcup X{\in}P$. X) ($\bigcup X{\in}P$. *bij* X)
by (*rule bij-betw-UNION-disjoint*)
moreover have ($\bigcup X{\in}P$. X) = A ($\bigcup X{\in}P'$. X) = A
using ‹*partition-on A P*› ‹*partition-on A P*′› *partition-onD1* **by** *auto*
moreover have ($\bigcup X{\in}P$. *bij* X) = ($\bigcup X{\in}P'$. X)
using ‹*bij-betw bij P P*′› *bij-betw-imp-surj-on* **by** *force*
ultimately have *bij-betw* (λa. p' (*THE X*. $a \in X \wedge X \in P$) a) A A **by** *simp*
moreover have $\forall\, a \in A$. p' (*THE X*. $a \in X \wedge X \in P$) $a = p\ a$
unfolding *p-def* **by** *auto*
ultimately show *?thesis* **by** (*rule bij-betw-congI*)
qed
moreover have $p\ x = x$ **if** $x \notin A$ **for** x
using ‹$x \notin A$› *p-def* **by** *auto*
ultimately show *?thesis* **by** (*rule bij-imp-permutes*)
qed
moreover have $P' = (\lambda X.\ p$ ‘ $X)$ ‘ P
proof
show $P' \subseteq (\lambda X.\ p$ ‘ $X)$ ‘ P
proof
fix X
assume $X \in P'$
have *in-P*: *the-inv-into P bij X* $\in P$
using ‹$X \in P'$› ‹*bij-betw bij P P*′› *bij-betwE bij-betw-the-inv-into* **by** *blast*
have *eq-X*: *bij* (*the-inv-into P bij X*) = X
using ‹$X \in P'$› ‹*bij-betw bij P P*′›
by (*meson f-the-inv-into-f-bij-betw*)
have $X = p$ ‘ (*the-inv-into P bij X*)
proof
from *in-P* **have** *the-inv-into P bij X* $\subseteq A$
using ‹*partition-on A P*› *partition-onD1* **by** *fastforce*
have (λa. p' (*THE X*. $a \in X \wedge X \in P$) a) ‘ *the-inv-into P bij X* = X
proof
show (λa. p' (*THE X*. $a \in X \wedge X \in P$) a) ‘ *the-inv-into P bij X* $\subseteq X$
proof
fix x
assume $x \in$ (λa. p' (*THE X*. $a \in X \wedge X \in P$) a) ‘ *the-inv-into P bij X*
from *this* **obtain** a **where** *a-in*: $a \in$ *the-inv-into P bij X*
and *x-eq*: $x = p'$ (*THE X*. $a \in X \wedge X \in P$) a **by** *blast*
have (*THE X*. $a \in X \wedge X \in P$) = *the-inv-into P bij X*
using *a-in in-P* ‹*partition-on A P*› *partition-on-the-part-eq*
by *fastforce*
from *this x-eq* **have** *x-eq*: $x = p'$ (*the-inv-into P bij X*) a
by *auto*
from *this* **have** $x \in$ *bij* (*the-inv-into P bij X*)
using *a-in in-P bij-betwE* p' **by** *blast*
from *this eq-X* **show** $x \in X$ **by** *blast*
qed
next

show $X \subseteq (\lambda a.\ p'\ (\mathit{THE}\ X.\ a \in X \wedge X \in P)\ a)$ ‘ *the-inv-into P bij X*
proof
fix x
assume $x \in X$
let $?X' = $ *the-inv-into P bij X*
define x' **where** $x' = $ *the-inv-into* $?X'$ $(p'\ ?X')$ x
from *in-P p' eq-X* **have** *bij-betw*: *bij-betw* $(p'\ ?X')\ ?X'\ X$ **by** *auto*
from *bij-betw* ‹$x \in X$› **have** $x' \in ?X'$
unfolding *x'-def*
using *bij-betwE bij-betw-the-inv-into* **by** *blast*
from *this in-P* **have** $(\mathit{THE}\ X.\ x' \in X \wedge X \in P) = ?X'$
using ‹*partition-on A P*› *partition-on-the-part-eq* **by** *fastforce*
from *this* ‹$x \in X$› **have** $x = p'\ (\mathit{THE}\ X.\ x' \in X \wedge X \in P)\ x'$
unfolding *x'-def*
using *bij-betw f-the-inv-into-f-bij-betw* **by** *fastforce*
from *this* ‹$x' \in ?X'$› **show** $x \in (\lambda a.\ p'\ (\mathit{THE}\ X.\ a \in X \wedge X \in P)\ a)$ ‘ *the-inv-into P bij X* **..**
qed
qed
from *this* ‹*the-inv-into P bij X* $\subseteq A$› **show** $X \subseteq p$ ‘ *the-inv-into P bij X*
unfolding *p-def* **by** *auto*
next
show p ‘ *the-inv-into P bij X* $\subseteq X$
proof
fix x
assume $x \in p$ ‘ *the-inv-into P bij X*
from *this* **obtain** x' **where** $x = p\ x'$ **and** $x' \in$ *the-inv-into P bij X*
by *auto*
have $x' \in A$
using ‹$x' \in$ *the-inv-into P bij X*› *assms*(2) *in-P partition-onD1* **by** *fastforce*
have *eq*: $(\mathit{THE}\ X.\ x' \in X \wedge X \in P) = $ *the-inv-into P bij X*
using ‹$x' \in$ *the-inv-into P bij X*› *assms*(2) *in-P partition-on-the-part-eq* **by** *fastforce*
have p': p' (*the-inv-into P bij X*) $x' \in X$
using ‹$x' \in$ *the-inv-into P bij X*› *bij-betwE eq-X in-P p'* **by** *blast*
from ‹$x = p\ x'$› ‹$x' \in A$› *eq p'* **show** $x \in X$
unfolding *p-def* **by** *auto*
qed
qed
moreover from ‹$X \in P'$› ‹*bij-betw bij P P'*› **have** *the-inv-into P bij X* $\in P$
using *bij-betwE bij-betw-the-inv-into* **by** *blast*
ultimately show $X \in (\lambda X.\ p$ ‘ $X)$ ‘ P **..**
qed
next
show $(\lambda X.\ p$ ‘ $X)$ ‘ $P \subseteq P'$
proof
fix X'
assume $X' \in (\lambda X.\ p$ ‘ $X)$ ‘ P

from *this* **obtain** *X* **where** *X′-eq*: *X′* = *p* ‘ *X* **and** *X* ∈ *P* **..**
from ‹*X* ∈ *P*› **have** *X* ⊆ *A*
using *assms*(*2*) *partition-onD1* **by** *force*
from ‹*X* ∈ *P*› *p′* **have** *bij*: *bij-betw* (*p′* *X*) *X* (*bij* *X*) **by** *auto*
have *p* ‘ *X* ∈ *P′*
proof −
from ‹*X* ∈ *P*› ‹*bij-betw bij P P′*› **have** *bij X* ∈ *P′*
using *bij-betwE* **by** *blast*
moreover have (λ*a*. *p′* (*THE X*. *a* ∈ *X* ∧ *X* ∈ *P*) *a*) ‘ *X* = *bij X*
proof
show (λ*a*. *p′* (*THE X*. *a* ∈ *X* ∧ *X* ∈ *P*) *a*) ‘ *X* ⊆ *bij X*
proof
fix *x′*
assume *x′* ∈ (λ*a*. *p′* (*THE X*. *a* ∈ *X* ∧ *X* ∈ *P*) *a*) ‘ *X*
from *this* **obtain** *x* **where** *x* ∈ *X* **and** *x′-eq*: *x′* = *p′* (*THE X*. *x* ∈ *X* ∧ *X* ∈ *P*) *x* **..**
from ‹*X* ∈ *P*› ‹*x* ∈ *X*› **have** *eq-X*: (*THE X*. *x* ∈ *X* ∧ *X* ∈ *P*) = *X*
using *assms*(*2*) *partition-on-the-part-eq* **by** *fastforce*
from *bij* ‹*x* ∈ *X*› *x′-eq eq-X* **show** *x′* ∈ *bij X*
using *bij-betwE* **by** *blast*
qed
next
show *bij X* ⊆ (λ*a*. *p′* (*THE X*. *a* ∈ *X* ∧ *X* ∈ *P*) *a*) ‘ *X*
proof
fix *x′*
assume *x′* ∈ *bij X*
let *?x* = *inv-into X* (*p′* *X*) *x′*
from ‹*x′* ∈ *bij X*› *bij* **have** *?x* ∈ *X*
by (*metis bij-betw-imp-surj-on inv-into-into*)
from *this* ‹*X* ∈ *P*› **have** (*THE X*. *?x* ∈ *X* ∧ *X* ∈ *P*) = *X*
using *assms*(*2*) *partition-on-the-part-eq* **by** *fastforce*
from *this* ‹*x′* ∈ *bij X*› *bij* **have** *x′* = *p′* (*THE X*. *?x* ∈ *X* ∧ *X* ∈ *P*) *?x*
using *bij-betw-inv-into-right* **by** *fastforce*
moreover from ‹*x′* ∈ *bij X*› *bij* **have** *?x* ∈ *X*
by (*metis bij-betw-imp-surj-on inv-into-into*)
ultimately show *x′* ∈ (λ*a*. *p′* (*THE X*. *a* ∈ *X* ∧ *X* ∈ *P*) *a*) ‘ *X* **..**
qed
qed
ultimately have (λ*a*. *p′* (*THE X*. *a* ∈ *X* ∧ *X* ∈ *P*) *a*) ‘ *X* ∈ *P′* **by** *simp*
have (λ*a*. *p′* (*THE X*. *a* ∈ *X* ∧ *X* ∈ *P*) *a*) ‘ *X* = (λ*a*. *if a* ∈ *A then p′* (*THE X*. *a* ∈ *X* ∧ *X* ∈ *P*) *a else a*) ‘ *X*
using ‹*X* ⊆ *A*› **by** (*auto intro*: *image-cong*)
from *this* **show** *?thesis*
using ‹(λ*a*. *p′* (*THE X*. *a* ∈ *X* ∧ *X* ∈ *P*) *a*) ‘ *X* ∈ *P′*› **unfolding** *p-def* **by** *auto*
qed
from *this* *X′-eq* **show** *X′* ∈ *P′* **by** *simp*
qed
qed

ultimately show *thesis* **using** *that* **by** *blast*
qed

lemma *permutes-domain-partition-eq*:
  assumes $f \in A \to B$
  assumes p_A *permutes* A
  assumes $b \in B$
  shows p_A ` $\{x \in A.\ f\ x = b\} = \{x \in A.\ f\ (inv\ p_A\ x) = b\}$
proof
  show p_A ` $\{x \in A.\ f\ x = b\} \subseteq \{x \in A.\ f\ (inv\ p_A\ x) = b\}$
    using ‹p_A *permutes* A› *permutes-in-image permutes-inverses*(*2*) **by** *fastforce*
next
  show $\{x \in A.\ f\ (inv\ p_A\ x) = b\} \subseteq p_A$ ` $\{x \in A.\ f\ x = b\}$
  proof
    fix x
    assume $x \in \{x \in A.\ f\ (inv\ p_A\ x) = b\}$
    from *this* **have** $x \in A$ $f\ (inv\ p_A\ x) = b$ **by** *auto*
    from ‹$x \in A$› **have** $x = p_A\ (inv\ p_A\ x)$
      using ‹p_A *permutes* A› *permutes-inverses*(*1*) **by** *fastforce*
    moreover from ‹$f\ (inv\ p_A\ x) = b$› ‹$x \in A$› **have** $inv\ p_A\ x \in \{x \in A.\ f\ x = b\}$
      by (*simp add*: ‹p_A *permutes* A› *permutes-in-image permutes-inv*)
    ultimately show $x \in p_A$ ` $\{x \in A.\ f\ x = b\}$ **..**
  qed
qed

lemma *image-domain-partition-eq*:
  assumes $f \in A \to_E B$
  assumes p_A *permutes* A
  shows $(\lambda X.\ p_A$ ` $X)$ ` $((\lambda b.\ \{x \in A.\ f\ x = b\})$ ` $B) = (\lambda b.\ \{x \in A.\ f\ (inv\ p_A\ x) = b\})$ ` B
proof
  from ‹$f \in A \to_E B$› **have** $f \in A \to B$ **by** *auto*
  note *eq* = *permutes-domain-partition-eq*[*OF* ‹$f \in A \to B$› ‹p_A *permutes* A›]
  show $(\lambda X.\ p_A$ ` $X)$ ` $(\lambda b.\ \{x \in A.\ f\ x = b\})$ ` $B \subseteq (\lambda b.\ \{x \in A.\ f\ (inv\ p_A\ x) = b\})$ ` B
  proof
    fix X
    assume $X \in (\lambda X.\ p_A$ ` $X)$ ` $(\lambda b.\ \{x \in A.\ f\ x = b\})$ ` B
    from *this* **obtain** b **where** $b \in B$ **and** *X-eq*: $X = p_A$ ` $\{x \in A.\ f\ x = b\}$ **by** *auto*
    from *this eq* **have** $X = \{x \in A.\ f\ (inv\ p_A\ x) = b\}$ **by** *simp*
    from *this* ‹$b \in B$› **show** $X \in (\lambda b.\ \{x \in A.\ f\ (inv\ p_A\ x) = b\})$ ` B **..**
  qed
next
  from ‹$f \in A \to_E B$› **have** $f \in A \to B$ **by** *auto*
  note *eq* = *permutes-domain-partition-eq*[*OF* ‹$f \in A \to B$› ‹p_A *permutes* A›, *symmetric*]
  show $(\lambda b.\ \{x \in A.\ f\ (inv\ p_A\ x) = b\})$ ` $B \subseteq (\lambda X.\ p_A$ ` $X)$ ` $(\lambda b.\ \{x \in A.\ f\ x = b\})$ ` B

proof
fix X
assume $X \in (\lambda b.\ \{x \in A.\ f\ (inv\ p_A\ x) = b\})$ ` B
from *this* **obtain** b **where** $b \in B$ **and** *X-eq*: $X = \{x \in A.\ f\ (inv\ p_A\ x) = b\}$ **by** *auto*
from *this eq* **have** $X = p_A$ ` $\{x \in A.\ f\ x = b\}$ **by** *simp*
from *this* ‹$b \in B$› **show** $X \in (\lambda X.\ p_A$ ` $X)$ ` $(\lambda b.\ \{x \in A.\ f\ x = b\})$ ` B **by** *auto*
qed
qed

lemma *multiset-of-partition-cards-eq-implies-permutes*:
assumes *finite* A *finite* B $f \in A \to_E B$ $f' \in A \to_E B$
assumes *eq*: *image-mset card* (*mset-set* $((\lambda b.\ \{x \in A.\ f\ x = b\})$ ` $B - \{\{\}\}))$ = *image-mset card* (*mset-set* $((\lambda b.\ \{x \in A.\ f'\ x = b\})$ ` $B - \{\{\}\}))$
obtains p_A p_B **where** p_A *permutes* A p_B *permutes* B $\forall x{\in}A.\ f\ x = p_B\ (f'\ (p_A\ x))$
proof −
have *partition-on* A $((\lambda b.\ \{x \in A.\ f\ x = b\})$ ` $B - \{\{\}\})$
using ‹$f \in A \to_E B$› **by** (*auto intro*!: *partition-onI*)
moreover have *partition-on* A $((\lambda b.\ \{x \in A.\ f'\ x = b\})$ ` $B - \{\{\}\})$
using ‹$f' \in A \to_E B$› **by** (*auto intro*!: *partition-onI*)
moreover note *partition-implies-permutes*[*OF* ‹*finite* A› - - *eq*]
ultimately obtain p_A **where** p_A *permutes* A **and**
inv-image-eq: $(\lambda b.\ \{x \in A.\ f\ x = b\})$ ` $B - \{\{\}\}$ =
(`) $p_A$ ` $((\lambda b.\ \{x \in A.\ f'\ x = b\})$ ` $B - \{\{\}\})$ **by** *blast*
from ‹p_A *permutes* A› **have** *inj* ((`) p_A)
by (*meson injI inj-image-eq-iff permutes-inj*)
have *inv-image-eq'*: $(\lambda b.\ \{x \in A.\ f\ x = b\})$ ` $B - \{\{\}\} = (\lambda b.\ \{x \in A.\ f'\ (inv\ p_A\ x) = b\})$ ` $B - \{\{\}\}$
proof −
note *inv-image-eq*
also have $(\lambda X.\ p_A$ ` $X)$ ` $((\lambda b.\ \{x \in A.\ f'\ x = b\})$ ` $B - \{\{\}\}) = (\lambda b.\ \{x \in A.\ f'\ (inv\ p_A\ x) = b\})$ ` $B - \{\{\}\}$
using *image-domain-partition-eq*[*OF* ‹$f' \in A \to_E B$› ‹p_A *permutes* A›]
by (*simp add*: *image-set-diff*[*OF* ‹*inj* ((`) p_A)›])
finally show *?thesis* **.**
qed
from ‹p_A *permutes* A› **have** *inv* p_A *permutes* A
using *permutes-inv* **by** *blast*
have $(\lambda x.\ f'\ (inv\ p_A\ x)) \in A \to_E B$
using ‹$f' \in A \to_E B$› ‹*inv* p_A *permutes* A› *permutes-in-image* **by** *fastforce*
from ‹$f \in A \to_E B$› *this* ‹*finite* B› **obtain** p_B
where p_B *permutes* B **and** *eq''*: $\forall x{\in}A.\ f\ x = p_B\ (f'\ (inv\ p_A\ x))$
using *partitions-eq-implies-permutes*[*OF* - - - *inv-image-eq'*] **by** *blast*
from ‹*inv* p_A *permutes* A› ‹p_B *permutes* B› *eq''* *that* **show** *thesis* **by** *blast*
qed

2.4 Bijections on Same Domain and Range

2.4.1 Existence of Domain Permutation

lemma *obtain-domain-permutation-for-two-bijections*:
  assumes *bij-betw f A B bij-betw f′ A B*
  obtains *p* **where** *p permutes A* **and** $\forall a \in A.\ f\ a = f'\ (p\ a)$
proof −
  let *?p* = *λa. if a ∈ A then the-inv-into A f′ (f a) else a*
  have *?p permutes A*
  proof (*rule bij-imp-permutes*)
    show *bij-betw ?p A A*
    proof (*rule bij-betw-imageI*)
      show *inj-on ?p A*
      proof (*rule inj-onI*)
        fix *a a′*
        assume *a ∈ A a′ ∈ A ?p a = ?p a′*
        from *this* **have** *the-inv-into A f′ (f a) = the-inv-into A f′ (f a′)*
          using ‹*a ∈ A*› ‹*a′ ∈ A*› **by** *simp*
        from *this* **have** *f a = f a′*
          using ‹*a ∈ A*› ‹*a′ ∈ A*› *assms*
          by (*metis bij-betwE f-the-inv-into-f-bij-betw*)
        from *this* **show** *a = a′*
          using ‹*a ∈ A*› ‹*a′ ∈ A*› *assms*
          by (*metis bij-betw-inv-into-left*)
      qed
    next
      show *?p ‘ A = A*
      proof
        show *?p ‘ A ⊆ A*
        proof
          fix *a*
          assume *a ∈ ?p ‘ A*
          from *this* **obtain** *a′* **where** *a′ ∈ A* **and** *a = the-inv-into A f′ (f a′)* **by** *auto*
          from *this assms* **show** *a ∈ A*
            by (*metis bij-betwE bij-betw-imp-inj-on bij-betw-imp-surj-on subset-iff the-inv-into-into*)
        qed
      next
        show *A ⊆ ?p ‘ A*
        proof
          fix *a*
          assume *a ∈ A*
          from *this assms* **have** *the-inv-into A f (f′ a) ∈ A*
            by (*meson bij-betwE bij-betw-the-inv-into*)
          moreover from ‹*a ∈ A*› *assms* **have** *a = the-inv-into A f′ (f (the-inv-into A f (f′ a)))*
          by (*metis bij-betwE bij-betw-imp-inj-on f-the-inv-into-f-bij-betw the-inv-into-f-eq*)
          ultimately show *a ∈ ?p ‘ A* **by** *auto*

```
      qed
    qed
  qed
next
  fix a
  assume a ∉ A
  from this show ?p a = a by auto
qed
moreover have ∀ a∈A. f a = f' (?p a)
  using ‹bij-betw f A B› ‹bij-betw f' A B›
  using bij-betwE f-the-inv-into-f-bij-betw by fastforce
moreover note that
ultimately show thesis by auto
qed
```

2.4.2 Existence of Range Permutation

```
lemma obtain-range-permutation-for-two-bijections:
  assumes bij-betw f A B bij-betw f' A B
  obtains p where p permutes B and ∀ a∈A. f a = p (f' a)
proof −
  let ?p = λb. if b ∈ B then f (inv-into A f' b) else b
  have ?p permutes B
  proof (rule bij-imp-permutes)
    show bij-betw ?p B B
    proof (rule bij-betw-imageI)
      show inj-on ?p B
      proof (rule inj-onI)
        fix b b'
        assume b ∈ B b' ∈ B ?p b = ?p b'
        from this have f (inv-into A f' b) = f (inv-into A f' b')
          using ‹b ∈ B› ‹b' ∈ B› by simp
        from this have inv-into A f' b = inv-into A f' b'
          using ‹b ∈ B› ‹b' ∈ B› assms
          by (metis bij-betw-imp-surj-on bij-betw-inv-into-left inv-into-into)
        from this show b = b'
          using ‹b ∈ B› ‹b' ∈ B› assms(2)
          by (metis bij-betw-inv-into-right)
      qed
    next
      show ?p ‘ B = B
      proof
        from assms show ?p ‘ B ⊆ B
          by (auto simp add: bij-betwE bij-betw-def inv-into-into)
      next
        show B ⊆ ?p ‘ B
        proof
          fix b
          assume b ∈ B
```

from *this assms* **have** f' (*inv-into A f b*) ∈ *B*
by (*metis bij-betwE bij-betw-imp-surj-on inv-into-into*)
moreover have *b* = *?p* (f' (*inv-into A f b*))
using *assms* ‹f' (*inv-into A f b*) ∈ *B*› ‹*b* ∈ *B*›
by (*auto simp add*: *bij-betw-imp-surj-on bij-betw-inv-into-left bij-betw-inv-into-right inv-into-into*)
ultimately show *b* ∈ *?p* ' *B* **by** *auto*
qed
qed
qed
next
fix *b*
assume *b* ∉ *B*
from *this* **show** *?p b* = *b* **by** *auto*
qed
moreover have ∀ *a*∈*A*. *f a* = *?p* (f' *a*)
using ‹*bij-betw* f' *A B*› *bij-betw-inv-into-left bij-betwE* **by** *fastforce*
moreover note *that*
ultimately show *thesis* **by** *auto*
qed

end

3 Definition of Equivalence Classes

theory *Equiv-Relations-on-Functions*
imports
Preliminaries
Twelvefold-Way-Core
begin

3.1 Permutation on the Domain

definition *domain-permutation*
where
domain-permutation A B = {(*f*, f') ∈ ($A \rightarrow_E B$) × ($A \rightarrow_E B$). ∃ *p*. *p permutes A* ∧ (∀ *x* ∈ *A*. *f x* = f' (*p x*))}

lemma *equiv-domain-permutation*:
equiv ($A \rightarrow_E B$) (*domain-permutation A B*)
proof (*rule equivI*)
show *domain-permutation A B* ⊆ ($A \rightarrow_E B$) × ($A \rightarrow_E B$)
unfolding *domain-permutation-def* **by** *auto*
next
show *refl-on* ($A \rightarrow_E B$) (*domain-permutation A B*)
proof (*rule refl-onI*)
fix *f*
assume *f* ∈ $A \rightarrow_E B$
from *this* **show** (*f*, *f*) ∈ *domain-permutation A B*

using *permutes-id* **unfolding** *domain-permutation-def* **by** *fastforce*
qed
next
show *sym* (*domain-permutation A B*)
proof (*rule symI*)
fix $f\, f'$
assume $(f, f') \in$ *domain-permutation A B*
from *this* **obtain** p **where** p *permutes* A **and** $\forall x{\in}A.\ f\ x = f'\ (p\ x)$
unfolding *domain-permutation-def* **by** *auto*
from ‹$(f, f') \in$ *domain-permutation A B*› **have** $f \in A \to_E B\ f' \in A \to_E B$
unfolding *domain-permutation-def* **by** *auto*
moreover from ‹p *permutes* A› **have** *inv p permutes A*
by (*simp add*: *permutes-inv*)
moreover from ‹p *permutes* A› ‹$\forall x{\in}A.\ f\ x = f'\ (p\ x)$› **have** $\forall x{\in}A.\ f'\ x = f$
(*inv p x*)
using *permutes-in-image permutes-inverses*(*1*) **by** (*metis* (*mono-tags*, *opaque-lifting*))
ultimately show $(f', f) \in$ *domain-permutation A B*
unfolding *domain-permutation-def* **by** *auto*
qed
next
show *trans* (*domain-permutation A B*)
proof (*rule transI*)
fix $f\, f'\, f''$
assume $(f, f') \in$ *domain-permutation A B* $(f', f'') \in$ *domain-permutation A B*
from ‹$(f, f') \in$ -› **obtain** p **where** p *permutes* A **and** $\forall x{\in}A.\ f\ x = f'\ (p\ x)$
unfolding *domain-permutation-def* **by** *auto*
from ‹$(f', f'') \in$ -› **obtain** p' **where** p' *permutes* A **and** $\forall x{\in}A.\ f'\ x = f''\ (p'$
$x)$
unfolding *domain-permutation-def* **by** *auto*
from ‹$(f, f') \in$ *domain-permutation A B*› **have** $f \in A \to_E B$
unfolding *domain-permutation-def* **by** *auto*
moreover from ‹$(f', f'') \in$ *domain-permutation A B*› **have** $f'' \in A \to_E B$
unfolding *domain-permutation-def* **by** *auto*
moreover from ‹p *permutes* A› ‹p' *permutes* A› **have** $(p' \circ p)$ *permutes* A
by (*simp add*: *permutes-compose*)
moreover have $\forall x{\in}A.\ f\ x = f''\ ((p' \circ p)\ x)$
using ‹$\forall x{\in}A.\ f\ x = f'\ (p\ x)$› ‹$\forall x{\in}A.\ f'\ x = f''\ (p'\ x)$› ‹$p$ *permutes* A›
by (*simp add*: *permutes-in-image*)
ultimately show $(f, f'') \in$ *domain-permutation A B*
unfolding *domain-permutation-def* **by** *auto*
qed
qed

3.1.1 Respecting Functions

lemma *inj-on-respects-domain-permutation*:
(λf. *inj-on f A*) *respects domain-permutation A B*
proof (*rule congruentI*)
fix $f\, f'$

```
  assume (f, f′) ∈ domain-permutation A B
  from this obtain p where p: p permutes A ∀x∈A. f x = f′ (p x)
    unfolding domain-permutation-def by auto
  have inv-p: ∀x∈A. f′ x = f (inv p x)
    using p by (metis permutes-inverses(1) permutes-not-in)
  show inj-on f A ⟷ inj-on f′ A
  proof
    assume inj-on f A
    show inj-on f′ A
    proof (rule inj-onI)
      fix a a′
      assume a ∈ A a′ ∈ A f′ a = f′ a′
      from this ‹p permutes A› have inv p a ∈ A inv p a′ ∈ A
        by (simp add: permutes-in-image permutes-inv)+
      have f (inv p a) = f (inv p a′)
        using ‹f′ a = f′ a′› ‹a ∈ A› ‹a′ ∈ A› inv-p by auto
      from ‹inj-on f A› this ‹inv p a ∈ A› ‹inv p a′ ∈ A› have inv p a = inv p a′
        using inj-on-contraD by fastforce
      from this show a = a′
        by (metis ‹p permutes A› permutes-inverses(1))
    qed
  next
    assume inj-on f′ A
    from this p show inj-on f A
      unfolding inj-on-def
      by (metis inj-on-contraD permutes-in-image permutes-inj-on)
  qed
qed

lemma image-respects-domain-permutation:
  (λf. f ‘ A) respects (domain-permutation A B)
proof (rule congruentI)
  fix f f′
  assume (f, f′) ∈ domain-permutation A B
  from this obtain p where p: p permutes A and f-eq: ∀x∈A. f x = f′ (p x)
    unfolding domain-permutation-def by auto
  show f ‘ A = f′ ‘ A
  proof
    from p f-eq show f ‘ A ⊆ f′ ‘ A
      by (auto simp add: permutes-in-image)
  next
    from ‹p permutes A› ‹∀x∈A. f x = f′ (p x)› have ∀x∈A. f′ x = f (inv p x)
    using permutes-in-image permutes-inverses(1) by (metis (mono-tags, opaque-lifting))
    from this show f′ ‘ A ⊆ f ‘ A
      using ‹p permutes A› by (auto simp add: permutes-inv permutes-in-image)
  qed
qed

lemma surjective-respects-domain-permutation:
```

($\lambda f.\ f$ ‘ $A = B$) *respects domain-permutation* A B
by (*metis image-respects-domain-permutation congruentD congruentI*)

lemma *bij-betw-respects-domain-permutation*:
($\lambda f.$ *bij-betw* f A B) *respects domain-permutation* A B
proof (*rule congruentI*)
fix f f'
assume $(f, f') \in$ *domain-permutation* A B
from *this* **obtain** p **where** p *permutes* A **and** $\forall x{\in}A.\ f\ x = f'\ (p\ x)$
unfolding *domain-permutation-def* **by** *auto*
have *bij-betw* f A $B \longleftrightarrow$ *bij-betw* $(f'\ o\ p)$ A B
using ‹$\forall x{\in}A.\ f\ x = f'\ (p\ x)$›
by (*metis* (*mono-tags*, *opaque-lifting*) *comp-apply*[*of* f' p] *bij-betw-cong*[*of* A f $f' \circ p$ B])
also have ... $\longleftrightarrow$ *bij-betw* f' A B
using ‹p *permutes* A›
by (*auto intro*!: *bij-betw-comp-iff*[*symmetric*] *permutes-imp-bij*)
finally show *bij-betw* f A $B \longleftrightarrow$ *bij-betw* f' A B **.**
qed

lemma *image-mset-respects-domain-permutation*:
shows ($\lambda f.$ *image-mset* f (*mset-set* A)) *respects* (*domain-permutation* A B)
proof (*rule congruentI*)
fix f f'
assume $(f, f') \in$ *domain-permutation* A B
from *this* **obtain** p **where** p *permutes* A **and** $\forall x{\in}A.\ f\ x = f'\ (p\ x)$
unfolding *domain-permutation-def* **by** *auto*
from *this* **show** *image-mset* f (*mset-set* A) = *image-mset* f' (*mset-set* A)
using *permutes-implies-image-mset-eq* **by** *fastforce*
qed

3.2 Permutation on the Range

definition *range-permutation*
where
range-permutation A B = $\{(f, f') \in (A \rightarrow_E B) \times (A \rightarrow_E B).\ \exists p.\ p$ *permutes* $B \wedge (\forall x \in A.\ f\ x = p\ (f'\ x))\}$

lemma *equiv-range-permutation*:
equiv $(A \rightarrow_E B)$ (*range-permutation* A B)
proof (*rule equivI*)
show *range-permutation* A $B \subseteq (A \rightarrow_E B) \times (A \rightarrow_E B)$
unfolding *range-permutation-def* **by** *auto*
next
show *refl-on* $(A \rightarrow_E B)$ (*range-permutation* A B)
proof (*rule refl-onI*)
fix f
assume $f \in A \rightarrow_E B$
from *this* **show** $(f, f) \in$ *range-permutation* A B

using *permutes-id* **unfolding** *range-permutation-def* **by** *fastforce*
qed
next
show *sym* (*range-permutation A B*)
proof (*rule symI*)
fix *f f'*
assume $(f, f') \in$ *range-permutation A B*
from *this* **obtain** *p* **where** *p permutes B* **and** $\forall x \in A.\ f\ x = p\ (f'\ x)$
unfolding *range-permutation-def* **by** *auto*
from ‹$(f, f') \in$ *range-permutation A B*› **have** $f \in A \to_E B$ $f' \in A \to_E B$
unfolding *range-permutation-def* **by** *auto*
moreover from ‹*p permutes B*› **have** *inv p permutes B*
by (*simp add*: *permutes-inv*)
moreover from ‹*p permutes B*› ‹$\forall x \in A.\ f\ x = p\ (f'\ x)$› **have** $\forall x \in A.\ f'\ x =$ *inv p* $(f\ x)$
by (*simp add*: *permutes-inverses*(*2*))
ultimately show $(f', f) \in$ *range-permutation A B*
unfolding *range-permutation-def* **by** *auto*
qed
next
show *trans* (*range-permutation A B*)
proof (*rule transI*)
fix *f f' f''*
assume $(f, f') \in$ *range-permutation A B* $(f', f'') \in$ *range-permutation A B*
from ‹$(f, f') \in$ -› **obtain** *p* **where** *p permutes B* **and** $\forall x \in A.\ f\ x = p\ (f'\ x)$
unfolding *range-permutation-def* **by** *auto*
from ‹$(f', f'') \in$ -› **obtain** *p'* **where** *p' permutes B* **and** $\forall x \in A.\ f'\ x = p'\ (f''\ x)$
unfolding *range-permutation-def* **by** *auto*
from ‹$(f, f') \in$ *range-permutation A B*› **have** $f \in A \to_E B$
unfolding *range-permutation-def* **by** *auto*
moreover from ‹$(f', f'') \in$ *range-permutation A B*› **have** $f'' \in A \to_E B$
unfolding *range-permutation-def* **by** *auto*
moreover from ‹*p permutes B*› ‹*p' permutes B*› **have** $(p \circ p')$ *permutes B*
by (*simp add*: *permutes-compose*)
moreover have $\forall x \in A.\ f\ x = (p \circ p')\ (f''\ x)$
using ‹$\forall x \in A.\ f\ x = p\ (f'\ x)$› ‹$\forall x \in A.\ f'\ x = p'\ (f''\ x)$› **by** *auto*
ultimately show $(f, f'') \in$ *range-permutation A B*
unfolding *range-permutation-def* **by** *auto*
qed
qed

3.2.1 Respecting Functions

lemma *inj-on-respects-range-permutation*:
$(\lambda f.$ *inj-on f A*) *respects range-permutation A B*
proof (*rule congruentI*)
fix *f f'*
assume $(f, f') \in$ *range-permutation A B*

from *this* **obtain** p **where** p: p *permutes* B $\forall x \in A.\ f\ x = p\ (f'\ x)$
unfolding *range-permutation-def* **by** *auto*
have *inv-p*: $\forall x \in A.\ f'\ x = inv\ p\ (f\ x)$
using p **by** (*simp add*: *permutes-inverses*(*2*))
show *inj-on* f $A \longleftrightarrow$ *inj-on* f' A
proof
assume *inj-on* f A
from *this* p **show** *inj-on* f' A
unfolding *inj-on-def* **by** *auto*
next
assume *inj-on* f' A
from *this inv-p* **show** *inj-on* f A
unfolding *inj-on-def* **by** *auto*
qed
qed

lemma *surj-on-respects-range-permutation*:
$(\lambda f.\ f\ `\ A = B)$ *respects range-permutation* A B
proof (*rule congruentI*)
fix $f\ f'$
assume a: $(f, f') \in$ *range-permutation* A B
from *this* **have** $f \in A \rightarrow_E B$ $f' \in A \rightarrow_E B$
unfolding *range-permutation-def* **by** *auto*
from a **obtain** p **where** p: p *permutes* B $\forall x \in A.\ f\ x = p\ (f'\ x)$
unfolding *range-permutation-def* **by** *auto*
have *1*: $f\ `\ A = (\lambda x.\ p\ (f'\ x))\ `\ A$
using p **by** (*meson image-cong*)
have *2*: $inv\ p\ `\ ((\lambda x.\ p\ (f'\ x))\ `\ A) = f'\ `\ A$
using p **by** (*simp add*: *image-image image-inv-f-f permutes-inj*)
show $(f\ `\ A = B) = (f'\ `\ A = B)$
proof
assume $f\ `\ A = B$
from *this 1 2* **show** $f'\ `\ A = B$
using p **by** (*simp add*: *permutes-image permutes-inv*)
next
assume $f'\ `\ A = B$
from *this 1 2* **show** $f\ `\ A = B$
using p **by** (*metis image-image permutes-image*)
qed
qed

lemma *bij-betw-respects-range-permutation*:
$(\lambda f.$ *bij-betw* f A $B)$ *respects range-permutation* A B
proof (*rule congruentI*)
fix $f\ f'$
assume $(f, f') \in$ *range-permutation* A B
from *this* **obtain** p **where** p *permutes* B **and** $\forall x \in A.\ f\ x = p\ (f'\ x)$
and $f' \in A \rightarrow_E B$
unfolding *range-permutation-def* **by** *auto*

have *bij-betw* $f\ A\ B \longleftrightarrow$ *bij-betw* $(p\ o\ f')\ A\ B$
using $\langle\forall x{\in}A.\ f\ x = p\ (f'\ x)\rangle$
by (*metis* (*mono-tags*, *opaque-lifting*) *bij-betw-cong comp-apply*)
also have ... $\longleftrightarrow$ *bij-betw* $f'\ A\ B$
using $\langle f' \in A \rightarrow_E B\rangle$ $\langle p$ *permutes* $B\rangle$
by (*auto intro*!: *bij-betw-comp-iff2*[*symmetric*] *permutes-imp-bij*)
finally show *bij-betw* $f\ A\ B \longleftrightarrow$ *bij-betw* $f'\ A\ B$ **.**
qed

lemma *domain-partitions-respects-range-permutation*:
$(\lambda f.\ (\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\})$ *respects range-permutation* $A\ B$
proof (*rule congruentI*)
fix $f\ f'$
assume $(f, f') \in$ *range-permutation* $A\ B$
from *this* **obtain** p **where** p: p *permutes* $B\ \forall x \in A.\ f\ x = p\ (f'\ x)$
unfolding *range-permutation-def* **by** *blast*
have $\{\} \in (\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B \longleftrightarrow \neg\ (\forall b \in B.\ \exists x \in A.\ f'\ x = b)$ **by** *auto*
also have $(\forall b \in B.\ \exists x \in A.\ f'\ x = b) \longleftrightarrow (\forall b \in B.\ \exists x \in A.\ p\ (f'\ x) = b)$
proof
assume $\forall b{\in}B.\ \exists x{\in}A.\ f'\ x = b$
from *this* **show** $\forall b{\in}B.\ \exists x{\in}A.\ p\ (f'\ x) = b$
using $\langle p$ *permutes* $B\rangle$ **unfolding** *permutes-def* **by** *metis*
next
assume $\forall b{\in}B.\ \exists x{\in}A.\ p\ (f'\ x) = b$
from *this* **show** $\forall b{\in}B.\ \exists x{\in}A.\ f'\ x = b$
using $\langle p$ *permutes* $B\rangle$ **by** (*metis bij-betwE permutes-imp-bij permutes-inverses*(*2*))
qed
also have $\neg\ (\forall b{\in}B.\ \exists x{\in}A.\ p\ (f'\ x) = b) \longleftrightarrow \{\} \in (\lambda b.\ \{x \in A.\ p\ (f'\ x) = b\})$ ‘ B **by** *auto*
finally have $\{\} \in (\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B \longleftrightarrow \{\} \in (\lambda b.\ \{x \in A.\ p\ (f'\ x) = b\})$ ‘ B **.**
moreover have $(\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B = (\lambda b.\ \{x \in A.\ p\ (f'\ x) = b\})$ ‘ B
using $\langle p$ *permutes* $B\rangle$ *permutes-implies-inv-image-on-eq* **by** *blast*
ultimately have $(\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\} = (\lambda b.\ \{x \in A.\ p\ (f'\ x) = b\})$ ‘ $B - \{\{\}\}$ **by** *auto*
also have $\ldots = (\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\}$
using $\langle\forall x \in A.\ f\ x = p\ (f'\ x)\rangle$ *Collect-cong image-cong* **by** *auto*
finally show $(\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\} = (\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\}$ **..**
qed

3.3 Permutation on the Domain and the Range

definition *domain-and-range-permutation*
where
domain-and-range-permutation $A\ B = \{(f, f') \in (A \rightarrow_E B) \times (A \rightarrow_E B).$
$\exists p_A\ p_B.\ p_A$ *permutes* $A \wedge p_B$ *permutes* $B \wedge (\forall x \in A.\ f\ x = p_B\ (f'\ (p_A\ x)))\}$

lemma *equiv-domain-and-range-permutation*:
  equiv $(A \rightarrow_E B)$ (*domain-and-range-permutation* A B)
proof (*rule equivI*)
  show *domain-and-range-permutation* A $B \subseteq (A \rightarrow_E B) \times (A \rightarrow_E B)$
    unfolding *domain-and-range-permutation-def* **by** *auto*
next
  show *refl-on* $(A \rightarrow_E B)$ (*domain-and-range-permutation* A B)
  proof (*rule refl-onI*)
    fix f
    assume $f \in A \rightarrow_E B$
    from *this* **show** $(f, f) \in$ *domain-and-range-permutation* A B
      using *permutes-id*[*of* A] *permutes-id*[*of* B]
      unfolding *domain-and-range-permutation-def* **by** *fastforce*
  qed
next
  show *sym* (*domain-and-range-permutation* A B)
  proof (*rule symI*)
    fix $f\, f'$
    assume $(f, f') \in$ *domain-and-range-permutation* A B
    from *this* **obtain** p_A p_B **where** p_A *permutes* A p_B *permutes* B **and** $\forall x \in A.\ f\ x = p_B\ (f'\ (p_A\ x))$
      unfolding *domain-and-range-permutation-def* **by** *auto*
    from ‹$(f, f') \in$ *domain-and-range-permutation* A B› **have** f: $f \in A \rightarrow_E B$ $f' \in A \rightarrow_E B$
      unfolding *domain-and-range-permutation-def* **by** *auto*
    moreover from ‹p_A *permutes* A› ‹p_B *permutes* B› **have** *inv* p_A *permutes* A *inv* p_B *permutes* B
      by (*auto simp add*: *permutes-inv*)
    moreover from ‹$\forall x \in A.\ f\ x = p_B\ (f'\ (p_A\ x))$› **have** $\forall x \in A.\ f'\ x = inv\ p_B\ (f\ (inv\ p_A\ x))$
     using ‹p_A *permutes* A› ‹p_B *permutes* B› ‹*inv* p_A *permutes* A› ‹*inv* p_B *permutes* B›
     by (*metis* (*no-types*, *lifting*) *bij-betwE bij-inv-eq-iff permutes-bij permutes-imp-bij*)
    ultimately show $(f', f) \in$ *domain-and-range-permutation* A B
      unfolding *domain-and-range-permutation-def* **by** *auto*
  qed
next
  show *trans* (*domain-and-range-permutation* A B)
  proof (*rule transI*)
    fix $f\, f'\, f''$
    assume $(f, f') \in$ *domain-and-range-permutation* A B
    assume $(f', f'') \in$ *domain-and-range-permutation* A B
    from ‹$(f, f') \in$ -› **obtain** p_A p_B **where**
      p_A *permutes* A p_B *permutes* B **and** $\forall x \in A.\ f\ x = p_B\ (f'\ (p_A\ x))$
      unfolding *domain-and-range-permutation-def* **by** *auto*
    from ‹$(f', f'') \in$ -› **obtain** p'_A p'_B **where**
      p'_A *permutes* A p'_B *permutes* B **and** $\forall x \in A.\ f'\ x = p'_B\ (f''\ (p'_A\ x))$
      unfolding *domain-and-range-permutation-def* **by** *auto*
    from ‹$(f, f') \in$ *domain-and-range-permutation* A B› **have** $f \in A \rightarrow_E B$

unfolding *domain-and-range-permutation-def* **by** *auto*
moreover from ‹$(f', f'') \in$ *domain-and-range-permutation A B*› **have** $f'' \in A \to_E B$
unfolding *domain-and-range-permutation-def* **by** *auto*
moreover from ‹p_A *permutes A*› ‹p'_A *permutes A*› **have** $(p'_A \circ p_A)$ *permutes A*
by (*simp add*: *permutes-compose*)
moreover from ‹p_B *permutes B*› ‹p'_B *permutes B*› **have** $(p_B \circ p'_B)$ *permutes B*
by (*simp add*: *permutes-compose*)
moreover have $\forall x{\in}A.\ f\ x = (p_B \circ p'_B)\ (f''\ ((p'_A \circ p_A)\ x))$
using ‹$\forall x{\in}A.\ f'\ x = p'_B\ (f''\ (p'_A\ x))$› ‹$\forall x{\in}A.\ f\ x = p_B\ (f'\ (p_A\ x))$› ‹$p_A$ *permutes A*›
by (*simp add*: *permutes-in-image*)
ultimately show $(f, f'') \in$ *domain-and-range-permutation A B*
unfolding *domain-and-range-permutation-def* **by** *fastforce*
qed
qed

3.3.1 Respecting Functions

lemma *inj-on-respects-domain-and-range-permutation*:
(λf. *inj-on f A*) *respects domain-and-range-permutation A B*
proof (*rule congruentI*)
fix $f\ f'$
assume $(f, f') \in$ *domain-and-range-permutation A B*
from *this* **obtain** $p_A\ p_B$ **where** p_A *permutes A* p_B *permutes B* **and** $\forall x{\in}A.\ f\ x = p_B\ (f'\ (p_A\ x))$
unfolding *domain-and-range-permutation-def* **by** *auto*
from ‹$(f, f') \in$ *domain-and-range-permutation A B*› **have** f' ‘ $A \subseteq B$
unfolding *domain-and-range-permutation-def* **by** *auto*
from ‹p_A *permutes A*› **have** p_A ‘ $A = A$ **by** (*auto simp add*: *permutes-image*)
from ‹p_A *permutes A*› **have** *inj-on* p_A *A*
using *bij-betw-imp-inj-on permutes-imp-bij* **by** *blast*
from ‹p_B *permutes B*› **have** *inj-on* p_B *B*
using *bij-betw-imp-inj-on permutes-imp-bij* **by** *blast*
show *inj-on f A* $\longleftrightarrow$ *inj-on* f' *A*
proof −
have *inj-on f A* $\longleftrightarrow$ *inj-on* $(\lambda x.\ p_B\ (f'\ (p_A\ x)))$ *A*
using ‹$\forall x{\in}A.\ f\ x = p_B\ (f'\ (p_A\ x))$› *inj-on-cong comp-apply* **by** *fastforce*
have *inj-on f A* $\longleftrightarrow$ *inj-on* $(p_B\ o\ f'\ o\ p_A)$ *A*
by (*simp add*: ‹$\forall x{\in}A.\ f\ x = p_B\ (f'\ (p_A\ x))$› *inj-on-def*)
also have *inj-on* $(p_B\ o\ f'\ o\ p_A)$ *A* $\longleftrightarrow$ *inj-on* $(p_B\ o\ f')$ *A*
using ‹*inj-on* p_A *A*› ‹p_A ‘ $A = A$›
by (*auto dest*: *inj-on-imageI intro*: *comp-inj-on*)
also have *inj-on* $(p_B\ o\ f')$ *A* $\longleftrightarrow$ *inj-on* f' *A*
using ‹*inj-on* p_B *B*› ‹f' ‘ $A \subseteq B$›
by (*auto dest*: *inj-on-imageI2 intro*: *comp-inj-on inj-on-subset*)
finally show *?thesis* **.**

qed
qed

lemma *surjective-respects-domain-and-range-permutation*:
$(\lambda f.\ f\ `\ A = B)$ *respects domain-and-range-permutation* $A\ B$
proof (*rule congruentI*)
fix $f\ f'$
assume $(f, f') \in$ *domain-and-range-permutation* $A\ B$
from *this* **obtain** $p_A\ p_B$ **where**
permutes: p_A *permutes* A p_B *permutes* B **and** $\forall x \in A.\ f\ x = p_B\ (f'\ (p_A\ x))$
unfolding *domain-and-range-permutation-def* **by** *auto*
from *permutes* **have** $p_A\ `\ A = A$ $p_B\ `\ B = B$ **by** (*auto simp add*: *permutes-image*)
from ‹p_B *permutes* B› **have** *inj* p_B **by** (*simp add*: *permutes-inj*)
show $(f\ `\ A = B) \longleftrightarrow (f'\ `\ A = B)$
proof −
have $f\ `\ A = B \longleftrightarrow (\lambda x.\ p_B\ (f'\ (p_A\ x)))\ `\ A = B$
using ‹$\forall x \in A.\ f\ x = p_B\ (f'\ (p_A\ x))$› **by** (*metis* (*mono-tags*, *lifting*) *image-cong*)
also have $(\lambda x.\ p_B\ (f'\ (p_A\ x)))\ `\ A = B \longleftrightarrow (\lambda x.\ p_B\ (f'\ x))\ `\ A = B$
using ‹$p_A\ `\ A = A$› **by** (*metis image-image*)
also have $(\lambda x.\ p_B\ (f'\ x))\ `\ A = B \longleftrightarrow (f'\ `\ A = B)$
using ‹$p_B\ `\ B = B$› ‹*inj* p_B› **by** (*metis image-image image-inv-f-f*)
finally show *?thesis* **.**
qed
qed

lemma *bij-betw-respects-domain-and-range-permutation*:
$(\lambda f.$ *bij-betw* $f\ A\ B)$ *respects domain-and-range-permutation* $A\ B$
proof (*rule congruentI*)
fix $f\ f'$
assume $(f, f') \in$ *domain-and-range-permutation* $A\ B$
from *this* **obtain** $p_A\ p_B$ **where** p_A *permutes* A p_B *permutes* B
and $\forall x \in A.\ f\ x = p_B\ (f'\ (p_A\ x))$ **and** $f' \in A \rightarrow_E B$
unfolding *domain-and-range-permutation-def* **by** *auto*
have *bij-betw* $f\ A\ B \longleftrightarrow$ *bij-betw* $(p_B\ o\ f'\ o\ p_A)\ A\ B$
using ‹$\forall x \in A.\ f\ x = p_B\ (f'\ (p_A\ x))$› *bij-betw-congI* **by** *fastforce*
also have ... $\longleftrightarrow$ *bij-betw* $(p_B\ o\ f')\ A\ B$
using ‹p_A *permutes* A›
by (*auto intro*!: *bij-betw-comp-iff*[*symmetric*] *permutes-imp-bij*)
also have ... $\longleftrightarrow$ *bij-betw* $f'\ A\ B$
using ‹$f' \in A \rightarrow_E B$› ‹p_B *permutes* B›
by (*auto intro*!: *bij-betw-comp-iff2*[*symmetric*] *permutes-imp-bij*)
finally show *bij-betw* $f\ A\ B \longleftrightarrow$ *bij-betw* $f'\ A\ B$ **.**
qed

lemma *count-image-mset'*:
count (*image-mset* $f\ A$) x = *sum* (*count* A) $\{x' \in$ *set-mset* $A.\ f\ x' = x\}$
proof −
have *count* (*image-mset* $f\ A$) x = *sum* (*count* A) $(f\ -`\ \{x\} \cap$ *set-mset* $A)$
unfolding *count-image-mset* **..**

also have ... = *sum* (*count* *A*) {x' ∈ *set-mset* *A*. *f* x' = *x*}
proof −
have (*f* −' {*x*} ∩ *set-mset* *A*) = {x' ∈ *set-mset* *A*. *f* x' = *x*} **by** *blast*
from *this* **show** *?thesis* **by** *simp*
qed
finally show *?thesis* **.**
qed

lemma *multiset-of-partition-cards-respects-domain-and-range-permutation*:
assumes *finite B*
shows (λ*f*. *image-mset* (λ*X*. *card X*) (*mset-set* (((λ*b*. {*x* ∈ *A*. *f x* = *b*})) ' *B* − {{}}))) *respects domain-and-range-permutation A B*
proof (*rule congruentI*)
fix *f* f'
assume (*f*, f') ∈ *domain-and-range-permutation A B*
from *this* **obtain** p_A p_B **where** p_A *permutes A* p_B *permutes B* ∀ *x*∈*A*. *f x* = p_B (f' (p_A *x*))
unfolding *domain-and-range-permutation-def* **by** *auto*
have (λ*b*. {*x* ∈ *A*. *f x* = *b*}) ' *B* = (λ*b*. {*x* ∈ *A*. p_B (f' (p_A *x*)) = *b*}) ' *B*
using ‹∀ *x*∈*A*. *f x* = p_B (f' (p_A *x*))› **by** *auto*
from *this* **have** *image-mset card* (*mset-set* ((λ*b*. {*x* ∈ *A*. *f x* = *b*}) ' *B* − {{}})) =
image-mset card (*mset-set* ((λ*b*. {*x* ∈ *A*. p_B (f' (p_A *x*)) = *b*}) ' *B* − {{}})) **by** *simp*
also have *image-mset card* (*mset-set* ((λ*b*. {*x* ∈ *A*. p_B (f' (p_A *x*)) = *b*}) ' *B* − {{}})) =
image-mset card (*mset-set* ((λ*b*. {*x* ∈ *A*. f' (p_A *x*) = *b*}) ' *B* − {{}}))
using *permutes-implies-inv-image-on-eq*[*OF* ‹p_B *permutes B*›, *of A*] **by** *metis*
also have *image-mset card* (*mset-set* ((λ*b*. {*x* ∈ *A*. f' (p_A *x*) = *b*}) ' *B* − {{}})) =
image-mset card (*mset-set* ((λ*b*. {*x* ∈ *A*. f' *x* = *b*}) ' *B* − {{}}))
proof (*rule multiset-eqI*)
fix *n*
have *bij-betw* (λ*X*. p_A ' *X*) {*X* ∈ (λ*b*. {*x* ∈ *A*. f' (p_A *x*) = *b*}) ' *B* − {{}}. *card X* = *n*} {*X* ∈ (λ*b*. {*x* ∈ *A*. f' *x* = *b*}) ' *B* − {{}}. *card X* = *n*}
proof (*rule bij-betw-byWitness*)
show ∀ *X*∈{*X* ∈ (λ*b*. {*x* ∈ *A*. f' (p_A *x*) = *b*}) ' *B* − {{}}. *card X* = *n*}. *inv* p_A ' p_A ' *X* = *X*
by (*meson* ‹p_A *permutes A*› *image-inv-f-f permutes-inj*)
show ∀ *X*∈{*X* ∈ (λ*b*. {*x* ∈ *A*. f' *x* = *b*}) ' *B* − {{}}. *card X* = *n*}. p_A ' *inv* p_A ' *X* = *X*
by (*meson* ‹p_A *permutes A*› *image-f-inv-f permutes-surj*)
show (λ*X*. p_A ' *X*) ' {*X* ∈ (λ*b*. {*x* ∈ *A*. f' (p_A *x*) = *b*}) ' *B* − {{}}. *card X* = *n*} ⊆ {*X* ∈ (λ*b*. {*x* ∈ *A*. f' *x* = *b*}) ' *B* − {{}}. *card X* = *n*}
proof −
have *card* (p_A ' {*x* ∈ *A*. f' (p_A *x*) = *b*}) = *card* {*x* ∈ *A*. f' (p_A *x*) = *b*} **for** *b*
proof −
have *inj-on* p_A {*x* ∈ *A*. f' (p_A *x*) = *b*}

by (*metis* (*no-types*, *lifting*) ‹p_A *permutes* A› *injD inj-onI permutes-inj*)
from *this* **show** *?thesis* **by** (*simp add*: *card-image*)
qed
moreover have p_A ‘ $\{x \in A.\ f'\ (p_A\ x) = b\} = \{x \in A.\ f'\ x = b\}$ **for** b
proof
show p_A ‘ $\{x \in A.\ f'\ (p_A\ x) = b\} \subseteq \{x \in A.\ f'\ x = b\}$
by (*auto simp add*: ‹p_A *permutes* A› *permutes-in-image*)
show $\{x \in A.\ f'\ x = b\} \subseteq p_A$ ‘ $\{x \in A.\ f'\ (p_A\ x) = b\}$
proof
fix x
assume $x \in \{x \in A.\ f'\ x = b\}$
moreover have p_A (*inv* p_A x) = x
using ‹p_A *permutes* A› *permutes-inverses*(*1*) **by** *fastforce*
moreover from ‹$x \in \{x \in A.\ f'\ x = b\}$› **have** *inv* p_A $x \in A$
by (*simp add*: ‹p_A *permutes* A› *permutes-in-image permutes-inv*)
ultimately show $x \in p_A$ ‘ $\{x \in A.\ f'\ (p_A\ x) = b\}$
by (*auto intro*: *image-eqI*[**where** x=*inv* p_A x])
qed
qed
ultimately show *?thesis* **by** *auto*
qed
show (λX. *inv* p_A ‘ X) ‘ $\{X \in (\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\}.$ *card* $X = n\} \subseteq \{X \in (\lambda b.\ \{x \in A.\ f'\ (p_A\ x) = b\})$ ‘ $B - \{\{\}\}.$ *card* $X = n\}$
proof −
have *card* (*inv* p_A ‘ $\{x \in A.\ f'\ x = b\}$) = *card* $\{x \in A.\ f'\ x = b\}$ **for** b
proof −
have *inj-on* (*inv* p_A) $\{x \in A.\ f'\ x = b\}$
by (*metis* (*no-types*, *lifting*) ‹p_A *permutes* A› *injD inj-onI permutes-surj surj-imp-inj-inv*)
from *this* **show** *?thesis* **by** (*simp add*: *card-image*)
qed
moreover have *inv* p_A ‘ $\{x \in A.\ f'\ x = b\} = \{x \in A.\ f'\ (p_A\ x) = b\}$ **for** b
proof
show *inv* p_A ‘ $\{x \in A.\ f'\ x = b\} \subseteq \{x \in A.\ f'\ (p_A\ x) = b\}$
using ‹p_A *permutes* A›
by (*auto simp add*: *permutes-in-image permutes-inv permutes-inverses*(*1*))
show $\{x \in A.\ f'\ (p_A\ x) = b\} \subseteq$ *inv* p_A ‘ $\{x \in A.\ f'\ x = b\}$
proof
fix x
assume $x \in \{x \in A.\ f'\ (p_A\ x) = b\}$
moreover have *inv* p_A (p_A x) = x
by (*meson* ‹p_A *permutes* A› *permutes-inverses*(*2*))
moreover from ‹$x \in \{x \in A.\ f'\ (p_A\ x) = b\}$› **have** p_A $x \in A$
by (*simp add*: ‹p_A *permutes* A› *permutes-in-image*)
ultimately show $x \in$ *inv* p_A ‘ $\{x \in A.\ f'\ x = b\}$
by (*auto intro*: *image-eqI*[**where** x=p_A x])
qed
qed
ultimately show *?thesis* **by** *auto*

qed
qed
from *this* **have** $card\ \{x' \in (\lambda b.\ \{x \in A.\ f'\ (p_A\ x) = b\})\ `\ B - \{\{\}\}.\ card\ x' = n\} = card\ \{x' \in (\lambda b.\ \{x \in A.\ f'\ x = b\})\ `\ B - \{\{\}\}.\ card\ x' = n\}$
by (*rule bij-betw-same-card*)
from *this* **show** $count\ (image\text{-}mset\ card\ (mset\text{-}set\ ((\lambda b.\ \{x \in A.\ f'\ (p_A\ x) = b\})\ `\ B - \{\{\}\})))\ n =$
$count\ (image\text{-}mset\ card\ (mset\text{-}set\ ((\lambda b.\ \{x \in A.\ f'\ x = b\})\ `\ B - \{\{\}\})))\ n$
using ‹*finite B*› **by** (*simp add*: *count-image-mset'*)
qed
finally show $image\text{-}mset\ card\ (mset\text{-}set\ ((\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\})) =$
$image\text{-}mset\ card\ (mset\text{-}set\ ((\lambda b.\ \{x \in A.\ f'\ x = b\})\ `\ B - \{\{\}\}))$ **.**
qed

end

4 Functions from A to B

theory *Twelvefold-Way-Entry1*
imports *Preliminaries*
begin

Note that the cardinality theorems of both structures, lists and finite functions, are already available. Hence, this development creates the bijection between those two structures and transfers the one cardinality theorem to the other structures and vice versa, although not strictly needed as both cardinality theorems were already available.

4.1 Definition of Bijections

definition *sequence-of* :: $'a\ set \Rightarrow (nat \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b\ list$
where
$sequence\text{-}of\ A\ enum\ f = map\ (\lambda n.\ f\ (enum\ n))\ [0..<card\ A]$

definition *function-of* :: $'a\ set \Rightarrow (nat \Rightarrow 'a) \Rightarrow 'b\ list \Rightarrow ('a \Rightarrow 'b)$
where
$function\text{-}of\ A\ enum\ xs = (\lambda a.\ if\ a \in A\ then\ xs\ !\ inv\text{-}into\ \{0..<length\ xs\}\ enum\ a\ else\ undefined)$

4.2 Properties for Bijections

lemma *nth-sequence-of*:
assumes $i < card\ A$
shows $(sequence\text{-}of\ A\ enum\ f)\ !\ i = f\ (enum\ i)$
using *assms* **unfolding** *sequence-of-def* **by** *auto*

lemma *nth-sequence-of-inv-into*:
assumes $bij\text{-}betw\ enum\ \{0..<card\ A\}\ A$

```
  assumes a ∈ A
  shows (sequence-of A enum f) ! (inv-into {0..<card A} enum a) = f a
proof −
  have inv-into {0..<card A} enum a ∈ {0..<card A}
    using assms bij-betwE bij-betw-inv-into by blast
  from this assms show (sequence-of A enum f) ! (inv-into {0..<card A} enum a)
= f a
    unfolding sequence-of-def by (simp add: bij-betw-inv-into-right)
qed

lemma set-sequence-of:
  assumes bij-betw enum {0..<card A} A
  assumes f ∈ A →E B
  shows set (sequence-of A enum f) ⊆ B
using PiE bij-betwE assms
unfolding sequence-of-def by fastforce

lemma length-sequence-of:
  assumes bij-betw enum {0..<card A} A
  assumes f ∈ A →E B
  shows length (sequence-of A enum f) = card A
using assms unfolding sequence-of-def by simp

lemma function-of-enum:
  assumes bij-betw enum {0..<card A} A
  assumes length xs = card A
  assumes i < card A
  shows function-of A enum xs (enum i) = xs ! i
using assms unfolding function-of-def
by (auto simp add: bij-betw-inv-into-left bij-betwE)

lemma function-of-in-extensional-funcset:
  assumes bij-betw enum {0..<card A} A
  assumes set xs ⊆ B length xs = card A
  shows function-of A enum xs ∈ A →E B
proof
  fix x
  assume x ∈ A
  have inv-into {0..<length xs} enum x ∈ {0..<length xs}
    using ‹x ∈ A› assms(1, 3) by (metis bij-betw-def inv-into-into)
  from this have xs ! inv-into {0..<length xs} enum x ∈ set xs by simp
  from this ‹set xs ⊆ B› show function-of A enum xs x ∈ B
    using ‹x ∈ A› unfolding function-of-def by auto
next
  fix x
  assume x ∉ A
  from this show function-of A enum xs x = undefined
    unfolding function-of-def by simp
qed
```

lemma *sequence-of-function-of*:
  assumes *bij-betw enum* {*0*..<*card A*} *A*
  assumes *set xs* ⊆ *B length xs* = *card A*
  shows *sequence-of A enum* (*function-of A enum xs*) = *xs*
proof (*rule nth-equalityI*)
  have *function-of A enum xs* ∈ $A \to_E B$
    using *assms* **by** (*rule function-of-in-extensional-funcset*)
  from *this* **show** *length* (*sequence-of A enum* (*function-of A enum xs*)) = *length xs*
    using *assms*(*1*,*3*) **by** (*simp add*: *length-sequence-of*)
  from *this* **show** ⋀*i*. *i* < *length* (*sequence-of A enum* (*function-of A enum xs*)) ⟹ *sequence-of A enum* (*function-of A enum xs*) ! *i* = *xs* ! *i*
    using *assms* **by** (*auto simp add*: *nth-sequence-of function-of-enum*)
qed

lemma *function-of-sequence-of*:
  assumes *bij-betw enum* {*0*..<*card A*} *A*
  assumes *f* ∈ $A \to_E B$
  shows *function-of A enum* (*sequence-of A enum f*) = *f*
proof
  fix *x*
  show *function-of A enum* (*sequence-of A enum f*) *x* = *f x*
    using *assms* **unfolding** *function-of-def*
    by (*auto simp add*: *length-sequence-of nth-sequence-of-inv-into*)
qed

4.3 Bijections

lemma *bij-betw-sequence-of*:
  assumes *bij-betw enum* {*0*..<*card A*} *A*
  shows *bij-betw* (*sequence-of A enum*) ($A \to_E B$) {*xs*. *set xs* ⊆ *B* ∧ *length xs* = *card A*}
proof (*rule bij-betw-byWitness*[**where** *f'*=*function-of A enum*])
  show ∀*f*∈$A \to_E B$. *function-of A enum* (*sequence-of A enum f*) = *f*
    using *assms* **by** (*simp add*: *function-of-sequence-of*)
  show ∀*xs*∈{*xs*. *set xs* ⊆ *B* ∧ *length xs* = *card A*}. *sequence-of A enum* (*function-of A enum xs*) = *xs*
    using *assms* **by** (*auto simp add*: *sequence-of-function-of*)
  show *sequence-of A enum* ‘ ($A \to_E B$) ⊆ {*xs*. *set xs* ⊆ *B* ∧ *length xs* = *card A*}
    using *assms set-sequence-of*[*OF assms*] *length-sequence-of* **by** *auto*
  show *function-of A enum* ‘ {*xs*. *set xs* ⊆ *B* ∧ *length xs* = *card A*} ⊆ $A \to_E B$
    using *assms function-of-in-extensional-funcset* **by** *blast*
qed

lemma *bij-betw-function-of*:
  assumes *bij-betw enum* {*0*..<*card A*} *A*
  shows *bij-betw* (*function-of A enum*) {*xs*. *set xs* ⊆ *B* ∧ *length xs* = *card A*} ($A \to_E B$)

proof (*rule bij-betw-byWitness*[**where** f'=*sequence-of A enum*])
show $\forall f \in A \rightarrow_E B$. *function-of A enum* (*sequence-of A enum f*) = *f*
using *assms* **by** (*simp add*: *function-of-sequence-of*)
show $\forall xs \in \{xs.\ set\ xs \subseteq B \wedge length\ xs = card\ A\}$. *sequence-of A enum* (*function-of A enum xs*) = *xs*
using *assms* **by** (*auto simp add*: *sequence-of-function-of*)
show *sequence-of A enum* ‘ $(A \rightarrow_E B) \subseteq \{xs.\ set\ xs \subseteq B \wedge length\ xs = card\ A\}$
using *assms set-sequence-of*[*OF assms*] *length-sequence-of* **by** *auto*
show *function-of A enum* ‘ $\{xs.\ set\ xs \subseteq B \wedge length\ xs = card\ A\} \subseteq A \rightarrow_E B$
using *assms function-of-in-extensional-funcset* **by** *blast*
qed

4.4 Cardinality

lemma
assumes *finite A*
shows *card* $(A \rightarrow_E B)$ = *card B* ^ *card A*
proof –
obtain *enum* **where** *bij-betw enum* {*0..<card A*} *A*
using ‹*finite A*› *ex-bij-betw-nat-finite* **by** *blast*
have *bij-betw* (*sequence-of A enum*) $(A \rightarrow_E B)$ $\{xs.\ set\ xs \subseteq B \wedge length\ xs = card\ A\}$
using ‹*bij-betw enum* {*0..<card A*} *A*› **by** (*rule bij-betw-sequence-of*)
from *this* **have** *card* $(A \rightarrow_E B)$ = *card* $\{xs.\ set\ xs \subseteq B \wedge length\ xs = card\ A\}$
by (*rule bij-betw-same-card*)
also have *card* $\{xs.\ set\ xs \subseteq B \wedge length\ xs = card\ A\}$ = *card B* ^ *card A*
by (*rule card-lists-length-eq*)
finally show *?thesis* **.**
qed

lemma *card-sequences*:
assumes *finite A*
shows *card* $\{xs.\ set\ xs \subseteq B \wedge length\ xs = card\ A\}$ = *card B* ^ *card A*
proof –
obtain *enum* **where** *bij-betw enum* {*0..<card A*} *A*
using ‹*finite A*› *ex-bij-betw-nat-finite* **by** *blast*
have *bij-betw* (*function-of A enum*) $\{xs.\ set\ xs \subseteq B \wedge length\ xs = card\ A\}$ $(A \rightarrow_E B)$
using ‹*bij-betw enum* {*0..<card A*} *A*› **by** (*rule bij-betw-function-of*)
from *this* **have** *card* $\{xs.\ set\ xs \subseteq B \wedge length\ xs = card\ A\}$ = *card* $(A \rightarrow_E B)$
by (*rule bij-betw-same-card*)
also have *card* $(A \rightarrow_E B)$ = *card B* ^ *card A*
using ‹*finite A*› **by** (*rule card-extensional-funcset*)
finally show *?thesis* **.**
qed

lemma
shows *card* $\{xs.\ set\ xs \subseteq A \wedge length\ xs = n\}$ = *card A* ^ *n*
proof –

```
  have card {xs. set xs ⊆ A ∧ length xs = n} = card {xs. set xs ⊆ A ∧ length xs
= card {0..<n}}
    by auto
  also have ... = card A ^ card {0..<n} by (subst card-sequences) auto
  also have ... = card A ^ n by auto
  finally show ?thesis .
qed

end
```

5 Injections from A to B

```
theory Twelvefold-Way-Entry2
imports Twelvefold-Way-Entry1
begin
```

Note that the cardinality theorems of both structures, distinct lists and finite injective functions, are already available. Hence, this development creates the bijection between those two structures and transfers the one cardinality theorem to the other structures and vice versa, although not strictly needed as both cardinality theorems were already available.

5.1 Properties for Bijections

```
lemma inj-on-implies-distinct:
  assumes bij-betw enum {0..<card A} A
  assumes f ∈ A →E B
  assumes inj-on f A
  shows distinct (sequence-of A enum f)
proof –
  {
    fix i j
    assume bounds: i < length (sequence-of A enum f) j < length (sequence-of A
enum f)
    assume i ≠ j
    from bounds assms(1, 2) have bounds': i < card A j < card A
      using length-sequence-of by fastforce+
    from this assms(1) have in-A: enum i ∈ A enum j ∈ A
      using bij-betwE by fastforce+
    from ‹i ≠ j› bounds' assms(1) have enum i ≠ enum j
      by (metis bij-betw-inv-into-left lessThan-iff atLeast0LessThan)
    from this have f (enum i) ≠ f (enum j)
      using assms(3) in-A inj-onD by fastforce
    from this bounds' have sequence-of A enum f ! i ≠ sequence-of A enum f ! j
      by (simp add: nth-sequence-of)
  }
  from this show ?thesis
    by (auto simp add: distinct-conv-nth)
```

qed

lemma *distinct-implies-inj-on*:
  assumes *bij-betw enum* $\{0..<card\ A\}$ *A*
  assumes *length xs* = *card A*
  assumes *distinct xs*
  shows *inj-on* (*function-of A enum xs*) *A*
proof (*rule inj-onI*)
  let *?idx-of* = λx. *inv-into* $\{0..<length\ xs\}$ *enum x*
  fix *x y*
  assume $x \in A$ $y \in A$ *function-of A enum xs x* = *function-of A enum xs y*
  from *this* **have** *xs* ! *?idx-of x* = *xs* ! *?idx-of y*
    unfolding *function-of-def* **by** *simp*
  have *?idx-of x* = *?idx-of y*
  proof −
    have *?idx-of x* < *length xs*
      using ‹$x \in A$› *assms*(*1*,*2*)
      by (*metis atLeast0LessThan bij-betw-imp-surj-on inv-into-into lessThan-iff*)
    moreover have *?idx-of y* < *length xs*
      using ‹$y \in A$› *assms*(*1*,*2*)
      by (*metis atLeast0LessThan bij-betw-imp-surj-on inv-into-into lessThan-iff*)
    moreover note ‹*xs* ! *?idx-of x* = *xs* ! *?idx-of y*› ‹*distinct xs*›
    ultimately show *?thesis*
      by (*auto dest*: *nth-eq-iff-index-eq*[**where** *i*=*?idx-of x* **and** *j*=*?idx-of y*])
  qed
  from *this* ‹*bij-betw* - - -› **show** *x* = *y*
    by (*metis* ‹$x \in A$› ‹$y \in A$› ‹*length xs* = *card A*› *bij-betw-inv-into-right*)
qed

lemma *image-sequence-of-inj*:
  assumes *bij-betw enum* $\{0..<card\ A\}$ *A*
  shows *sequence-of A enum* ‘ $\{f \in A \rightarrow_E B.\ inj\text{-}on\ f\ A\} \subseteq \{xs.\ set\ xs \subseteq B \wedge length\ xs = card\ A \wedge distinct\ xs\}$
proof
  fix *xs*
  assume $xs \in$ *sequence-of A enum* ‘ $\{f \in A \rightarrow_E B.\ inj\text{-}on\ f\ A\}$
  from *this* **obtain** *f* **where** *xs*: *xs* = *sequence-of A enum f* **and** *f*: $f \in A \rightarrow_E B$ *inj-on f A* **by** *auto*
  moreover from *xs f* ‹*bij-betw* - - -› **have** *set xs* ⊆ *B*
    using *set-sequence-of subsetCE* **by** *blast*
  moreover from *xs f* ‹*bij-betw* - - -› **have** *length xs* = *card A*
    using *length-sequence-of* **by** *auto*
  moreover from *xs f* ‹*bij-betw* - - -› **have** *distinct xs*
    using *inj-on-implies-distinct* **by** *simp*
  ultimately show $xs \in \{xs.\ set\ xs \subseteq B \wedge length\ xs = card\ A \wedge distinct\ xs\}$ **by** *auto*
qed

lemma *image-function-of-distinct*:

assumes *bij-betw enum* {*0..<card A*} *A*
shows *function-of A enum* ‘ {*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*} ⊆ {*f* ∈ *A* →$_E$ *B. inj-on f A*}
proof
fix *f*
assume *f*: *f* ∈ *function-of A enum* ‘ {*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*}
from *f assms* **have** *f* ∈ *A* →$_E$ *B*
using *function-of-in-extensional-funcset* **by** *blast*
moreover from *f assms* **have** *inj-on f A*
by (*auto simp add*: *assms distinct-implies-inj-on*)
ultimately show *f* ∈ {*f* ∈ *A* →$_E$ *B. inj-on f A*} **by** *auto*
qed

5.2 Bijections

lemma *bij-betw-sequence-of*:
assumes *bij-betw enum* {*0..<card A*} *A*
shows *bij-betw* (*sequence-of A enum*) {*f. f* ∈ *A* →$_E$ *B* ∧ *inj-on f A*} {*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*}
proof (*rule bij-betw-byWitness*[**where** *f'*=*function-of A enum*])
show ∀ *f*∈{*f* ∈ *A* →$_E$ *B. inj-on f A*}. *function-of A enum* (*sequence-of A enum f*) = *f*
using *assms* **by** (*auto simp add*: *function-of-sequence-of*)
show ∀ *xs*∈{*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*}. *sequence-of A enum* (*function-of A enum xs*) = *xs*
using *assms* **by** (*auto simp add*: *sequence-of-function-of*)
show *sequence-of A enum* ‘ {*f* ∈ *A* →$_E$ *B. inj-on f A*} ⊆ {*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*}
using *assms* **by** (*simp add*: *image-sequence-of-inj*)
show *function-of A enum* ‘ {*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*} ⊆ {*f* ∈ *A* →$_E$ *B. inj-on f A*}
using *assms* **by** (*simp add*: *image-function-of-distinct*)
qed

lemma *bij-betw-function-of*:
assumes *bij-betw enum* {*0..<card A*} *A*
shows *bij-betw* (*function-of A enum*) {*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*} {*f* ∈ *A* →$_E$ *B. inj-on f A*}
proof (*rule bij-betw-byWitness*[**where** *f'*=*sequence-of A enum*])
show ∀ *f*∈{*f* ∈ *A* →$_E$ *B. inj-on f A*}. *function-of A enum* (*sequence-of A enum f*) = *f*
using *assms* **by** (*auto simp add*: *function-of-sequence-of*)
show ∀ *xs*∈{*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*}. *sequence-of A enum* (*function-of A enum xs*) = *xs*
using *assms* **by** (*auto simp add*: *sequence-of-function-of*)
show *sequence-of A enum* ‘ {*f* ∈ *A* →$_E$ *B. inj-on f A*} ⊆ {*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*}
using *assms* **by** (*simp add*: *image-sequence-of-inj*)

show *function-of A enum* ‘ {*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*} ⊆ {*f* ∈ *A* $\to_E$ *B. inj-on f A*}
  using *assms* **by** (*simp add*: *image-function-of-distinct*)
qed

5.3 Cardinality

lemma
  assumes *finite A finite B card A* ≤ *card B*
  shows *card* {*f* ∈ *A* $\to_E$ *B. inj-on f A*} = ∏{*card B* − *card A* + *1..card B*}
proof −
  obtain *enum* **where** *bij-betw enum* {*0..<card A*} *A*
    using ‹*finite A*› *ex-bij-betw-nat-finite* **by** *blast*
  have *bij-betw* (*sequence-of A enum*) {*f* ∈ *A* $\to_E$ *B. inj-on f A*} {*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*}
    using ‹*bij-betw enum* {*0..<card A*} *A*› **by** (*rule bij-betw-sequence-of*)
  from *this* **have** *card* {*f* ∈ *A* $\to_E$ *B. inj-on f A*} = *card* {*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*}
    by (*rule bij-betw-same-card*)
  also have *card* {*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*} = *card* {*xs. length xs* = *card A* ∧ *distinct xs* ∧ *set xs* ⊆ *B*}
    by *meson*
  also have *card* {*xs. length xs* = *card A* ∧ *distinct xs* ∧ *set xs* ⊆ *B*} = ∏{*card B* − *card A* + *1..card B*}
    using ‹*finite B*› ‹*card A* ≤ *card B*› **by** (*rule List.card-lists-distinct-length-eq*)
  finally show *?thesis* **.**
qed

lemma *card-sequences*:
  assumes *finite A finite B card A* ≤ *card B*
  shows *card* {*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*} = *fact* (*card B*) *div fact* (*card B* − *card A*)
proof −
  obtain *enum* **where** *bij-betw enum* {*0..<card A*} *A*
    using ‹*finite A*› *ex-bij-betw-nat-finite* **by** *blast*
  have *bij-betw* (*function-of A enum*) {*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*} {*f* ∈ *A* $\to_E$ *B. inj-on f A*}
    using ‹*bij-betw enum* {*0..<card A*} *A*› **by** (*rule bij-betw-function-of*)
  from *this* **have** *card* {*xs. set xs* ⊆ *B* ∧ *length xs* = *card A* ∧ *distinct xs*} = *card* {*f* ∈ *A* $\to_E$ *B. inj-on f A*}
    by (*rule bij-betw-same-card*)
  also have *card* {*f* ∈ *A* $\to_E$ *B. inj-on f A*} = *fact* (*card B*) *div fact* (*card B* − *card A*)
    using ‹*finite A*› ‹*finite B*› ‹*card A* ≤ *card B*› **by** (*rule card-extensional-funcset-inj-on*)
  finally show *?thesis* **.**
qed

end

6 Functions from A to B, up to a Permutation of A

theory *Twelvefold-Way-Entry4*
imports *Equiv-Relations-on-Functions*
begin

6.1 Definition of Bijections

definition *msubset-of* :: $'a\ set \Rightarrow ('a \ \Rightarrow 'b)\ set \Rightarrow 'b\ multiset$
where
  msubset-of A F = *univ* (λf. *image-mset f* (*mset-set A*)) *F*

definition *functions-of* :: $'a\ set \Rightarrow 'b\ multiset \Rightarrow ('a \Rightarrow 'b)\ set$
where
  functions-of A B = $\{f \in A \rightarrow_E$ *set-mset B*. *image-mset f* (*mset-set A*) = $B\}$

6.2 Properties for Bijections

lemma *msubset-of*:
  assumes $F \in (A \rightarrow_E B)$ // *domain-permutation A B*
  shows *size* (*msubset-of A F*) = *card A*
  and *set-mset* (*msubset-of A F*) $\subseteq B$
proof −
  from ‹$F \in (A \rightarrow_E B)$ // *domain-permutation A B*› **obtain** *f* **where** $f \in A \rightarrow_E B$
    and *F-eq*: *F* = *domain-permutation A B* ‘‘ $\{f\}$ **using** *quotientE* **by** *blast*
  have *msubset-of A F* = *univ* (λf. *image-mset f* (*mset-set A*)) *F*
    unfolding *msubset-of-def* **..**
  also have ... = *univ* (λf. *image-mset f* (*mset-set A*)) (*domain-permutation A B* ‘‘ $\{f\}$)
    unfolding *F-eq* **..**
  also have ... = *image-mset f* (*mset-set A*)
    using *equiv-domain-permutation image-mset-respects-domain-permutation* ‹$f \in A \rightarrow_E B$›
    by (*subst univ-commute′*) *auto*
  finally have *msubset-of-eq*: *msubset-of A F* = *image-mset f* (*mset-set A*) **.**
  show *size* (*msubset-of A F*) = *card A*
  proof −
    have *size* (*msubset-of A F*) = *size* (*image-mset f* (*mset-set A*))
      unfolding *msubset-of-eq* **..**
    also have ... = *card A*
      by (*cases* ‹*finite A*›) *auto*
    finally show *?thesis* **.**
  qed
  show *set-mset* (*msubset-of A F*) $\subseteq B$
  proof −
    have *set-mset* (*msubset-of A F*) = *set-mset* (*image-mset f* (*mset-set A*))
      unfolding *msubset-of-eq* **..**

```
    also have ... ⊆ B
      using ‹f ∈ A →E B› by (cases finite A) auto
    finally show ?thesis .
  qed
qed

lemma functions-of:
  assumes finite A
  assumes set-mset M ⊆ B
  assumes size M = card A
  shows functions-of A M ∈ (A →E B) // domain-permutation A B
proof −
  obtain f where f ∈ A →E set-mset M and image-mset f (mset-set A) = M
    using obtain-function-on-ext-funcset ‹finite A› ‹size M = card A› by blast
  from ‹f ∈ A →E set-mset M› have f ∈ A →E B
    using ‹set-mset M ⊆ B› PiE-iff subset-eq by blast
  have functions-of A M = (domain-permutation A B) `` {f}
  proof
    show functions-of A M ⊆ domain-permutation A B `` {f}
    proof
      fix f'
      assume f' ∈ functions-of A M
      from this have M = image-mset f' (mset-set A) and f' ∈ A →E f' ` A
        using ‹finite A› unfolding functions-of-def by auto
      from this assms(1, 2) have f' ∈ A →E B
        by (simp add: PiE-iff image-subset-iff)
      obtain p where p permutes A ∧ (∀ x∈A. f x = f' (p x))
        using ‹finite A› ‹image-mset f (mset-set A) = M› ‹M = image-mset f'
(mset-set A)›
        image-mset-eq-implies-permutes by blast
      from this show f' ∈ domain-permutation A B `` {f}
        using ‹f ∈ A →E B› ‹f' ∈ A →E B›
        unfolding domain-permutation-def by auto
    qed
  next
    show domain-permutation A B `` {f} ⊆ functions-of A M
    proof
      fix f'
      assume f' ∈ domain-permutation A B `` {f}
      from this have (f, f') ∈ domain-permutation A B by auto
      from this ‹image-mset f (mset-set A) = M› have image-mset f' (mset-set A)
= M
        using congruentD[OF image-mset-respects-domain-permutation] by metis
      moreover from this ‹(f, f') ∈ domain-permutation A B› have f' ∈ A →E
set-mset M
        using ‹finite A› unfolding domain-permutation-def by auto
      ultimately show f' ∈ functions-of A M
        unfolding functions-of-def by auto
    qed
```

qed
from *this* ‹$f \in A \rightarrow_E B$› **show** *?thesis* **by** (*auto intro*: *quotientI*)
qed

lemma *functions-of-msubset-of*:
assumes *finite A*
assumes $F \in (A \rightarrow_E B)$ // *domain-permutation A B*
shows *functions-of A* (*msubset-of A F*) = *F*
proof −
from ‹$F \in (A \rightarrow_E B)$ // *domain-permutation A B*› **obtain** *f* **where** $f \in A \rightarrow_E B$
and *F-eq*: *F* = *domain-permutation A B* ‘‘ {*f*} **using** *quotientE* **by** *blast*
have *msubset-of A F* = *univ* (λ*f*. *image-mset f* (*mset-set A*)) *F*
unfolding *msubset-of-def* **..**
also have ... = *univ* (λ*f*. *image-mset f* (*mset-set A*)) (*domain-permutation A B* ‘‘ {*f*})
unfolding *F-eq* **..**
also have ... = *image-mset f* (*mset-set A*)
using *equiv-domain-permutation image-mset-respects-domain-permutation* ‹$f \in A \rightarrow_E B$›
by (*subst univ-commute′*) *auto*
finally have *msubset-of-eq*: *msubset-of A F* = *image-mset f* (*mset-set A*) **.**
show *?thesis*
proof
show *functions-of A* (*msubset-of A F*) ⊆ *F*
proof
fix *f′*
assume *f′* ∈ *functions-of A* (*msubset-of A F*)
from *this* **have** *f′*: $f' \in A \rightarrow_E f$ ‘ *set-mset* (*mset-set A*)
image-mset f′ (*mset-set A*) = *image-mset f* (*mset-set A*)
unfolding *functions-of-def* **by** (*auto simp add*: *msubset-of-eq*)
from ‹$f \in A \rightarrow_E B$› **have** *f* ‘ *A* ⊆ *B* **by** *auto*
note ‹$f \in A \rightarrow_E B$›
moreover from *f′*(*1*) ‹*finite A*› ‹*f* ‘ *A* ⊆ *B*› **have** $f' \in A \rightarrow_E B$ **by** *auto*
moreover obtain *p* **where** *p permutes A* ∧ (∀ *x*∈*A*. *f x* = *f′* (*p x*))
using ‹*finite A*› ‹*image-mset f′* (*mset-set A*) = *image-mset f* (*mset-set A*)›
by (*metis image-mset-eq-implies-permutes*)
ultimately show *f′* ∈ *F*
unfolding *F-eq domain-permutation-def* **by** *auto*
qed
next
show *F* ⊆ *functions-of A* (*msubset-of A F*)
proof
fix *f′*
assume *f′* ∈ *F*
from *this* **have** $f' \in A \rightarrow_E B$
unfolding *F-eq domain-permutation-def* **by** *auto*
from ‹*f′* ∈ *F*› **obtain** *p* **where** *p permutes A* ∧ (∀ *x*∈*A*. *f x* = *f′* (*p x*))
unfolding *F-eq domain-permutation-def* **by** *auto*

from *this* **have** *eq*: *image-mset f′* (*mset-set A*) = *image-mset f* (*mset-set A*)
using *permutes-implies-image-mset-eq* **by** *blast*
moreover have $f' \in A \rightarrow_E$ *set-mset* (*image-mset f* (*mset-set A*))
using ‹*finite A*› ‹$f' \in A \rightarrow_E B$› *eq*[*symmetric*] **by** *auto*
ultimately show $f' \in$ *functions-of A* (*msubset-of A F*)
unfolding *functions-of-def msubset-of-eq* **by** *auto*
qed
qed
qed

lemma *msubset-of-functions-of*:
assumes *set-mset M* ⊆ *B size M* = *card A finite A*
shows *msubset-of A* (*functions-of A M*) = *M*
proof −
from *assms* **have** *functions-of A M* ∈ ($A \rightarrow_E B$) // *domain-permutation A B*
using *functions-of* **by** *fastforce*
from *this* **obtain** *f* **where** $f \in A \rightarrow_E B$ **and** *functions-of A M* = *domain-permutation A B* ‘‘ {*f*}
by (*rule quotientE*)
from *this* **have** *f* ∈ *functions-of A M*
using *equiv-domain-permutation equiv-class-self* **by** *fastforce*
have *msubset-of A* (*functions-of A M*) = *univ* (λ*f*. *image-mset f* (*mset-set A*)) (*functions-of A M*)
unfolding *msubset-of-def* **..**
also have . . . = *univ* (λ*f*. *image-mset f* (*mset-set A*)) (*domain-permutation A B* ‘‘ {*f*})
unfolding ‹*functions-of A M* = *domain-permutation A B* ‘‘ {*f*}› **..**
also have . . . = *image-mset f* (*mset-set A*)
using *equiv-domain-permutation image-mset-respects-domain-permutation* ‹$f \in A \rightarrow_E B$›
by (*subst univ-commute′*) *auto*
also have *image-mset f* (*mset-set A*) = *M*
using ‹*f* ∈ *functions-of A M*› **unfolding** *functions-of-def* **by** *simp*
finally show *?thesis* **.**
qed

6.3 Bijections

lemma *bij-betw-msubset-of*:
assumes *finite A*
shows *bij-betw* (*msubset-of A*) (($A \rightarrow_E B$) // *domain-permutation A B*) {*M*. *set-mset M* ⊆ *B* ∧ *size M* = *card A*}
proof (*rule bij-betw-byWitness*[**where** *f′*=λ*M*. *functions-of A M*])
show ∀ *F*∈($A \rightarrow_E B$) // *domain-permutation A B*. *functions-of A* (*msubset-of A F*) = *F*
using ‹*finite A*› **by** (*auto simp add*: *functions-of-msubset-of*)
show ∀ *M*∈{*M*. *set-mset M* ⊆ *B* ∧ *size M* = *card A*}. *msubset-of A* (*functions-of A M*) = *M*
using ‹*finite A*› **by** (*auto simp add*: *msubset-of-functions-of*)

show *msubset-of A* ‘ $((A \rightarrow_E B)$ // *domain-permutation A B*) ⊆ {*M*. *set-mset M* ⊆ *B* ∧ *size M* = *card A*}
using *msubset-of* **by** *blast*
show *functions-of A* ‘ {*M*. *set-mset M* ⊆ *B* ∧ *size M* = *card A*} ⊆ $(A \rightarrow_E B)$ // *domain-permutation A B*
using *functions-of* ‹*finite A*› **by** *blast*
qed

6.4 Cardinality

lemma
assumes *finite A finite B*
shows *card* $((A \rightarrow_E B)$ // *domain-permutation A B*) = *card B* + *card A* − *1 choose card A*
proof −
have *bij-betw* (*msubset-of A*) $((A \rightarrow_E B)$ // *domain-permutation A B*) {*M*. *set-mset M* ⊆ *B* ∧ *size M* = *card A*}
using ‹*finite A*› **by** (*rule bij-betw-msubset-of*)
from *this* **have** *card* $((A \rightarrow_E B)$ // *domain-permutation A B*) = *card* {*M*. *set-mset M* ⊆ *B* ∧ *size M* = *card A*}
by (*rule bij-betw-same-card*)
also have *card* {*M*. *set-mset M* ⊆ *B* ∧ *size M* = *card A*} = *card B* + *card A* − *1 choose card A*
using ‹*finite B*› **by** (*rule card-multisets*)
finally show *?thesis* **.**
qed

end

7 Injections from A to B up to a Permutation of A

theory *Twelvefold-Way-Entry5*
imports
Equiv-Relations-on-Functions
begin

7.1 Definition of Bijections

definition *subset-of* :: *'a set* ⇒ (*'a* ⇒ *'b*) *set* ⇒ *'b set*
where
subset-of A F = *univ* (λ*f*. *f* ‘ *A*) *F*

definition *functions-of* :: *'a set* ⇒ *'b set* ⇒ (*'a* ⇒ *'b*) *set*
where
functions-of A B = {*f* ∈ $A \rightarrow_E B$. *f* ‘ *A* = *B*}

7.2 Properties for Bijections

```
lemma functions-of-eq:
  assumes finite A
  assumes f ∈ {f ∈ A →E B. inj-on f A}
  shows functions-of A (f ‘ A) = domain-permutation A B ‘‘ {f}
proof
  have bij: bij-betw f A (f ‘ A)
    using assms by (simp add: bij-betw-imageI)
  show functions-of A (f ‘ A) ⊆ domain-permutation A B ‘‘ {f}
  proof
    fix f′
    assume f′ ∈ functions-of A (f ‘ A)
    from this have f′ ∈ A →E f ‘ A and f′ ‘ A = f ‘ A
      unfolding functions-of-def by auto
    from this assms have f′ ∈ A →E B and inj-on f A
      using PiE-mem by fastforce+
    moreover have ∃ p. p permutes A ∧ (∀ x∈A. f x = f′ (p x))
    proof
      let ?p = λx. if x ∈ A then inv-into A f′ (f x) else x
      show ?p permutes A ∧ (∀ x∈A. f x = f′ (?p x))
      proof
        show ?p permutes A
        proof (rule bij-imp-permutes)
          show bij-betw ?p A A
          proof (rule bij-betw-imageI)
            show inj-on ?p A
            proof (rule inj-onI)
              fix a a′
              assume a ∈ A a′ ∈ A ?p a = ?p a′
              from this have inv-into A f′ (f a) = inv-into A f′ (f a′) by auto
              from this ‹a ∈ A› ‹a′ ∈ A› ‹f′ ‘ A = f ‘ A› have f a = f a′
                using inv-into-injective by fastforce
              from this ‹a ∈ A› ‹a′ ∈ A› show a = a′
                by (metis bij bij-betw-inv-into-left)
            qed
          next
            show ?p ‘ A = A
            proof
              show ?p ‘ A ⊆ A
                using ‹f′ ‘ A = f ‘ A› by (simp add: image-subsetI inv-into-into)
            next
              show A ⊆ ?p ‘ A
              proof
                fix a
                assume a ∈ A
                have inj-on f′ A
                  using ‹finite A› ‹f′ ‘ A = f ‘ A› ‹inj-on f A›
                  by (simp add: card-image eq-card-imp-inj-on)
                from ‹a ∈ A› ‹f′ ‘ A = f ‘ A› have inv-into A f (f′ a) ∈ A
```

by (*metis image-eqI inv-into-into*)
moreover have $a = inv\text{-}into\ A\ f'\ (f\ (inv\text{-}into\ A\ f\ (f'\ a)))$
using ‹$a \in A$› ‹f' ‘ $A = f$ ‘ A› ‹*inj-on* f' A›
by (*metis f-inv-into-f image-eqI inv-into-f-f*)
ultimately show $a \in$ *?p* ‘ A **by** *auto*
qed
qed
qed
next
fix x
assume $x \notin A$
from *this* **show** *?p* $x = x$ **by** *simp*
qed
next
from ‹f' ‘ $A = f$ ‘ A› **show** $\forall x \in A.\ f\ x = f'$ (*?p* x)
by (*simp add*: *f-inv-into-f*)
qed
qed
moreover have $f \in A \rightarrow_E B$ **using** *assms* **by** *auto*
ultimately show $f' \in$ *domain-permutation* A B ‘‘ $\{f\}$
unfolding *domain-permutation-def* **by** *auto*
qed
next
show *domain-permutation* A B ‘‘ $\{f\} \subseteq$ *functions-of* A (f ‘ A)
proof
fix f'
assume $f' \in$ *domain-permutation* A B ‘‘ $\{f\}$
from *this* **obtain** p **where** p: p *permutes* A $\forall x \in A.\ f\ x = f'\ (p\ x)$
and $f \in A \rightarrow_E B$ $f' \in A \rightarrow_E B$
unfolding *domain-permutation-def* **by** *auto*
have f' ‘ $A = f$ ‘ A
proof
show f' ‘ $A \subseteq f$ ‘ A
proof
fix x
assume $x \in f'$ ‘ A
from *this* **obtain** x' **where** $x = f'\ x'$ **and** $x' \in A$ **..**
from *this* **have** $x = f\ (inv\ p\ x')$
using p **by** (*metis* (*mono-tags*, *lifting*) *permutes-in-image permutes-inverses*(*1*))
moreover have *inv* p $x' \in A$
using p ‹$x' \in A$› **by** (*simp add*: *permutes-in-image permutes-inv*)
ultimately show $x \in f$ ‘ A **..**
qed
next
show f ‘ $A \subseteq f'$ ‘ A
using p *permutes-in-image* **by** *fastforce*
qed
moreover from *this* ‹$f' \in A \rightarrow_E B$› **have** $f' \in A \rightarrow_E f$ ‘ A **by** *auto*
ultimately show $f' \in$ *functions-of* A (f ‘ A)

unfolding *functions-of-def* **by** *auto*
qed
qed

lemma *subset-of*:
assumes $F \in \{f \in A \rightarrow_E B.\ inj\text{-}on\ f\ A\}\ //\ domain\text{-}permutation\ A\ B$
shows $subset\text{-}of\ A\ F \subseteq B$ **and** $card\ (subset\text{-}of\ A\ F) = card\ A$
proof −
from *assms* **obtain** *f* **where** $F\text{-}eq$: $F = (domain\text{-}permutation\ A\ B)\ ``\ \{f\}$
and *f*: $f \in A \rightarrow_E B\ inj\text{-}on\ f\ A$
using *mem-Collect-eq quotientE* **by** *force*
from *this* **have** $subset\text{-}of\ A\ (domain\text{-}permutation\ A\ B\ ``\ \{f\}) = f\ `\ A$
using *equiv-domain-permutation image-respects-domain-permutation*
unfolding *subset-of-def* **by** (*intro univ-commute′*) *auto*
from *this f F-eq* **show** $subset\text{-}of\ A\ F \subseteq B$ **and** $card\ (subset\text{-}of\ A\ F) = card\ A$
by (*auto simp add*: *card-image*)
qed

lemma *functions-of*:
assumes *finite A finite B* $X \subseteq B$ $card\ X = card\ A$
shows $functions\text{-}of\ A\ X \in \{f \in A \rightarrow_E B.\ inj\text{-}on\ f\ A\}\ //\ domain\text{-}permutation\ A$
B
proof −
from *assms* **obtain** *f* **where** *f*: $f \in A \rightarrow_E X \wedge bij\text{-}betw\ f\ A\ X$
using ‹*finite A*› ‹*finite B*› **by** (*metis finite-same-card-bij-on-ext-funcset finite-subset*)
from *this* **have** $X = f\ `\ A$ **by** (*simp add*: *bij-betw-def*)
from *f* ‹$X \subseteq B$› **have** $f \in \{f \in A \rightarrow_E B.\ inj\text{-}on\ f\ A\}$
by (*auto simp add*: *bij-betw-imp-inj-on*)
have $functions\text{-}of\ A\ X = domain\text{-}permutation\ A\ B\ ``\ \{f\}$
using ‹*finite A*› ‹$X = f\ `\ A$› ‹$f \in \{f \in A \rightarrow_E B.\ inj\text{-}on\ f\ A\}$›
by (*simp add*: *functions-of-eq*)
from *this* **show** $functions\text{-}of\ A\ X \in \{f \in A \rightarrow_E B.\ inj\text{-}on\ f\ A\}\ //\ domain\text{-}permutation$
$A\ B$
using ‹$f \in \{f \in A \rightarrow_E B.\ inj\text{-}on\ f\ A\}$› **by** (*auto intro*: *quotientI*)
qed

lemma *subset-of-functions-of*:
assumes *finite A finite X* $card\ A = card\ X$
shows $subset\text{-}of\ A\ (functions\text{-}of\ A\ X) = X$
proof −
from *assms* **obtain** *f* **where** $f \in A \rightarrow_E X$ **and** *bij-betw f A X*
using *finite-same-card-bij-on-ext-funcset* **by** *blast*
from *this* **have** *subset-of*: $subset\text{-}of\ A\ (domain\text{-}permutation\ A\ X\ ``\ \{f\}) = f\ `\ A$
using *equiv-domain-permutation image-respects-domain-permutation*
unfolding *subset-of-def* **by** (*intro univ-commute′*) *auto*
from ‹*bij-betw f A X*› **have** *inj-on f A* **and** $f\ `\ A = X$
by (*auto simp add*: *bij-betw-def*)
have $subset\text{-}of\ A\ (functions\text{-}of\ A\ X) = subset\text{-}of\ A\ (functions\text{-}of\ A\ (f\ `\ A))$

using ‹*f* ‘ *A* = *X*› **by** *simp*
also have ... = *subset-of A* (*domain-permutation A X* ‘‘ {*f*})
using ‹*finite A*› ‹*inj-on f A*› ‹$f \in A \to_E X$› **by** (*auto simp add*: *functions-of-eq*)
also have ... = *f* ‘ *A*
using ‹*inj-on f A*› ‹$f \in A \to_E X$› **by** (*simp add*: *subset-of*)
also have ... = *X*
using ‹*f* ‘ *A* = *X*› **by** *simp*
finally show *?thesis* **.**
qed

lemma *functions-of-subset-of*:
assumes *finite A*
assumes $F \in \{f \in A \to_E B.$ *inj-on f A*} // *domain-permutation A B*
shows *functions-of A* (*subset-of A F*) = *F*
using *assms*(*2*) **proof** (*rule quotientE*)
fix *f*
assume *f*: $f \in \{f \in A \to_E B.$ *inj-on f A*}
and *F-eq*: *F* = *domain-permutation A B* ‘‘ {*f*}
from *this* **have** *subset-of A* (*domain-permutation A B* ‘‘ {*f*}) = *f* ‘ *A*
using *equiv-domain-permutation image-respects-domain-permutation*
unfolding *subset-of-def* **by** (*intro univ-commute'*) *auto*
from *this f F-eq* ‹*finite A*› **show** *functions-of A* (*subset-of A F*) = *F*
by (*simp add*: *functions-of-eq*)
qed

7.3 Bijections

lemma *bij-betw-subset-of*:
assumes *finite A finite B*
shows *bij-betw* (*subset-of A*) ($\{f \in A \to_E B.$ *inj-on f A*} // *domain-permutation A B*) {*X*. $X \subseteq B \wedge$ *card X* = *card A*}
proof (*rule bij-betw-byWitness*[**where** *f'*=*functions-of A*])
show $\forall F \in \{f \in A \to_E B.$ *inj-on f A*} // *domain-permutation A B*. *functions-of A* (*subset-of A F*) = *F*
using ‹*finite A*› *functions-of-subset-of* **by** *auto*
show $\forall X \in$ {*X*. $X \subseteq B \wedge$ *card X* = *card A*}. *subset-of A* (*functions-of A X*) = *X*
using *subset-of-functions-of* ‹*finite A*› ‹*finite B*›
by (*metis* (*mono-tags*) *finite-subset mem-Collect-eq*)
show *subset-of A* ‘ ($\{f \in A \to_E B.$ *inj-on f A*} // *domain-permutation A B*) $\subseteq$ {*X*. $X \subseteq B \wedge$ *card X* = *card A*}
using *subset-of* **by** *fastforce*
show *functions-of A* ‘ {*X*. $X \subseteq B \wedge$ *card X* = *card A*} $\subseteq \{f \in A \to_E B.$ *inj-on f A*} // *domain-permutation A B*
using ‹*finite A*› ‹*finite B*› *functions-of* **by** *auto*
qed

lemma *bij-betw-functions-of*:
assumes *finite A finite B*
shows *bij-betw* (*functions-of A*) {*X*. $X \subseteq B \wedge$ *card X* = *card A*} ($\{f \in A \to_E$

B. inj-on f A} // *domain-permutation A B*)
proof (*rule bij-betw-byWitness*[**where** *f'=subset-of A*])
show ∀ *F*∈{*f* ∈ *A* →$_E$ *B. inj-on f A*} // *domain-permutation A B. functions-of A* (*subset-of A F*) = *F*
using ‹*finite A*› *functions-of-subset-of* **by** *auto*
show ∀ *X*∈{*X. X* ⊆ *B* ∧ *card X* = *card A*}. *subset-of A* (*functions-of A X*) = *X*
using *subset-of-functions-of* ‹*finite A*› ‹*finite B*›
by (*metis* (*mono-tags*) *finite-subset mem-Collect-eq*)
show *subset-of A* ‘ ({*f* ∈ *A* →$_E$ *B. inj-on f A*} // *domain-permutation A B*) ⊆ {*X. X* ⊆ *B* ∧ *card X* = *card A*}
using *subset-of* **by** *fastforce*
show *functions-of A* ‘ {*X. X* ⊆ *B* ∧ *card X* = *card A*} ⊆ {*f* ∈ *A* →$_E$ *B. inj-on f A*} // *domain-permutation A B*
using ‹*finite A*› ‹*finite B*› *functions-of* **by** *auto*
qed

lemma *bij-betw-mset-set*:
shows *bij-betw mset-set* {*A. finite A*} {*M.* ∀ *x. count M x* ≤ *1*}
proof (*rule bij-betw-byWitness*[**where** *f'=set-mset*])
show ∀ *A*∈{*A. finite A*}. *set-mset* (*mset-set A*) = *A* **by** *auto*
show ∀ *M*∈{*M.* ∀ *x. count M x* ≤ *1*}. *mset-set* (*set-mset M*) = *M*
by (*auto simp add*: *mset-set-set-mset'*)
show *mset-set* ‘ {*A. finite A*} ⊆ {*M.* ∀ *x. count M x* ≤ *1*}
using *nat-le-linear* **by** *fastforce*
show *set-mset* ‘ {*M.* ∀ *x. count M x* ≤ *1*} ⊆ {*A. finite A*} **by** *auto*
qed

lemma *bij-betw-mset-set-card*:
assumes *finite A*
shows *bij-betw mset-set* {*X. X* ⊆ *A* ∧ *card X* = *k*} {*M. M* ⊆# *mset-set A* ∧ *size M* = *k*}
proof (*rule bij-betw-byWitness*[**where** *f'=set-mset*])
show ∀ *X*∈{*X. X* ⊆ *A* ∧ *card X* = *k*}. *set-mset* (*mset-set X*) = *X*
using ‹*finite A*› *rev-finite-subset*[*of A*] **by** *auto*
show ∀ *M*∈{*M. M* ⊆# *mset-set A* ∧ *size M* = *k*}. *mset-set* (*set-mset M*) = *M*
by (*auto simp add*: *mset-set-set-mset*)
show *mset-set* ‘ {*X. X* ⊆ *A* ∧ *card X* = *k*} ⊆ {*M. M* ⊆# *mset-set A* ∧ *size M* = *k*}
using ‹*finite A*› *rev-finite-subset*[*of A*]
by (*auto simp add*: *mset-set-subseteq-mset-set*)
show *set-mset* ‘ {*M. M* ⊆# *mset-set A* ∧ *size M* = *k*} ⊆ {*X. X* ⊆ *A* ∧ *card X* = *k*}
using *assms mset-subset-eqD card-set-mset* **by** *fastforce*
qed

lemma *bij-betw-mset-set-card'*:
assumes *finite A*
shows *bij-betw mset-set* {*X. X* ⊆ *A* ∧ *card X* = *k*} {*M. set-mset M* ⊆ *A* ∧ *size M* = *k* ∧ (∀ *x. count M x* ≤ *1*)}

proof (*rule bij-betw-byWitness*[**where** *f′=set-mset*])
show ∀ *X*∈{*X*. *X* ⊆ *A* ∧ *card X* = *k*}. *set-mset* (*mset-set X*) = *X*
using ‹*finite A*› *rev-finite-subset*[*of A*] **by** *auto*
show ∀ *M*∈{*M*. *set-mset M* ⊆ *A* ∧ *size M* = *k* ∧ (∀ *x*. *count M x* ≤ *1*)}. *mset-set* (*set-mset M*) = *M*
by (*auto simp add*: *mset-set-set-mset′*)
show *mset-set* ‘ {*X*. *X* ⊆ *A* ∧ *card X* = *k*} ⊆ {*M*. *set-mset M* ⊆ *A* ∧ *size M* = *k* ∧ (∀ *x*. *count M x* ≤ *1*)}
using ‹*finite A*› *rev-finite-subset*[*of A*] **by** (*auto simp add*: *count-mset-set-leq′*)
show *set-mset* ‘ {*M*. *set-mset M* ⊆ *A* ∧ *size M* = *k* ∧ (∀ *x*. *count M x* ≤ *1*)} ⊆ {*X*. *X* ⊆ *A* ∧ *card X* = *k*}
by (*auto simp add*: *card-set-mset′*)
qed

7.4 Cardinality

lemma *card-injective-functions-domain-permutation*:
assumes *finite A finite B*
shows *card* ({*f* ∈ *A* $\rightarrow_E$ *B*. *inj-on f A*} // *domain-permutation A B*) = *card B choose card A*
proof −
have *bij-betw* (*subset-of A*) ({*f* ∈ *A* $\rightarrow_E$ *B*. *inj-on f A*} // *domain-permutation A B*) {*X*. *X* ⊆ *B* ∧ *card X* = *card A*}
using ‹*finite A*› ‹*finite B*› **by** (*rule bij-betw-subset-of*)
from *this* **have** *card* ({*f* ∈ *A* $\rightarrow_E$ *B*. *inj-on f A*} // *domain-permutation A B*) = *card* {*X*. *X* ⊆ *B* ∧ *card X* = *card A*}
by (*rule bij-betw-same-card*)
also have *card* {*X*. *X* ⊆ *B* ∧ *card X* = *card A*} = *card B choose card A*
using ‹*finite B*› **by** (*rule n-subsets*)
finally show *?thesis* **.**
qed

lemma *card-multiset-only-sets*:
assumes *finite A*
shows *card* {*M*. *M* ⊆# *mset-set A* ∧ *size M* = *k*} = *card A choose k*
proof −
have *bij-betw mset-set* {*X*. *X* ⊆ *A* ∧ *card X* = *k*} {*M*. *M* ⊆# *mset-set A* ∧ *size M* = *k*}
using ‹*finite A*› **by** (*rule bij-betw-mset-set-card*)
from *this* **have** *card* {*M*. *M* ⊆# *mset-set A* ∧ *size M* = *k*} = *card* {*X*. *X* ⊆ *A* ∧ *card X* = *k*}
by (*simp add*: *bij-betw-same-card*)
also have *card* {*X*. *X* ⊆ *A* ∧ *card X* = *k*} = *card A choose k*
using ‹*finite A*› **by** (*rule n-subsets*)
finally show *?thesis* **.**
qed

lemma *card-multiset-only-sets′*:
assumes *finite A*

shows *card* {*M*. *set-mset M* ⊆ *A* ∧ *size M* = *k* ∧ (∀ *x*. *count M x* ≤ *1*)} = *card A choose k*
proof −
from ‹*finite A*› **have** {*M*. *set-mset M* ⊆ *A* ∧ *size M* = *k* ∧ (∀ *x*. *count M x* ≤ *1*)} =
{*M*. *M* ⊆# *mset-set A* ∧ *size M* = *k*}
using *msubset-mset-set-iff* **by** *auto*
from *this* ‹*finite A*› *card-multiset-only-sets* **show** *?thesis* **by** *simp*
qed

end

8 Surjections from A to B up to a Permutation on A

theory *Twelvefold-Way-Entry6*
imports *Twelvefold-Way-Entry4*
begin

8.1 Properties for Bijections

lemma *set-mset-eq-implies-surj-on*:
assumes *finite A*
assumes *size M* = *card A set-mset M* = *B*
assumes *f* ∈ *functions-of A M*
shows *f* ‘ *A* = *B*
proof −
from ‹*f* ∈ *functions-of A M*› **have** *image-mset f* (*mset-set A*) = *M*
unfolding *functions-of-def* **by** *auto*
from ‹*image-mset f* (*mset-set A*) = *M*› **show** *f* ‘ *A* = *B*
using ‹*set-mset M* = *B*› ‹*finite A*› *finite-set-mset-mset-set set-image-mset* **by** *force*
qed

lemma *surj-on-implies-set-mset-eq*:
assumes *finite A*
assumes *F* ∈ (*A* $\rightarrow_E$ *B*) // *domain-permutation A B*
assumes *univ* (λ*f*. *f* ‘ *A* = *B*) *F*
shows *set-mset* (*msubset-of A F*) = *B*
proof −
from ‹*F* ∈ (*A* $\rightarrow_E$ *B*) // *domain-permutation A B*› **obtain** *f* **where** *f* ∈ *A* $\rightarrow_E$ *B*
and *F-eq*: *F* = *domain-permutation A B* ‘‘ {*f*} **using** *quotientE* **by** *blast*
have *msubset-of A F* = *univ* (λ*f*. *image-mset f* (*mset-set A*)) *F*
unfolding *msubset-of-def* **..**
also have . . . = *univ* (λ*f*. *image-mset f* (*mset-set A*)) (*domain-permutation A B* ‘‘ {*f*})
unfolding *F-eq* **..**

also have . . . = *image-mset f* (*mset-set A*)
using *equiv-domain-permutation image-mset-respects-domain-permutation* ‹$f \in A \to_E B$›
by (*subst univ-commute'*) *auto*
finally have *eq*: *msubset-of A F* = *image-mset f* (*mset-set A*) **.**
from *iffD1*[*OF univ-commute'*, *OF equiv-domain-permutation*, *OF surjective-respects-domain-permutation*, *OF* ‹$f \in A \to_E B$›]
‹*univ* (λ*f*. *f* ' *A* = *B*) *F*› **have** *f* ' *A* = *B* **by** (*simp add*: *F-eq*)
have *set-mset* (*image-mset f* (*mset-set A*)) = *B*
proof
show *set-mset* (*image-mset f* (*mset-set A*)) ⊆ *B*
using ‹*finite A*› ‹*f* ' *A* = *B*› **by** *auto*
next
show *B* ⊆ *set-mset* (*image-mset f* (*mset-set A*))
using ‹*finite A*› **by** (*simp add*: ‹*f* ' *A* = *B*›[*symmetric*] *in-image-mset*)
qed
from *this* **show** *set-mset* (*msubset-of A F*) = *B*
unfolding *eq* **.**
qed

lemma *functions-of-is-surj-on*:
assumes *finite A*
assumes *size M* = *card A set-mset M* = *B*
shows *univ* (λ*f*. *f* ' *A* = *B*) (*functions-of A M*)
proof −
have *functions-of A M* ∈ ($A \to_E B$) // *domain-permutation A B*
using *functions-of* ‹*finite A*› ‹*size M* = *card A*› ‹*set-mset M* = *B*› **by** *fastforce*
from *this* **obtain** *f* **where** *eq-f*: *functions-of A M* = *domain-permutation A B* '' {*f*} **and** $f \in A \to_E B$
using *quotientE* **by** *blast*
from *eq-f* **have** *f* ∈ *functions-of A M*
using ‹$f \in A \to_E B$› *equiv-domain-permutation equiv-class-self* **by** *fastforce*
have *f* ' *A* = *B*
using ‹*f* ∈ *functions-of A M*› *assms set-mset-eq-implies-surj-on* **by** *fastforce*
from *this* **show** *?thesis*
unfolding *eq-f* **using** *equiv-domain-permutation surjective-respects-domain-permutation* ‹$f \in A \to_E B$›
by (*subst univ-commute'*) *assumption+*
qed

8.2 Bijections

lemma *bij-betw-msubset-of*:
assumes *finite A*
shows *bij-betw* (*msubset-of A*) ({$f \in A \to_E B$. *f* ' *A* = *B*} // *domain-permutation A B*)
{*M*. *set-mset M* = *B* ∧ *size M* = *card A*}
(**is** *bij-betw - ?FSet ?MSet*)
proof (*rule bij-betw-byWitness*[**where** *f'*=λ*M*. *functions-of A M*])

have *quotient-eq*: *?FSet* = {*F* ∈ ((*A* $\to_E$ *B*) // *domain-permutation A B*). *univ* (λ*f*. *f* ‘ *A* = *B*) *F*}
using *equiv-domain-permutation*[*of A B*] *surjective-respects-domain-permutation*[*of A B*]
by (*simp only*: *univ-preserves-predicate*)
show ∀ *f*∈*?FSet*. *functions-of A* (*msubset-of A f*) = *f*
using ‹*finite A*› **by** (*auto simp only*: *quotient-eq functions-of-msubset-of*)
show ∀ *M*∈*?MSet*. *msubset-of A* (*functions-of A M*) = *M*
using ‹*finite A*› *msubset-of-functions-of* **by** *blast*
show *msubset-of A* ‘ *?FSet* ⊆ *?MSet*
using ‹*finite A*› **by** (*auto simp add*: *quotient-eq surj-on-implies-set-mset-eq msubset-of*)
show *functions-of A* ‘ *?MSet* ⊆ *?FSet*
using ‹*finite A*› **by** (*auto simp add*: *quotient-eq intro*: *functions-of functions-of-is-surj-on*)
qed

8.3 Cardinality

lemma *card-surjective-functions-domain-permutation*:
assumes *finite A finite B*
assumes *card B* ≤ *card A*
shows *card* ({*f* ∈ *A* $\to_E$ *B*. *f* ‘ *A* = *B*} // *domain-permutation A B*) = (*card A* − *1*) *choose* (*card A* − *card B*)
proof −
let *?FSet* = {*f* ∈ *A* $\to_E$ *B*. *f* ‘ *A* = *B*} // *domain-permutation A B*
and *?MSet* = {*M*. *set-mset M* = *B* ∧ *size M* = *card A*}
have *bij-betw* (*msubset-of A*) *?FSet ?MSet*
using ‹*finite A*› **by** (*rule bij-betw-msubset-of*)
from *this* **have** *card ?FSet* = *card ?MSet*
by (*rule bij-betw-same-card*)
also have *card ?MSet* = (*card A* − *1*) *choose* (*card A* − *card B*)
using ‹*finite B*› ‹*card B* ≤ *card A*› **by** (*rule card-multisets-covering-set*)
finally show *?thesis* **.**
qed

end

9 Functions from A to B up to a Permutation on B

theory *Twelvefold-Way-Entry7*
imports *Equiv-Relations-on-Functions*
begin

9.1 Definition of Bijections

definition *partitions-of* :: *'a set* ⇒ *'b set* ⇒ (*'a* ⇒ *'b*) *set* ⇒ *'a set set*
where

partitions-of A B F = *univ* (λf. (λb. $\{x \in A.\ f\ x = b\}$) ‘ B − $\{\{\}\}$) F

definition *functions-of* :: $'a$ *set set* ⇒ $'a$ *set* ⇒ $'b$ *set* ⇒ ($'a$ ⇒ $'b$) *set*
where
functions-of P A B = $\{f \in A \rightarrow_E B.$ (λb. $\{x \in A.\ f\ x = b\}$) ‘ B − $\{\{\}\}$ = $P\}$

9.2 Properties for Bijections

lemma *partitions-of*:
assumes *finite* B
assumes $F \in (A \rightarrow_E B)$ // *range-permutation* A B
shows *card* (*partitions-of* A B F) ≤ *card* B
and *partition-on* A (*partitions-of* A B F)
proof −
from ‹$F \in (A \rightarrow_E B)$ // *range-permutation* A B› **obtain** f **where** $f \in A \rightarrow_E B$
and *F-eq*: F = *range-permutation* A B ‘‘ $\{f\}$ **using** *quotientE* **by** *blast*
have *partitions-of* A B F = *univ* (λf. (λb. $\{x \in A.\ f\ x = b\}$) ‘ B − $\{\{\}\}$) F
unfolding *partitions-of-def* **..**
also have ... = *univ* (λf. (λb. $\{x \in A.\ f\ x = b\}$) ‘ B − $\{\{\}\}$) (*range-permutation* A B ‘‘ $\{f\}$)
unfolding *F-eq* **..**
also have ... = (λb. $\{x \in A.\ f\ x = b\}$) ‘ B − $\{\{\}\}$
using *equiv-range-permutation domain-partitions-respects-range-permutation* ‹$f \in A \rightarrow_E B$›
by (*subst univ-commute′*) *auto*
finally have *partitions-of-eq*: *partitions-of* A B F = (λb. $\{x \in A.\ f\ x = b\}$) ‘ B − $\{\{\}\}$ **.**
show *card* (*partitions-of* A B F) ≤ *card* B
proof −
have *card* (*partitions-of* A B F) = *card* ((λb. $\{x \in A.\ f\ x = b\}$) ‘ B − $\{\{\}\}$)
unfolding *partitions-of-eq* **..**
also have ... ≤ *card* ((λb. $\{x \in A.\ f\ x = b\}$) ‘ B)
using ‹*finite* B› **by** (*auto intro*: *card-mono*)
also have ... ≤ *card* B
using ‹*finite* B› **by** (*rule card-image-le*)
finally show *?thesis* **.**
qed
show *partition-on* A (*partitions-of* A B F)
proof −
have *partition-on* A ((λb. $\{x \in A.\ f\ x = b\}$) ‘ B − $\{\{\}\}$)
using ‹$f \in A \rightarrow_E B$› **by** (*auto intro*!: *partition-onI*)
from *this* **show** *?thesis*
unfolding *partitions-of-eq* **.**
qed
qed

lemma *functions-of*:
assumes *finite* A *finite* B

assumes *partition-on* A P
assumes *card* $P \leq$ *card* B
shows *functions-of* P A $B \in (A \rightarrow_E B)$ // *range-permutation* A B
proof −
obtain f **where** $f \in A \rightarrow_E B$ **and** *r1*: $(\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\} = P$
using *obtain-function-with-partition*[*OF* ‹*finite A*› ‹*finite B*› ‹*partition-on A P*› ‹*card P* ≤ *card B*›]
by *blast*
have *functions-of* P A B = *range-permutation* A B ‘‘ $\{f\}$
proof
show *functions-of* P A $B \subseteq$ *range-permutation* A B ‘‘ $\{f\}$
proof
fix f'
assume $f' \in$ *functions-of* P A B
from *this* **have** $f' \in A \rightarrow_E B$ **and** *r2*: $(\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\} = P$
unfolding *functions-of-def* **by** *auto*
from *r1 r2*
obtain p **where** p *permutes* $B \wedge (\forall x \in A.\ f\ x = p\ (f'\ x))$
using *partitions-eq-implies-permutes*[*OF* ‹$f \in A \rightarrow_E B$› ‹$f' \in A \rightarrow_E B$›] ‹*finite B*› **by** *metis*
from *this* **show** $f' \in$ *range-permutation* A B ‘‘ $\{f\}$
using ‹$f \in A \rightarrow_E B$› ‹$f' \in A \rightarrow_E B$›
unfolding *range-permutation-def* **by** *auto*
qed
next
show *range-permutation* A B ‘‘ $\{f\} \subseteq$ *functions-of* P A B
proof
fix f'
assume $f' \in$ *range-permutation* A B ‘‘ $\{f\}$
from *this* **have** $(f, f') \in$ *range-permutation* A B **by** *auto*
from *this* **have** $f' \in A \rightarrow_E B$
unfolding *range-permutation-def* **by** *auto*
from ‹$(f, f') \in$ *range-permutation A B*› **have**
$(\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\} = (\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\}$
using *congruentD*[*OF domain-partitions-respects-range-permutation*] **by** *blast*
from ‹$f' \in A \rightarrow_E B$› *this r1* **show** $f' \in$ *functions-of* P A B
unfolding *functions-of-def* **by** *auto*
qed
qed
from *this* ‹$f \in A \rightarrow_E B$› **show** *?thesis* **by** (*auto intro*: *quotientI*)
qed

lemma *functions-of-partitions-of*:
assumes *finite* B
assumes $F \in (A \rightarrow_E B)$ // *range-permutation* A B
shows *functions-of* (*partitions-of* A B F) A $B = F$
proof −
from ‹$F \in (A \rightarrow_E B)$ // *range-permutation A B*› **obtain** f **where** $f \in A \rightarrow_E$

B
  and *F-eq*: F = *range-permutation A B* ‘‘ $\{f\}$ **using** *quotientE* **by** *blast*
 have *partitions-of-eq*: *partitions-of A B F* = $(\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\}$
    unfolding *partitions-of-def F-eq*
    using *equiv-range-permutation domain-partitions-respects-range-permutation* ‹$f \in A \rightarrow_E B$›
    by (*subst univ-commute′*) *auto*
 show *?thesis*
 proof
  show *functions-of* (*partitions-of A B F*) *A B* ⊆ *F*
  proof
   fix f'
   assume f': $f' \in$ *functions-of* (*partitions-of A B F*) *A B*
   from *this* **have** $(\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\} = (\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\}$
    unfolding *functions-of-def* **by** (*auto simp add*: *partitions-of-eq*)
   note ‹$f \in A \rightarrow_E B$›
   moreover from f' **have** $f' \in A \rightarrow_E B$
    unfolding *functions-of-def* **by** *auto*
   moreover obtain *p* **where** *p permutes B* $\wedge$ $(\forall x \in A.\ f\ x = p\ (f'\ x))$
     using *partitions-eq-implies-permutes*[*OF* ‹$f \in A \rightarrow_E B$› ‹$f' \in A \rightarrow_E B$› ‹*finite B*›
     ‹$(\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\} = (\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\}$›]
    by *metis*
   ultimately show $f' \in F$
    unfolding *F-eq range-permutation-def* **by** *auto*
  qed
 next
  show $F \subseteq$ *functions-of* (*partitions-of A B F*) *A B*
  proof
   fix f'
   assume $f' \in F$
   from *this* **have** $f' \in A \rightarrow_E B$
    unfolding *F-eq range-permutation-def* **by** *auto*
   from ‹$f' \in F$› **obtain** *p* **where** *p permutes B* $\forall x \in A.\ f\ x = p\ (f'\ x)$
    unfolding *F-eq range-permutation-def* **by** *auto*
   have *eq*: $(\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\} = (\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\}$
   proof −
    have $(\lambda b.\ \{x \in A.\ f'\ x = b\})$ ‘ $B - \{\{\}\} = (\lambda b.\ \{x \in A.\ p\ (f'\ x) = b\})$ ‘ $B - \{\{\}\}$
      using *permutes-implies-inv-image-on-eq*[*OF* ‹*p permutes B*›, *of A f′*] **by** *simp*
    also have … = $(\lambda b.\ \{x \in A.\ f\ x = b\})$ ‘ $B - \{\{\}\}$
     using ‹$\forall x \in A.\ f\ x = p\ (f'\ x)$› **by** *auto*
    finally show *?thesis* **.**
   qed
   from *this* ‹$f' \in A \rightarrow_E B$› **show** $f' \in$ *functions-of* (*partitions-of A B F*) *A B*
    unfolding *functions-of-def partitions-of-eq* **by** *auto*

qed
qed
qed

lemma *partitions-of-functions-of*:
assumes *finite A finite B*
assumes *partition-on A P*
assumes *card P* $\leq$ *card B*
shows *partitions-of A B* (*functions-of P A B*) = *P*
proof −
have *functions-of P A B* $\in$ ($A \rightarrow_E B$) // *range-permutation A B*
using ‹*finite A*› ‹*finite B*› ‹*partition-on A P*› ‹*card P* $\leq$ *card B*› **by** (*rule functions-of*)
from *this* **obtain** *f* **where** *f* $\in A \rightarrow_E B$ **and** *functions-of-eq*: *functions-of P A B* = *range-permutation A B* ‘‘ {*f*}
using *quotientE* **by** *metis*
from *functions-of-eq* ‹*f* $\in A \rightarrow_E B$› **have** *f* $\in$ *functions-of P A B*
using *equiv-range-permutation equiv-class-self* **by** *fastforce*
have *partitions-of A B* (*functions-of P A B*) = *univ* (λf. (λb. {$x \in A$. $f\, x = b$}) ‘ *B* − {{}}) (*functions-of P A B*)
unfolding *partitions-of-def* ..
also have . . . = *univ* (λf. (λb. {$x \in A$. $f\, x = b$}) ‘ *B* − {{}}) (*range-permutation A B* ‘‘ {*f*})
unfolding ‹*functions-of P A B* = *range-permutation A B* ‘‘ {*f*}› ..
also have . . . = (λb. {$x \in A$. $f\, x = b$}) ‘ *B* − {{}}
using *equiv-range-permutation domain-partitions-respects-range-permutation* ‹*f* $\in A \rightarrow_E B$›
by (*subst univ-commute′*) *auto*
also have (λb. {$x \in A$. $f\, x = b$}) ‘ *B* − {{}} = *P*
using ‹*f* $\in$ *functions-of P A B*› **unfolding** *functions-of-def* **by** *simp*
finally show *?thesis* .
qed

9.3 Bijections

lemma *bij-betw-partitions-of*:
assumes *finite A finite B*
shows *bij-betw* (*partitions-of A B*) (($A \rightarrow_E B$) // *range-permutation A B*) {*P*. *partition-on A P* $\wedge$ *card P* $\leq$ *card B*}
proof (*rule bij-betw-byWitness*[**where** f'=λP. *functions-of P A B*])
show $\forall F \in (A \rightarrow_E B)$ // *range-permutation A B*. *functions-of* (*partitions-of A B F*) *A B* = *F*
using ‹*finite B*› **by** (*simp add*: *functions-of-partitions-of*)
show $\forall P \in$ {*P*. *partition-on A P* $\wedge$ *card P* $\leq$ *card B*}. *partitions-of A B* (*functions-of P A B*) = *P*
using ‹*finite A*› ‹*finite B*› **by** (*auto simp add*: *partitions-of-functions-of*)
show *partitions-of A B* ‘ (($A \rightarrow_E B$) // *range-permutation A B*) $\subseteq$ {*P*. *partition-on A P* $\wedge$ *card P* $\leq$ *card B*}
using ‹*finite B*› *partitions-of* **by** *auto*

show $(\lambda P.$ *functions-of* P A $B)$ ‘ $\{P.$ *partition-on* A $P \wedge$ *card* $P \leq$ *card* $B\} \subseteq$ $(A \rightarrow_E B)$ // *range-permutation* A B
using *functions-of* ‹*finite A*› ‹*finite B*› **by** *auto*
qed

9.4 Cardinality

lemma
assumes *finite A finite B*
shows *card* $((A \rightarrow_E B)$ // *range-permutation* A $B) = (\sum j \leq$ *card* B. *Stirling* (*card* A) $j)$
proof −
have *bij-betw* (*partitions-of* A B) $((A \rightarrow_E B)$ // *range-permutation* A $B)$ $\{P.$ *partition-on* A $P \wedge$ *card* $P \leq$ *card* $B\}$
using ‹*finite A*› ‹*finite B*› **by** (*rule bij-betw-partitions-of*)
from *this* **have** *card* $((A \rightarrow_E B)$ // *range-permutation* A $B)$ = *card* $\{P.$ *partition-on* A $P \wedge$ *card* $P \leq$ *card* $B\}$
by (*rule bij-betw-same-card*)
also have *card* $\{P.$ *partition-on* A $P \wedge$ *card* $P \leq$ *card* $B\} = (\sum j \leq$ *card* B. *Stirling* (*card* A) $j)$
using ‹*finite A*› **by** (*rule card-partition-on-at-most-size*)
finally show *?thesis* **.**
qed

end

10 Injections from A to B up to a Permutation on B

theory *Twelvefold-Way-Entry8*
imports *Twelvefold-Way-Entry7*
begin

10.1 Properties for Bijections

lemma *inj-on-implies-partitions-of*:
assumes $F \in (A \rightarrow_E B)$ // *range-permutation* A B
assumes *univ* $(\lambda f.$ *inj-on* f $A)$ F
shows $\forall X \in$ *partitions-of* A B F. *card* $X = 1$
proof −
from ‹$F \in (A \rightarrow_E B)$ // *range-permutation* A B› **obtain** f **where** $f \in A \rightarrow_E B$
and *F-eq*: F = *range-permutation* A B ‘‘ $\{f\}$ **using** *quotientE* **by** *blast*
from *this* ‹*univ* $(\lambda f.$ *inj-on* f $A)$ F› **have** *inj-on* f A
using *univ-commute'*[*OF equiv-range-permutation inj-on-respects-range-permutation* ‹$f \in A \rightarrow_E B$›] **by** *simp*
have $\forall X \in (\lambda b. \{x \in A. f\,x = b\})$ ‘ $B - \{\{\}\}$. *card* $X = 1$
proof
fix X

assume $X \in (\lambda b. \{x \in A. f\, x = b\})$ ‘ $B - \{\{\}\}$
from *this* **obtain** x **where** $X = \{xa \in A. f\, xa = f\, x\}$ $x \in A$ **by** *auto*
from *this* **have** $X = \{x\}$
using ‹*inj-on f A*› **by** (*auto dest*!: *inj-onD*)
from *this* **show** *card X = 1* **by** *simp*
qed
from *this* **show** *?thesis*
unfolding *partitions-of-def F-eq*
using *equiv-range-permutation domain-partitions-respects-range-permutation* ‹$f \in A \rightarrow_E B$›
by (*subst univ-commute'*) *assumption+*
qed

lemma *unique-part-eq-singleton*:
assumes *partition-on A P*
assumes $\forall X \in P.$ *card X = 1*
assumes $x \in A$
shows (*THE X*. $x \in X \wedge X \in P$) $= \{x\}$
proof −
have (*THE X*. $x \in X \wedge X \in P$) $\in P$
using ‹*partition-on A P*› ‹$x \in A$› **by** (*simp add*: *partition-on-the-part-mem*)
from *this* **have** *card* (*THE X*. $x \in X \wedge X \in P$) = *1*
using ‹$\forall X \in P.$ *card X = 1*› **by** *auto*
moreover have $x \in$ (*THE X*. $x \in X \wedge X \in P$)
using ‹*partition-on A P*› ‹$x \in A$› **by** (*simp add*: *partition-on-in-the-unique-part*)
ultimately show *?thesis*
by (*metis card-1-singletonE singleton-iff*)
qed

lemma *functions-of-is-inj-on*:
assumes *finite A finite B partition-on A P card P* $\leq$ *card B*
assumes $\forall X \in P.$ *card X = 1*
shows *univ* (λf. *inj-on f A*) (*functions-of P A B*)
proof −
have *functions-of P A B* $\in (A \rightarrow_E B)$ // *range-permutation A B*
using *functions-of* ‹*finite A*› ‹*finite B*› ‹*partition-on A P*› ‹*card P* $\leq$ *card B*› **by** *blast*
from *this* **obtain** f **where** *eq-f*: *functions-of P A B* = *range-permutation A B* ‘‘ $\{f\}$ **and** $f \in A \rightarrow_E B$
using *quotientE* **by** *blast*
from *eq-f* **have** $f \in$ *functions-of P A B*
using ‹$f \in A \rightarrow_E B$› *equiv-range-permutation equiv-class-self* **by** *fastforce*
from *this* **have** *eq*: $(\lambda b. \{x \in A. f\, x = b\})$ ‘ $B - \{\{\}\} = P$
unfolding *functions-of-def* **by** *auto*
have *inj-on f A*
proof (*rule inj-onI*)
fix $x\ y$
assume $x \in A$ $y \in A$ $f\, x = f\, y$
from ‹$x \in A$› **have** $x \in \{x' \in A. f\, x' = f\, x\}$ **by** *auto*

moreover from ‹$y \in A$› ‹$f\,x = f\,y$› **have** $y \in \{x' \in A.\ f\,x' = f\,x\}$ **by** *auto*
moreover have *card* $\{x' \in A.\ f\,x' = f\,x\} = 1$
proof −
from ‹$x \in A$› ‹$f \in A \rightarrow_E B$› **have** $f\,x \in B$ **by** *auto*
from *this* ‹$x \in A$› **have** $\{x' \in A.\ f\,x' = f\,x\} \in (\lambda b.\ \{x \in A.\ f\,x = b\})$ ‘ B − $\{\{\}\}$ **by** *auto*
from *this* ‹$\forall X \in P.$ *card* $X = 1$› *eq* **show** *?thesis* **by** *auto*
qed
ultimately show $x = y$ **by** (*metis card-1-singletonE singletonD*)
qed
from *this* **show** *?thesis*
unfolding *eq-f* **using** *equiv-range-permutation inj-on-respects-range-permutation* ‹$f \in A \rightarrow_E B$›
by (*subst univ-commute'*) *assumption+*
qed

10.2 Bijections

lemma *bij-betw-partitions-of*:
assumes *finite A finite B*
shows *bij-betw* (*partitions-of A B*) ($\{f \in A \rightarrow_E B.$ *inj-on f A*$\}$ // *range-permutation A B*) $\{P.$ *partition-on A P* $\wedge$ *card P* $\leq$ *card B* $\wedge$ ($\forall X \in P.$ *card X* $= 1$)$\}$
proof (*rule bij-betw-byWitness*[**where** $f' = \lambda P.$ *functions-of P A B*])
have *quotient-eq*: $\{f \in A \rightarrow_E B.$ *inj-on f A*$\}$ // *range-permutation A B* $= \{F \in ((A \rightarrow_E B)$ // *range-permutation A B*). *univ* ($\lambda f.$ *inj-on f A*) $F\}$
by (*simp add*: *equiv-range-permutation inj-on-respects-range-permutation univ-preserves-predicate*)
show $\forall F \in \{f \in A \rightarrow_E B.$ *inj-on f A*$\}$ // *range-permutation A B. functions-of* (*partitions-of A B F*) *A B* $= F$
using ‹*finite B*› **by** (*simp add*: *quotient-eq functions-of-partitions-of*)
show $\forall P \in \{P.$ *partition-on A P* $\wedge$ *card P* $\leq$ *card B* $\wedge$ ($\forall X \in P.$ *card X* $= 1$)$\}$. *partitions-of A B* (*functions-of P A B*) $= P$
using ‹*finite A*› ‹*finite B*› **by** (*simp add*: *partitions-of-functions-of*)
show *partitions-of A B* ‘ ($\{f \in A \rightarrow_E B.$ *inj-on f A*$\}$ // *range-permutation A B*) $\subseteq \{P.$ *partition-on A P* $\wedge$ *card P* $\leq$ *card B* $\wedge$ ($\forall X \in P.$ *card X* $= 1$)$\}$
using ‹*finite B*› *quotient-eq partitions-of inj-on-implies-partitions-of* **by** *fastforce*
show ($\lambda P.$ *functions-of P A B*) ‘ $\{P.$ *partition-on A P* $\wedge$ *card P* $\leq$ *card B* $\wedge$ ($\forall X \in P.$ *card X* $= 1$)$\} \subseteq \{f \in A \rightarrow_E B.$ *inj-on f A*$\}$ // *range-permutation A B*
using ‹*finite A*› ‹*finite B*› **by** (*auto simp add*: *quotient-eq intro*: *functions-of functions-of-is-inj-on*)
qed

10.3 Cardinality

lemma *card-injective-functions-range-permutation*:
assumes *finite A finite B*
shows *card* ($\{f \in A \rightarrow_E B.$ *inj-on f A*$\}$ // *range-permutation A B*) $=$ *iverson* (*card A* $\leq$ *card B*)
proof −
obtain *enum* **where** *bij-betw enum* $\{0..<$*card A*$\}$ *A*
using ‹*finite A*› *ex-bij-betw-nat-finite* **by** *blast*

have *bij-betw* (*partitions-of A B*) ({*f* ∈ *A* $\to_E$ *B*. *inj-on f A*} // *range-permutation A B*) {*P*. *partition-on A P* ∧ *card P* ≤ *card B* ∧ (∀ *X*∈*P*. *card X* = *1*)}
  using ‹*finite A*› ‹*finite B*› **by** (*rule bij-betw-partitions-of*)
from *this* **have** *card* ({*f* ∈ *A* $\to_E$ *B*. *inj-on f A*} // *range-permutation A B*) = *card* {*P*. *partition-on A P* ∧ *card P* ≤ *card B* ∧ (∀ *X*∈*P*. *card X* = *1*)}
  by (*rule bij-betw-same-card*)
also have *card* {*P*. *partition-on A P* ∧ *card P* ≤ *card B* ∧ (∀ *X*∈*P*. *card X* = *1*)} = *iverson* (*card A* ≤ *card B*)
  using ‹*finite A*› **by** (*rule card-partition-on-size1-eq-iverson*)
finally show *?thesis* **.**
qed

end

11 Surjections from A to B up to a Permutation on B

theory *Twelvefold-Way-Entry9*
imports *Twelvefold-Way-Entry7*
begin

11.1 Properties for Bijections

lemma *surjective-on-implies-card-eq*:
  assumes *f* ‘ *A* = *B*
  shows *card* ((λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}}) = *card B*
proof −
  from ‹*f* ‘ *A* = *B*› **have** {} ∉ (λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* **by** *auto*
  from ‹*f* ‘ *A* = *B*› **have** *inj-on* (λ*b*. {*x* ∈ *A*. *f x* = *b*}) *B* **by** (*fastforce intro*: *inj-onI*)
  have *card* ((λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}}) = *card* ((λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B*)
    using ‹{} ∉ (λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B*› **by** *simp*
  also have ... = *card B*
    using ‹*inj-on* (λ*b*. {*x* ∈ *A*. *f x* = *b*}) *B*› **by** (*rule card-image*)
  finally show *?thesis* **.**
qed

lemma *card-eq-implies-surjective-on*:
  assumes *finite B f* ∈ *A* $\to_E$ *B*
  assumes *card-eq*: *card* ((λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}}) = *card B*
  shows *f* ‘ *A* = *B*
proof
  from ‹*f* ∈ *A* $\to_E$ *B*› **show** *f* ‘ *A* ⊆ *B* **by** *auto*
next
  show *B* ⊆ *f* ‘ *A*
  proof
    fix *x*

```
    assume x ∈ B
    have {} ∉ (λb. {x ∈ A. f x = b}) ‘ B
    proof (cases card B ≥ 1)
      assume ¬ card B ≥ 1
      from this have card B = 0 by simp
      from this ‹finite B› have B = {} by simp
      from this show ?thesis by simp
    next
      assume card B ≥ 1
      show ?thesis
      proof (rule ccontr)
        assume ¬ {} ∉ (λb. {x ∈ A. f x = b}) ‘ B
        from this have {} ∈ (λb. {x ∈ A. f x = b}) ‘ B by simp
        moreover have card ((λb. {x ∈ A. f x = b}) ‘ B) ≤ card B
          using ‹finite B› card-image-le by blast
        moreover have finite ((λb. {x ∈ A. f x = b}) ‘ B)
          using ‹finite B› by auto
        ultimately have card ((λb. {x ∈ A. f x = b}) ‘ B − {{}}) ≤ card B − 1
          by (auto simp add: card-Diff-singleton)
        from this card-eq ‹card B ≥ 1› show False by auto
      qed
    qed
    from this ‹x ∈ B› show x ∈ f ‘ A by force
  qed
qed

lemma card-partitions-of:
  assumes F ∈ (A →E B) // range-permutation A B
  assumes univ (λf. f ‘ A = B) F
  shows card (partitions-of A B F) = card B
proof −
  from ‹F ∈ (A →E B) // range-permutation A B› obtain f where f ∈ A →E
B
    and F-eq: F = range-permutation A B ‘‘ {f} using quotientE by blast
  from this ‹univ (λf. f ‘ A = B) F› have f ‘ A = B
    using univ-commute′[OF equiv-range-permutation surj-on-respects-range-permutation
‹f ∈ A →E B›] by simp
  have card (partitions-of A B F) = card (univ (λf. (λb. {x ∈ A. f x = b}) ‘ B −
{{}}) F)
    unfolding partitions-of-def ..
  also have . . . = card (univ (λf. (λb. {x ∈ A. f x = b}) ‘ B − {{}}) (range-permutation
A B ‘‘ {f}))
    unfolding F-eq ..
  also have . . . = card ((λb. {x ∈ A. f x = b}) ‘ B − {{}})
    using equiv-range-permutation domain-partitions-respects-range-permutation ‹f
∈ A →E B›
    by (subst univ-commute′) auto
  also from ‹f ‘ A = B› have . . . = card B
    using surjective-on-implies-card-eq by auto
```

finally show *?thesis* **.**
qed

lemma *functions-of-is-surj-on*:
assumes *finite A finite B*
assumes *partition-on A P card P = card B*
shows *univ* (λ*f*. *f* ‘ *A* = *B*) (*functions-of P A B*)
proof −
have *functions-of P A B* ∈ (*A* →*E* *B*) // *range-permutation A B*
using *functions-of* ‹*finite A*› ‹*finite B*› ‹*partition-on A P*› ‹*card P = card B*›
by *fastforce*
from *this* **obtain** *f* **where** *eq-f*: *functions-of P A B* = *range-permutation A B* ‘‘ {*f*} **and** *f* ∈ *A* →*E* *B*
using *quotientE* **by** *blast*
from *eq-f* **have** *f* ∈ *functions-of P A B*
using ‹*f* ∈ *A* →*E* *B*› *equiv-range-permutation equiv-class-self* **by** *fastforce*
from ‹*f* ∈ *functions-of P A B*› **have** *eq*: (λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}} = *P*
unfolding *functions-of-def* **by** *auto*
from *this* **have** *card* ((λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}}) = *card B*
using ‹*card P = card B*› **by** *simp*
from ‹*finite B*› ‹*f* ∈ *A* →*E* *B*› *this* **have** *f* ‘ *A* = *B*
using *card-eq-implies-surjective-on* **by** *blast*
from *this* **show** *?thesis*
unfolding *eq-f* **using** *equiv-range-permutation surj-on-respects-range-permutation* ‹*f* ∈ *A* →*E* *B*›
by (*subst univ-commute′*) *assumption+*
qed

11.2 Bijections

lemma *bij-betw-partitions-of*:
assumes *finite A finite B*
shows *bij-betw* (*partitions-of A B*) ({*f* ∈ *A* →*E* *B*. *f* ‘ *A* = *B*} // *range-permutation A B*) {*P*. *partition-on A P* ∧ *card P* = *card B*}
proof (*rule bij-betw-byWitness*[**where** *f′*=λ*P*. *functions-of P A B*])
have *quotient-eq*: {*f* ∈ *A* →*E* *B*. *f* ‘ *A* = *B*} // *range-permutation A B* = {*F* ∈ ((*A* →*E* *B*) // *range-permutation A B*). *univ* (λ*f*. *f* ‘ *A* = *B*) *F*}
using *equiv-range-permutation*[*of A B*] *surj-on-respects-range-permutation*[*of A B*] **by** (*simp only*: *univ-preserves-predicate*)
show ∀ *F*∈{*f* ∈ *A* →*E* *B*. *f* ‘ *A* = *B*} // *range-permutation A B*. *functions-of* (*partitions-of A B F*) *A B* = *F*
using ‹*finite B*› **by** (*simp add*: *functions-of-partitions-of quotient-eq*)
show ∀ *P*∈{*P*. *partition-on A P* ∧ *card P* = *card B*}. *partitions-of A B* (*functions-of P A B*) = *P*
using ‹*finite A*› ‹*finite B*› **by** (*auto simp add*: *partitions-of-functions-of*)
show *partitions-of A B* ‘ ({*f* ∈ *A* →*E* *B*. *f* ‘ *A* = *B*} // *range-permutation A B*) ⊆ {*P*. *partition-on A P* ∧ *card P* = *card B*}
using ‹*finite B*› *quotient-eq card-partitions-of partitions-of* **by** *fastforce*
show (λ*P*. *functions-of P A B*) ‘ {*P*. *partition-on A P* ∧ *card P* = *card B*} ⊆

$\{f \in A \rightarrow_E B.\ f\ `\ A = B\}$ // *range-permutation A B*
using ‹*finite A*› ‹*finite B*› **by** (*auto simp add*: *quotient-eq intro*: *functions-of functions-of-is-surj-on*)
qed

11.3 Cardinality

lemma *card-surjective-functions-range-permutation*:
assumes *finite A finite B*
shows *card* ($\{f \in A \rightarrow_E B.\ f\ `\ A = B\}$ // *range-permutation A B*) = *Stirling* (*card A*) (*card B*)
proof −
have *bij-betw* (*partitions-of A B*) ($\{f \in A \rightarrow_E B.\ f\ `\ A = B\}$ // *range-permutation A B*) $\{P.$ *partition-on A P* $\wedge$ *card P* = *card B*$\}$
using ‹*finite A*› ‹*finite B*› **by** (*rule bij-betw-partitions-of*)
from *this* **have** *card* ($\{f \in A \rightarrow_E B.\ f\ `\ A = B\}$ // *range-permutation A B*) = *card* $\{P.$ *partition-on A P* $\wedge$ *card P* = *card B*$\}$
by (*rule bij-betw-same-card*)
also have *card* $\{P.$ *partition-on A P* $\wedge$ *card P* = *card B*$\}$ = *Stirling* (*card A*) (*card B*)
using ‹*finite A*› **by** (*rule card-partition-on*)
finally show *?thesis* **.**
qed

end

12 Surjections from A to B

theory *Twelvefold-Way-Entry3*
imports
Twelvefold-Way-Entry9
begin

lemma *card-of-equiv-class*:
assumes *finite B*
assumes $F \in \{f \in A \rightarrow_E B.\ f\ `\ A = B\}$ // *range-permutation A B*
shows *card F* = *fact* (*card B*)
proof −
from ‹$F \in \{f \in A \rightarrow_E B.\ f\ `\ A = B\}$ // *range-permutation A B*› **obtain** *f* **where**
$f \in A \rightarrow_E B$ **and** $f\ `\ A = B$
and *F-eq*: *F* = *range-permutation A B* `` $\{f\}$ **using** *quotientE* **by** *blast*
have *set-eq*: *range-permutation A B* `` $\{f\}$ = ($\lambda p\ x.$ *if* $x \in A$ *then* $p\ (f\ x)$ *else undefined*) ` $\{p.\ p$ *permutes B*$\}$
proof
show *range-permutation A B* `` $\{f\} \subseteq$ ($\lambda p\ x.$ *if* $x \in A$ *then* $p\ (f\ x)$ *else undefined*) ` $\{p.\ p$ *permutes B*$\}$
proof
fix f'

assume $f' \in$ *range-permutation* A B `` $\{f\}$
from *this* **obtain** p **where** p *permutes* B $\forall x{\in}A.\ f\ x = p\ (f'\ x)$
unfolding *range-permutation-def* **by** *auto*
from ‹$f' \in$ *range-permutation* A B `` $\{f\}$› **have** $f' \in A \to_E B$
unfolding *range-permutation-def* **by** *auto*
have $f' = (\lambda x.\ \mathit{if}\ x \in A\ \mathit{then}\ \mathit{inv}\ p\ (f\ x)\ \mathit{else}\ \mathit{undefined})$
proof
fix x
show $f'\ x = (\mathit{if}\ x \in A\ \mathit{then}\ \mathit{inv}\ p\ (f\ x)\ \mathit{else}\ \mathit{undefined})$
using ‹$f \in A \to_E B$› ‹$f' \in A \to_E B$› ‹$\forall x{\in}A.\ f\ x = p\ (f'\ x)$›
‹p *permutes* B› *permutes-inverses*(2) **by** *fastforce*
qed
moreover have *inv p permutes B* **using** ‹*p permutes B*› **by** (*simp add*: *permutes-inv*)
ultimately show $f' \in (\lambda p.\ (\lambda x.\ \mathit{if}\ x \in A\ \mathit{then}\ p\ (f\ x)\ \mathit{else}\ \mathit{undefined}))$ ` $\{p.\ p$ *permutes* $B\}$
by *auto*
qed
next
show $(\lambda p\ x.\ \mathit{if}\ x \in A\ \mathit{then}\ p\ (f\ x)\ \mathit{else}\ \mathit{undefined})$ ` $\{p.\ p$ *permutes* $B\} \subseteq$ *range-permutation* A B `` $\{f\}$
proof
fix f'
assume $f' \in (\lambda p\ x.\ \mathit{if}\ x \in A\ \mathit{then}\ p\ (f\ x)\ \mathit{else}\ \mathit{undefined})$ ` $\{p.\ p$ *permutes* $B\}$
from *this* **obtain** p **where** p *permutes* B **and** f'-*eq*: $f' = (\lambda x.\ \mathit{if}\ x \in A\ \mathit{then}\ p\ (f\ x)\ \mathit{else}\ \mathit{undefined})$ **by** *auto*
from *this* **have** $f' \in A \to_E B$
using ‹$f \in A \to_E B$› *permutes-in-image* **by** *fastforce*
moreover have *inv p permutes B* **using** ‹*p permutes B*› **by** (*simp add*: *permutes-inv*)
moreover have $\forall x{\in}A.\ f\ x = \mathit{inv}\ p\ (f'\ x)$
using ‹$f \in A \to_E B$› ‹$f' \in A \to_E B$› f'-*eq*
‹p *permutes* B› *permutes-inverses*(2) **by** *fastforce*
ultimately show $f' \in$ *range-permutation* A B `` $\{f\}$
using ‹$f \in A \to_E B$› **unfolding** *range-permutation-def* **by** *auto*
qed
qed
have *inj-on* $(\lambda p\ x.\ \mathit{if}\ x \in A\ \mathit{then}\ p\ (f\ x)\ \mathit{else}\ \mathit{undefined})$ $\{p.\ p$ *permutes* $B\}$
proof (*rule inj-onI*)
fix p p'
assume $p \in \{p.\ p$ *permutes* $B\}$ $p' \in \{p.\ p$ *permutes* $B\}$
and *eq*: $(\lambda x.\ \mathit{if}\ x \in A\ \mathit{then}\ p\ (f\ x)\ \mathit{else}\ \mathit{undefined}) = (\lambda x.\ \mathit{if}\ x \in A\ \mathit{then}\ p'\ (f\ x)\ \mathit{else}\ \mathit{undefined})$
{
fix x
have $p\ x = p'\ x$
proof *cases*
assume $x \in B$
from *this* **obtain** y **where** $y \in A$ **and** $x = f\ y$

```
      using ‹f ‘ A = B› by blast
    from eq this have p (f y) = p′ (f y) by meson
    from this ‹x = f y› show p x = p′ x by simp
  next
    assume x ∉ B
    from this show p x = p′ x
      using ‹p ∈ {p. p permutes B}› ‹p′ ∈ {p. p permutes B}›
      by (simp add: permutes-def)
  qed
 }
 from this show p = p′ by auto
qed
have card F = card ((λp x. if x ∈ A then p (f x) else undefined) ‘ {p. p permutes
B})
  unfolding F-eq set-eq ..
also have ... = card {p. p permutes B}
  using ‹inj-on (λp x. if x ∈ A then p (f x) else undefined) {p. p permutes B}›
  by (simp add: card-image)
also have ... = fact (card B)
  using ‹finite B› by (simp add: card-permutations)
finally show ?thesis .
qed

lemma card-extensional-funcset-surj-on:
 assumes finite A finite B
 shows card {f ∈ A →E B. f ‘ A = B} = fact (card B) ∗ Stirling (card A) (card
B) (is card ?F = -)
proof −
 have card ?F = fact (card B) ∗ card (?F // range-permutation A B)
  using ‹finite B›
  by (simp only: card-equiv-class-restricted-same-size[OF equiv-range-permutation
surj-on-respects-range-permutation card-of-equiv-class])
 also have ... = fact (card B) ∗ Stirling (card A) (card B)
  using ‹finite A› ‹finite B›
  by (simp only: card-surjective-functions-range-permutation)
 finally show ?thesis .
qed

end
```

13 Functions from A to B up to a Permutation on A and B

```
theory Twelvefold-Way-Entry10
imports Equiv-Relations-on-Functions
begin
```

13.1 Definition of Bijections

definition *number-partition-of* :: $'a\ set \Rightarrow 'b\ set \Rightarrow ('a \Rightarrow 'b)\ set \Rightarrow nat\ multiset$
where
number-partition-of $A\ B\ F = univ\ (\lambda f.\ image\text{-}mset\ (\lambda X.\ card\ X)\ (mset\text{-}set\ ((\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\})))\ F$

definition *functions-of* :: $'a\ set \Rightarrow 'b\ set \Rightarrow nat\ multiset \Rightarrow ('a \Rightarrow 'b)\ set$
where
functions-of $A\ B\ N = \{f \in A \rightarrow_E B.\ image\text{-}mset\ (\lambda X.\ card\ X)\ (mset\text{-}set\ ((\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\})) = N\}$

13.2 Properties for Bijections

lemma *card-setsum-partition*:
assumes *finite* A *finite* B $f \in A \rightarrow_E B$
shows *sum card* $((\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\}) = card\ A$
proof −
have *finite* $((\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\})$
using ‹*finite B*› **by** *blast*
moreover have $\forall X \in (\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\}.\ finite\ X$
using ‹*finite A*› **by** *auto*
moreover have $\bigcup((\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\}) = A$
using ‹$f \in A \rightarrow_E B$› **by** *auto*
ultimately show *?thesis*
by (*subst card-Union-disjoint*[*symmetric*]) (*auto simp*: *pairwise-def disjnt-def*)
qed

lemma *number-partition-of*:
assumes *finite* A *finite* B
assumes $F \in (A \rightarrow_E B)\ //\ domain\text{-}and\text{-}range\text{-}permutation\ A\ B$
shows *number-partition* (*card A*) (*number-partition-of A B F*)
and *size* (*number-partition-of A B F*) $\leq$ *card B*
proof −
from ‹$F \in (A \rightarrow_E B)\ //\ domain\text{-}and\text{-}range\text{-}permutation\ A\ B$› **obtain** f **where** $f \in A \rightarrow_E B$
and *F-eq*: $F = domain\text{-}and\text{-}range\text{-}permutation\ A\ B\ ``\ \{f\}$ **using** *quotientE* **by** *blast*
have *number-partition-of-eq*: *number-partition-of* $A\ B\ F = image\text{-}mset\ card\ (mset\text{-}set\ ((\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\}))$
proof −
have *number-partition-of* $A\ B\ F = univ\ (\lambda f.\ image\text{-}mset\ card\ (mset\text{-}set\ ((\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\})))\ F$
unfolding *number-partition-of-def* **..**
also have $\ldots = univ\ (\lambda f.\ image\text{-}mset\ card\ (mset\text{-}set\ ((\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\})))\ (domain\text{-}and\text{-}range\text{-}permutation\ A\ B\ ``\ \{f\})$
unfolding *F-eq* **..**
also have $\ldots = image\text{-}mset\ card\ (mset\text{-}set\ ((\lambda b.\ \{x \in A.\ f\ x = b\})\ `\ B - \{\{\}\}))$
using ‹*finite B*› *equiv-domain-and-range-permutation multiset-of-partition-cards-respects-domain-and-rang*

‹$f \in A \rightarrow_E B$›
by (*subst univ-commute*′) *auto*
finally show *?thesis* **.**
qed
show *number-partition* (*card A*) (*number-partition-of A B F*)
proof −
have *sum-mset* (*number-partition-of A B F*) = *card A*
using *number-partition-of-eq* ‹*finite A*› ‹*finite B*› ‹$f \in A \rightarrow_E B$›
by (*simp only*: *sum-unfold-sum-mset*[*symmetric*] *card-setsum-partition*)
moreover have *0* ∉# *number-partition-of A B F*
proof −
have ∀ *X* ∈ (λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B*. *finite X*
using ‹*finite A*› **by** *simp*
from *this* **have** ∀ *X* ∈ (λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}}. *card X* ≠ *0* **by** *auto*
from *this* **show** *?thesis*
using *number-partition-of-eq* ‹*finite B*› **by** (*simp add*: *image-iff*)
qed
ultimately show *?thesis* **unfolding** *number-partition-def* **by** *simp*
qed
show *size* (*number-partition-of A B F*) ≤ *card B*
using *number-partition-of-eq* ‹*finite A*› ‹*finite B*›
by (*metis* (*no-types*, *lifting*) *card-Diff1-le card-image-le finite-imageI le-trans size-image-mset size-mset-set*)
qed

lemma *functions-of*:
assumes *finite A finite B*
assumes *number-partition* (*card A*) *N*
assumes *size N* ≤ *card B*
shows *functions-of A B N* ∈ ($A \rightarrow_E B$) // *domain-and-range-permutation A B*
proof −
obtain *f* **where** $f \in A \rightarrow_E B$ **and** *eq-N*: *image-mset* (λ*X*. *card X*) (*mset-set* (((λ*b*. {*x* ∈ *A*. *f x* = *b*})) ‘ *B* − {{}})) = *N*
using *obtain-extensional-function-from-number-partition* ‹*finite A*› ‹*finite B*› ‹*number-partition* (*card A*) *N*› ‹*size N* ≤ *card B*› **by** *blast*
have *functions-of A B N* = (*domain-and-range-permutation A B*) ‘‘ {*f*}
proof
show *functions-of A B N* ⊆ *domain-and-range-permutation A B* ‘‘ {*f*}
proof
fix *f*′
assume *f*′ ∈ *functions-of A B N*
from *this* **have** *eq-N*′: *N* = *image-mset* (λ*X*. *card X*) (*mset-set* (((λ*b*. {*x* ∈ *A*. *f*′ *x* = *b*})) ‘ *B* − {{}}))
and $f' \in A \rightarrow_E B$
unfolding *functions-of-def* **by** *auto*
from ‹*finite A*› ‹*finite B*› ‹$f \in A \rightarrow_E B$› ‹$f' \in A \rightarrow_E B$›
obtain p_A p_B **where** p_A *permutes A* p_B *permutes B* ∀ *x*∈*A*. *f x* = p_B (*f*′ (p_A *x*))

using *eq-N eq-N′ multiset-of-partition-cards-eq-implies-permutes*[*of A B f f′*]
by *blast*
from *this* **show** $f' \in$ *domain-and-range-permutation A B* `` $\{f\}$
using ‹$f \in A \rightarrow_E B$› ‹$f' \in A \rightarrow_E B$›
unfolding *domain-and-range-permutation-def* **by** *auto*
qed
next
show *domain-and-range-permutation A B* `` $\{f\} \subseteq$ *functions-of A B N*
proof
fix f'
assume $f' \in$ *domain-and-range-permutation A B* `` $\{f\}$
from *this* **have** *in-equiv-relation*: $(f, f') \in$ *domain-and-range-permutation A B* **by** *auto*
from *eq-N* ‹*finite B*› **have** *image-mset* (λX. *card X*) (*mset-set* (((λb. $\{x \in A.\ f'\ x = b\}$)) ` $B - \{\{\}\}$)) = N
using *congruentD*[*OF multiset-of-partition-cards-respects-domain-and-range-permutation in-equiv-relation*]
by *metis*
moreover from ‹$(f, f') \in$ *domain-and-range-permutation A B*› **have** $f' \in A \rightarrow_E B$
unfolding *domain-and-range-permutation-def* **by** *auto*
ultimately show $f' \in$ *functions-of A B N*
unfolding *functions-of-def* **by** *auto*
qed
qed
from *this* ‹$f \in A \rightarrow_E B$› **show** *?thesis* **by** (*auto intro*: *quotientI*)
qed

lemma *functions-of-number-partition-of*:
assumes *finite A finite B*
assumes $F \in (A \rightarrow_E B)$ // *domain-and-range-permutation A B*
shows *functions-of A B* (*number-partition-of A B F*) = F
proof −
from ‹$F \in (A \rightarrow_E B)$ // *domain-and-range-permutation A B*› **obtain** f **where** $f \in A \rightarrow_E B$
and *F-eq*: F = *domain-and-range-permutation A B* `` $\{f\}$ **using** *quotientE* **by** *blast*
have *number-partition-of A B F* = *univ* (λf. *image-mset card* (*mset-set* ((λb. $\{x \in A.\ f\ x = b\}$) ` $B - \{\{\}\}$))) F
unfolding *number-partition-of-def* **..**
also have ... = *univ* (λf. *image-mset card* (*mset-set* ((λb. $\{x \in A.\ f\ x = b\}$) ` $B - \{\{\}\}$))) (*domain-and-range-permutation A B* `` $\{f\}$)
unfolding *F-eq* **..**
also have ... = *image-mset card* (*mset-set* ((λb. $\{x \in A.\ f\ x = b\}$) ` $B - \{\{\}\}$))
using ‹*finite B*›
using *equiv-domain-and-range-permutation multiset-of-partition-cards-respects-domain-and-range-permutati*
‹$f \in A \rightarrow_E B$›
by (*subst univ-commute′*) *auto*
finally have *number-partition-of-eq*: *number-partition-of A B F* = *image-mset*

card (*mset-set* ((λb. $\{x \in A.\ f\ x = b\}$) ' $B - \{\{\}\}$)) **.**
  show *?thesis*
  proof
    show *functions-of* A B (*number-partition-of* A B F) $\subseteq F$
    proof
      fix f'
      assume $f' \in$ *functions-of* A B (*number-partition-of* A B F)
      from *this* **have** $f' \in A \rightarrow_E B$
        and *eq*: *image-mset card* (*mset-set* ((λb. $\{x \in A.\ f'\ x = b\}$) ' $B - \{\{\}\}$))
= *image-mset card* (*mset-set* ((λb. $\{x \in A.\ f\ x = b\}$) ' $B - \{\{\}\}$))
        unfolding *functions-of-def* **by** (*auto simp add*: *number-partition-of-eq*)
      note ‹$f \in A \rightarrow_E B$› ‹$f' \in A \rightarrow_E B$›
      moreover obtain p_A p_B **where** p_A *permutes* A p_B *permutes* B $\forall x{\in}A.\ f\ x$
$= p_B\ (f'\ (p_A\ x))$
        using ‹*finite* A› ‹*finite* B› ‹$f \in A \rightarrow_E B$› ‹$f' \in A \rightarrow_E B$› *eq*
          multiset-of-partition-cards-eq-implies-permutes[*of* A B f f']
        by *metis*
      ultimately show $f' \in F$
        unfolding *F-eq domain-and-range-permutation-def* **by** *auto*
    qed
  next
    show $F \subseteq$ *functions-of* A B (*number-partition-of* A B F)
    proof
      fix f'
      assume $f' \in F$
      from ‹$f' \in F$› **obtain** p_A p_B **where** p_A *permutes* A p_B *permutes* B $\forall x{\in}A.$
$f\ x = p_B\ (f'\ (p_A\ x))$
        unfolding *F-eq domain-and-range-permutation-def* **by** *auto*
      have *eq*: *image-mset card* (*mset-set* ((λb. $\{x \in A.\ f\ x = b\}$) ' $B - \{\{\}\}$)) =
image-mset card (*mset-set* ((λb. $\{x \in A.\ f'\ x = b\}$) ' $B - \{\{\}\}$))
      proof −
        have (λb. $\{x \in A.\ f\ x = b\}$) ' B = (λb. $\{x \in A.\ p_B\ (f'\ (p_A\ x)) = b\}$) ' B
          using ‹$\forall x{\in}A.\ f\ x = p_B\ (f'\ (p_A\ x))$› **by** *auto*
        from *this* **have** *image-mset card* (*mset-set* ((λb. $\{x \in A.\ f\ x = b\}$) ' B −
$\{\{\}\}$)) =
          image-mset card (*mset-set* ((λb. $\{x \in A.\ p_B\ (f'\ (p_A\ x)) = b\}$) ' $B - \{\{\}\}$))
by *simp*
        also have ... = *image-mset card* (*mset-set* ((λb. $\{x \in A.\ f'\ x = b\}$) ' B −
$\{\{\}\}$))
        using ‹p_A *permutes* A› ‹p_B *permutes* B› *permutes-implies-multiset-of-partition-cards-eq*
by *blast*
        finally show *?thesis* **.**
      qed
      moreover from ‹$f' \in F$› **have** $f' \in A \rightarrow_E B$
        unfolding *F-eq domain-and-range-permutation-def* **by** *auto*
      ultimately show $f' \in$ *functions-of* A B (*number-partition-of* A B F)
        unfolding *functions-of-def number-partition-of-eq* **by** *auto*
    qed
  qed

qed

lemma *number-partition-of-functions-of*:
assumes *finite A finite B*
assumes *number-partition* (*card A*) *N size N* ≤ *card B*
shows *number-partition-of A B* (*functions-of A B N*) = *N*
proof −
from *assms* **have** *functions-of A B N* ∈ (*A* $\rightarrow_E$ *B*) // *domain-and-range-permutation A B*
using *functions-of assms* **by** *fastforce*
from *this* **obtain** *f* **where** *f* ∈ *A* $\rightarrow_E$ *B* **and** *functions-of A B N* = *domain-and-range-permutation A B* ‘‘ {*f*}
by (*meson quotientE*)
from *this* **have** *f* ∈ *functions-of A B N*
using *equiv-domain-and-range-permutation equiv-class-self* **by** *fastforce*
have *number-partition-of A B* (*functions-of A B N*) = *univ* (λ*f*. *image-mset card* (*mset-set* ((λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}}))) (*functions-of A B N*)
unfolding *number-partition-of-def* **..**
also have . . . = *univ* (λ*f*. *image-mset card* (*mset-set* ((λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}}))) (*domain-and-range-permutation A B* ‘‘ {*f*})
unfolding ‹*functions-of A B N* = *domain-and-range-permutation A B* ‘‘ {*f*}›
..
also have . . . = *image-mset card* (*mset-set* ((λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}}))
using ‹*finite B*› ‹*f* ∈ *A* $\rightarrow_E$ *B*› *equiv-domain-and-range-permutation*
multiset-of-partition-cards-respects-domain-and-range-permutation
by (*subst univ-commute′*) *auto*
also have *image-mset card* (*mset-set* ((λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}})) = *N*
using ‹*f* ∈ *functions-of A B N*› **unfolding** *functions-of-def* **by** *simp*
finally show *?thesis* **.**
qed

13.3 Bijections

lemma *bij-betw-number-partition-of*:
assumes *finite A finite B*
shows *bij-betw* (*number-partition-of A B*) ((*A* $\rightarrow_E$ *B*) // *domain-and-range-permutation A B*) {*N*. *number-partition* (*card A*) *N* ∧ *size N* ≤ *card B*}
proof (*rule bij-betw-byWitness*[**where** *f′*=λ*M*. *functions-of A B M*])
show ∀ *F*∈(*A* $\rightarrow_E$ *B*) // *domain-and-range-permutation A B*. *functions-of A B* (*number-partition-of A B F*) = *F*
using ‹*finite A*› ‹*finite B*› **by** (*auto simp add*: *functions-of-number-partition-of*)
show ∀ *N*∈{*N*. *number-partition* (*card A*) *N* ∧ *size N* ≤ *card B*}. *number-partition-of A B* (*functions-of A B N*) = *N*
using ‹*finite A*› ‹*finite B*› **by** (*auto simp add*: *number-partition-of-functions-of*)
show *number-partition-of A B* ‘ ((*A* $\rightarrow_E$ *B*) // *domain-and-range-permutation A B*) ⊆ {*N*. *number-partition* (*card A*) *N* ∧ *size N* ≤ *card B*}
using *number-partition-of*[*of A B*] ‹*finite A*› ‹*finite B*› **by** *auto*
show *functions-of A B* ‘ {*N*. *number-partition* (*card A*) *N* ∧ *size N* ≤ *card B*} ⊆ (*A* $\rightarrow_E$ *B*) // *domain-and-range-permutation A B*

using *functions-of* ‹*finite A*› ‹*finite B*› **by** *blast*
qed

13.4 Cardinality

lemma *card-domain-and-range-permutation*:
assumes *finite A finite B*
shows *card* $((A \rightarrow_E B)$ // *domain-and-range-permutation A B*) = *Partition* (*card A* + *card B*) (*card B*)
proof −
have *bij-betw* (*number-partition-of A B*) $((A \rightarrow_E B)$ // *domain-and-range-permutation A B*) {*N*. *number-partition* (*card A*) *N* ∧ *size N* ≤ *card B*}
using ‹*finite A*› ‹*finite B*› **by** (*rule bij-betw-number-partition-of*)
from *this* **have** *card* $((A \rightarrow_E B)$ // *domain-and-range-permutation A B*) = *card* {*N*. *number-partition* (*card A*) *N* ∧ *size N* ≤ *card B*}
by (*rule bij-betw-same-card*)
also have *card* {*N*. *number-partition* (*card A*) *N* ∧ *size N* ≤ *card B*} = *Partition* (*card A* + *card B*) (*card B*)
by (*rule card-number-partitions-with-atmost-k-parts*)
finally show *?thesis* **.**
qed

end

14 Injections from A to B up to a permutation on A and B

theory *Twelvefold-Way-Entry11*
imports *Twelvefold-Way-Entry10*
begin

14.1 Properties for Bijections

lemma *all-one-implies-inj-on*:
assumes *finite A finite B*
assumes ∀ *n*. *n*∈# *N* ⟶ *n* = *1* *number-partition* (*card A*) *N* *size N* ≤ *card B*
assumes *f* ∈ *functions-of A B N*
shows *inj-on f A*
proof −
from ‹*f* ∈ *functions-of A B N*› **have** $f \in A \rightarrow_E B$
and *N* = *image-mset card* (*mset-set* ((λ*b*. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}}))
unfolding *functions-of-def* **by** *auto*
from *this* ‹∀ *n*. *n*∈# *N* ⟶ *n* = *1*› **have** *parts*: ∀ *b* ∈ *B*. *card* {*x* ∈ *A*. *f x* = *b*} = *1* ∨ {*x* ∈ *A*. *f x* = *b*} = {}
using ‹*finite B*› **by** *auto*
show *inj-on f A*
proof
fix *x y*
assume *a*: *x* ∈ *A* *y* ∈ *A* *f x* = *f y*

from ‹$f \in A \to_E B$› ‹$x \in A$› **have** $f\,x \in B$ **by** *auto*
from *a* **have** *1*: $x \in \{x' \in A.\ f\,x' = f\,x\}$ $y \in \{x' \in A.\ f\,x' = f\,x\}$ **by** *auto*
from *this* **have** *2*: *card* $\{x' \in A.\ f\,x' = f\,x\} = 1$
using *parts* ‹$f\,x \in B$› **by** *blast*
from *this* **have** *is-singleton* $\{x' \in A.\ f\,x' = f\,x\}$
by (*simp add*: *is-singleton-altdef*)
from *1 this* **show** $x = y$
by (*metis is-singletonE singletonD*)
qed
qed

lemma *inj-on-implies-all-one*:
assumes *finite A finite B*
assumes $F \in (A \to_E B)$ // *domain-and-range-permutation A B*
assumes *univ* (λf. *inj-on f A*) *F*
shows $\forall n.\ n \in\#$ *number-partition-of A B F* $\longrightarrow n = 1$
proof −
from ‹$F \in (A \to_E B)$ // *domain-and-range-permutation A B*› **obtain** *f* **where** $f \in A \to_E B$
and *F-eq*: *F* = *domain-and-range-permutation A B* `` $\{f\}$ **using** *quotientE* **by** *blast*
have *number-partition-of A B F* = *univ* (λf. *image-mset card* (*mset-set* ((λb. $\{x \in A.\ f\,x = b\}$) ` $B - \{\{\}\}$))) *F*
unfolding *number-partition-of-def* **..**
also have ... = *univ* (λf. *image-mset card* (*mset-set* ((λb. $\{x \in A.\ f\,x = b\}$) ` $B - \{\{\}\}$))) (*domain-and-range-permutation A B* `` $\{f\}$)
unfolding *F-eq* **..**
also have ... = *image-mset card* (*mset-set* ((λb. $\{x \in A.\ f\,x = b\}$) ` $B - \{\{\}\}$))
using ‹*finite B*› *equiv-domain-and-range-permutation multiset-of-partition-cards-respects-domain-and-range-*
‹$f \in A \to_E B$›
by (*subst univ-commute'*) *auto*
finally have *eq*: *number-partition-of A B F* = *image-mset card* (*mset-set* ((λb. $\{x \in A.\ f\,x = b\}$) ` $B - \{\{\}\}$)) **.**
from *iffD1*[*OF univ-commute'*, *OF equiv-domain-and-range-permutation*, *OF inj-on-respects-domain-and-range-permutation*, *OF* ‹$f \in A \to_E B$›]
assms(*4*) **have** *inj-on f A* **by** (*simp add*: *F-eq*)
have $\forall n.\ n \in\#$ *image-mset card* (*mset-set* ((λb. $\{x \in A.\ f\,x = b\}$) ` $B - \{\{\}\}$)) $\longrightarrow n = 1$
proof −
have $\forall b \in B.$ *card* $\{x \in A.\ f\,x = b\} = 1 \lor \{x \in A.\ f\,x = b\} = \{\}$
proof
fix *b*
assume $b \in B$
show *card* $\{x \in A.\ f\,x = b\} = 1 \lor \{x \in A.\ f\,x = b\} = \{\}$
proof (*cases* $b \in f$ ` *A*)
assume $b \in f$ ` *A*
from ‹*inj-on f A*› *this* **have** *is-singleton* $\{x \in A.\ f\,x = b\}$
by (*auto simp add*: *inj-on-eq-iff intro*: *is-singletonI'*)
from *this* **have** *card* $\{x \in A.\ f\,x = b\} = 1$

by (*subst is-singleton-altdef*[*symmetric*])
from *this* **show** *?thesis* **..**
next
assume $b \notin f$ ‘ A
from *this* **have** $\{x \in A.\ f\ x = b\} = \{\}$ **by** *auto*
from *this* **show** *?thesis* **..**
qed
qed
from *this* **show** *?thesis*
using ‹*finite B*› **by** *auto*
qed
from *this* **show** $\forall n.\ n \in\#$ *number-partition-of A B F* $\longrightarrow n = 1$
unfolding *eq* **by** *auto*
qed

lemma *functions-of-is-inj-on*:
assumes *finite A finite B*
assumes $\forall n.\ n \in\#\ N \longrightarrow n = 1$ *number-partition* (*card A*) *N size N* $\leq$ *card B*
shows *univ* ($\lambda f.$ *inj-on f A*) (*functions-of A B N*)
proof −
have *functions-of A B N* $\in (A \rightarrow_E B)$ // *domain-and-range-permutation A B*
using *assms functions-of* **by** *auto*
from *this* **obtain** *f* **where** *eq-f*: *functions-of A B N* = *domain-and-range-permutation A B* ‘‘ $\{f\}$ **and** $f \in A \rightarrow_E B$
using *quotientE* **by** *blast*
from *eq-f* **have** $f \in$ *functions-of A B N*
using ‹$f \in A \rightarrow_E B$› *equiv-domain-and-range-permutation equiv-class-self* **by** *fastforce*
have *inj-on f A*
using ‹$f \in$ *functions-of A B N*› *assms all-one-implies-inj-on* **by** *blast*
from *this* **show** *?thesis*
unfolding *eq-f* **using** *equiv-domain-and-range-permutation inj-on-respects-domain-and-range-permutation* ‹$f \in A \rightarrow_E B$›
by (*subst univ-commute′*) *assumption+*
qed

14.2 Bijections

lemma *bij-betw-number-partition-of*:
assumes *finite A finite B*
shows *bij-betw* (*number-partition-of A B*) ($\{f \in A \rightarrow_E B.$ *inj-on f A*$\}$ // *domain-and-range-permutation A B*) $\{N.\ (\forall n.\ n \in\#\ N \longrightarrow n = 1) \wedge$ *number-partition* (*card A*) $N \wedge$ *size N* $\leq$ *card B*$\}$
proof (*rule bij-betw-byWitness*[**where** *f′*=*functions-of A B*])
have *quotient-eq*: $\{f \in A \rightarrow_E B.$ *inj-on f A*$\}$ // *domain-and-range-permutation A B* = $\{F \in ((A \rightarrow_E B)$ // *domain-and-range-permutation A B*). *univ* ($\lambda f.$ *inj-on f A*) $F\}$
using *equiv-domain-and-range-permutation*[*of A B*] *inj-on-respects-domain-and-range-permutation*[*of A B*] **by** (*simp only*: *univ-preserves-predicate*)

show ∀ *F*∈{*f* ∈ *A* $\to_E$ *B*. *inj-on f A*} // *domain-and-range-permutation A B*.
functions-of A B (*number-partition-of A B F*) = *F*
using ‹*finite A*› ‹*finite B*› **by** (*auto simp only*: *quotient-eq functions-of-number-partition-of*)
show ∀ *N*∈ {*N*. (∀ *n*. *n*∈# *N* ⟶ *n* = *1*) ∧ *number-partition* (*card A*) *N* ∧ *size N* ≤ *card B*}. *number-partition-of A B* (*functions-of A B N*) = *N*
using ‹*finite A*› ‹*finite B*› *number-partition-of-functions-of* **by** *auto*
show *number-partition-of A B* ‘ ({*f* ∈ *A* $\to_E$ *B*. *inj-on f A*} // *domain-and-range-permutation A B*)
⊆ {*N*. (∀ *n*. *n*∈# *N* ⟶ *n* = *1*) ∧ *number-partition* (*card A*) *N* ∧ *size N* ≤ *card B*}
using ‹*finite A*› ‹*finite B*›
by (*auto simp add*: *quotient-eq number-partition-of inj-on-implies-all-one simp del*: *One-nat-def*)
show *functions-of A B* ‘ {*N*. (∀ *n*. *n*∈# *N* ⟶ *n* = *1*) ∧ *number-partition* (*card A*) *N* ∧ *size N* ≤ *card B*}
⊆ {*f* ∈ *A* $\to_E$ *B*. *inj-on f A*} // *domain-and-range-permutation A B*
using ‹*finite A*› ‹*finite B*› **by** (*auto simp add*: *quotient-eq intro*: *functions-of functions-of-is-inj-on*)
qed

lemma *bij-betw-functions-of*:
assumes *finite A finite B*
shows *bij-betw* (*functions-of A B*) {*N*. (∀ *n*. *n*∈# *N* ⟶ *n* = *1*) ∧ *number-partition* (*card A*) *N* ∧ *size N* ≤ *card B*} ({*f* ∈ *A* $\to_E$ *B*. *inj-on f A*} // *domain-and-range-permutation A B*)
proof (*rule bij-betw-byWitness*[**where** *f'*=*number-partition-of A B*])
have *quotient-eq*: {*f* ∈ *A* $\to_E$ *B*. *inj-on f A*} // *domain-and-range-permutation A B* = {*F* ∈ ((*A* $\to_E$ *B*) // *domain-and-range-permutation A B*). *univ* (λ*f*. *inj-on f A*) *F*}
using *equiv-domain-and-range-permutation*[*of A B*] *inj-on-respects-domain-and-range-permutation*[*of A B*] **by** (*simp only*: *univ-preserves-predicate*)
show ∀ *F*∈{*f* ∈ *A* $\to_E$ *B*. *inj-on f A*} // *domain-and-range-permutation A B*.
functions-of A B (*number-partition-of A B F*) = *F*
using ‹*finite A*› ‹*finite B*› **by** (*auto simp only*: *quotient-eq functions-of-number-partition-of*)
show ∀ *N*∈ {*N*. (∀ *n*. *n*∈# *N* ⟶ *n* = *1*) ∧ *number-partition* (*card A*) *N* ∧ *size N* ≤ *card B*}. *number-partition-of A B* (*functions-of A B N*) = *N*
using ‹*finite A*› ‹*finite B*› *number-partition-of-functions-of* **by** *auto*
show *number-partition-of A B* ‘ ({*f* ∈ *A* $\to_E$ *B*. *inj-on f A*} // *domain-and-range-permutation A B*)
⊆ {*N*. (∀ *n*. *n*∈# *N* ⟶ *n* = *1*) ∧ *number-partition* (*card A*) *N* ∧ *size N* ≤ *card B*}
using ‹*finite A*› ‹*finite B*›
by (*auto simp add*: *quotient-eq number-partition-of inj-on-implies-all-one simp del*: *One-nat-def*)
show *functions-of A B* ‘ {*N*. (∀ *n*. *n*∈# *N* ⟶ *n* = *1*) ∧ *number-partition* (*card A*) *N* ∧ *size N* ≤ *card B*}
⊆ {*f* ∈ *A* $\to_E$ *B*. *inj-on f A*} // *domain-and-range-permutation A B*
using ‹*finite A*› ‹*finite B*› **by** (*auto simp add*: *quotient-eq intro*: *functions-of functions-of-is-inj-on*)

qed

14.3 Cardinality

lemma *card-injective-functions-domain-and-range-permutation*:
assumes *finite A finite B*
shows *card* ($\{f \in A \rightarrow_E B.$ *inj-on f A*$\}$ // *domain-and-range-permutation A B*) = *iverson* (*card A* $\leq$ *card B*)
proof −
have *bij-betw* (*number-partition-of A B*) ($\{f \in A \rightarrow_E B.$ *inj-on f A*$\}$ // *domain-and-range-permutation A B*) $\{N.\ (\forall n.\ n \in\# N \longrightarrow n = 1) \wedge$ *number-partition* (*card A*) $N \wedge$ *size* $N \leq$ *card B*$\}$
using ‹*finite A*› ‹*finite B*› **by** (*rule bij-betw-number-partition-of*)
from *this* **have** *card* ($\{f \in A \rightarrow_E B.$ *inj-on f A*$\}$ // *domain-and-range-permutation A B*) = *card* $\{N.\ (\forall n.\ n \in\# N \longrightarrow n = 1) \wedge$ *number-partition* (*card A*) $N \wedge$ *size* $N \leq$ *card B*$\}$
by (*rule bij-betw-same-card*)
also have *card* $\{N.\ (\forall n.\ n \in\# N \longrightarrow n = 1) \wedge$ *number-partition* (*card A*) $N \wedge$ *size* $N \leq$ *card B*$\}$ = *iverson* (*card A* $\leq$ *card B*)
by (*rule card-number-partitions-with-only-parts-1*)
finally show *?thesis* **.**
qed

end

15 Surjections from A to B up to a Permutation on A and B

theory *Twelvefold-Way-Entry12*
imports *Twelvefold-Way-Entry9 Twelvefold-Way-Entry10*
begin

15.1 Properties for Bijections

lemma *size-eq-card-implies-surj-on*:
assumes *finite A finite B*
assumes *size N* = *card B*
assumes $f \in$ *functions-of A B N*
shows f ‘ $A = B$
proof −
from ‹$f \in$ *functions-of A B N*› **have** $f \in A \rightarrow_E B$ **and**
N = *image-mset card* (*mset-set* (($\lambda b.\ \{x \in A.\ f\ x = b\}$) ‘ $B - \{\{\}\}$))
unfolding *functions-of-def* **by** *auto*
from *this* ‹*size N* = *card B*› **have** *card* (($\lambda b.\ \{x \in A.\ f\ x = b\}$) ‘ $B - \{\{\}\}$) = *card B* **by** *simp*
from *this* ‹*finite B*› ‹$f \in A \rightarrow_E B$› **show** f ‘ $A = B$
using *card-eq-implies-surjective-on* **by** *blast*
qed

lemma *surj-on-implies-size-eq-card*:
assumes *finite A finite B*
assumes $F \in (A \rightarrow_E B)$ // *domain-and-range-permutation A B*
assumes *univ* (λf. *f* ‘ *A* = *B*) *F*
shows *size* (*number-partition-of A B F*) = *card B*
proof −
from ‹$F \in (A \rightarrow_E B)$ // *domain-and-range-permutation A B*› **obtain** *f* **where** $f \in A \rightarrow_E B$
and *F-eq*: *F* = *domain-and-range-permutation A B* ‘‘ {*f*} **using** *quotientE* **by** *blast*
have *number-partition-of A B F* = *univ* (λf. *image-mset card* (*mset-set* ((λb. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}}))) *F*
unfolding *number-partition-of-def* **..**
also have ... = *univ* (λf. *image-mset card* (*mset-set* ((λb. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}}))) (*domain-and-range-permutation A B* ‘‘ {*f*})
unfolding *F-eq* **..**
also have ... = *image-mset card* (*mset-set* ((λb. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}}))
using ‹*finite B*› *equiv-domain-and-range-permutation multiset-of-partition-cards-respects-domain-and-range-*
‹$f \in A \rightarrow_E B$›
by (*subst univ-commute′*) *auto*
finally have *eq*: *number-partition-of A B F* = *image-mset card* (*mset-set* ((λb. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}})) **.**
from *iffD1*[*OF univ-commute′*, *OF equiv-domain-and-range-permutation*, *OF surjective-respects-domain-and-range-permutation*, *OF* ‹$f \in A \rightarrow_E B$›]
assms(*4*) **have** *f* ‘ *A* = *B* **by** (*simp add*: *F-eq*)
have *size* (*number-partition-of A B F*) = *size* (*image-mset card* (*mset-set* ((λb. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}})))
unfolding *eq* **..**
also have ... = *card* ((λb. {*x* ∈ *A*. *f x* = *b*}) ‘ *B* − {{}}) **by** *simp*
also from ‹*f* ‘ *A* = *B*› **have** ... = *card B*
using *surjective-on-implies-card-eq* **by** *auto*
finally show *?thesis* **.**
qed

lemma *functions-of-is-surj-on*:
assumes *finite A finite B*
assumes *number-partition* (*card A*) *N size N* = *card B*
shows *univ* (λf. *f* ‘ *A* = *B*) (*functions-of A B N*)
proof −
have *functions-of A B N* $\in (A \rightarrow_E B)$ // *domain-and-range-permutation A B*
using *functions-of* ‹*finite A*› ‹*finite B*› ‹*number-partition* (*card A*) *N* › ‹*size N* = *card B*›
by *fastforce*
from *this* **obtain** *f* **where** *eq-f*: *functions-of A B N* = *domain-and-range-permutation A B* ‘‘ {*f*} **and** $f \in A \rightarrow_E B$
using *quotientE* **by** *blast*
from *eq-f* **have** *f* ∈ *functions-of A B N*
using ‹$f \in A \rightarrow_E B$› *equiv-domain-and-range-permutation equiv-class-self* **by** *fastforce*

have *f* ' *A* = *B*
using ‹*f* ∈ *functions-of A B N*› *assms size-eq-card-implies-surj-on* **by** *blast*
from *this* **show** *?thesis*
unfolding *eq-f* **using** *equiv-domain-and-range-permutation surjective-respects-domain-and-range-permutatio*
‹*f* ∈ *A* $\rightarrow_E$ *B*›
by (*subst univ-commute'*) *assumption+*
qed

15.2 Bijections

lemma *bij-betw-number-partition-of*:
assumes *finite A finite B*
shows *bij-betw* (*number-partition-of A B*) ({*f* ∈ *A* $\rightarrow_E$ *B*. *f* ' *A* = *B*} // *domain-and-range-permutation A B*) {*N*. *number-partition* (*card A*) *N* ∧ *size N* = *card B*}
proof (*rule bij-betw-byWitness*[**where** *f'*=*functions-of A B*])
have *quotient-eq*: {*f* ∈ *A* $\rightarrow_E$ *B*. *f* ' *A* = *B*} // *domain-and-range-permutation A B* = {*F* ∈ ((*A* $\rightarrow_E$ *B*) // *domain-and-range-permutation A B*). *univ* (λ*f*. *f* ' *A* = *B*) *F*}
using *equiv-domain-and-range-permutation*[*of A B*] *surjective-respects-domain-and-range-permutation*[*of A B*] **by** (*simp only*: *univ-preserves-predicate*)
show ∀ *F*∈{*f* ∈ *A* $\rightarrow_E$ *B*. *f* ' *A* = *B*} // *domain-and-range-permutation A B*.
functions-of A B (*number-partition-of A B F*) = *F*
using ‹*finite A*› ‹*finite B*› **by** (*auto simp only*: *quotient-eq functions-of-number-partition-of*)
show ∀ *N*∈{*N*. *number-partition* (*card A*) *N* ∧ *size N* = *card B*}. *number-partition-of A B* (*functions-of A B N*) = *N*
using ‹*finite A*› ‹*finite B*› **by** (*simp add*: *number-partition-of-functions-of*)
show *number-partition-of A B* ' ({*f* ∈ *A* $\rightarrow_E$ *B*. *f* ' *A* = *B*} // *domain-and-range-permutation A B*)
⊆ {*N*. *number-partition* (*card A*) *N* ∧ *size N* = *card B*}
using ‹*finite A*› ‹*finite B*› **by** (*auto simp add*: *quotient-eq number-partition-of surj-on-implies-size-eq-card*)
show *functions-of A B* ' {*N*. *number-partition* (*card A*) *N* ∧ *size N* = *card B*}
⊆ {*f* ∈ *A* $\rightarrow_E$ *B*. *f* ' *A* = *B*} // *domain-and-range-permutation A B*
using ‹*finite A*› ‹*finite B*› **by** (*auto simp add*: *quotient-eq intro*: *functions-of functions-of-is-surj-on*)
qed

lemma *bij-betw-functions-of*:
assumes *finite A finite B*
shows *bij-betw* (*functions-of A B*) {*N*. *number-partition* (*card A*) *N* ∧ *size N* = *card B*} ({*f* ∈ *A* $\rightarrow_E$ *B*. *f* ' *A* = *B*} // *domain-and-range-permutation A B*)
proof (*rule bij-betw-byWitness*[**where** *f'*=*number-partition-of A B*])
have *quotient-eq*: {*f* ∈ *A* $\rightarrow_E$ *B*. *f* ' *A* = *B*} // *domain-and-range-permutation A B* = {*F* ∈ ((*A* $\rightarrow_E$ *B*) // *domain-and-range-permutation A B*). *univ* (λ*f*. *f* ' *A* = *B*) *F*}
using *equiv-domain-and-range-permutation*[*of A B*] *surjective-respects-domain-and-range-permutation*[*of A B*] **by** (*simp only*: *univ-preserves-predicate*)
show ∀ *F*∈{*f* ∈ *A* $\rightarrow_E$ *B*. *f* ' *A* = *B*} // *domain-and-range-permutation A B*.

functions-of A B (*number-partition-of A B F*) = *F*
using ‹*finite A*› ‹*finite B*› **by** (*auto simp only*: *quotient-eq functions-of-number-partition-of*)
show ∀ *N*∈{*N*. *number-partition* (*card A*) *N* ∧ *size N* = *card B*}. *number-partition-of A B* (*functions-of A B N*) = *N*
using ‹*finite A*› ‹*finite B*› **by** (*simp add*: *number-partition-of-functions-of*)
show *number-partition-of A B* ‘ ({*f* ∈ *A* $\rightarrow_E$ *B*. *f* ‘ *A* = *B*} // *domain-and-range-permutation A B*)
⊆ {*N*. *number-partition* (*card A*) *N* ∧ *size N* = *card B*}
using ‹*finite A*› ‹*finite B*› **by** (*auto simp add*: *quotient-eq number-partition-of surj-on-implies-size-eq-card*)
show *functions-of A B* ‘ {*N*. *number-partition* (*card A*) *N* ∧ *size N* = *card B*}
⊆ {*f* ∈ *A* $\rightarrow_E$ *B*. *f* ‘ *A* = *B*} // *domain-and-range-permutation A B*
using ‹*finite A*› ‹*finite B*› **by** (*auto simp add*: *quotient-eq intro*: *functions-of functions-of-is-surj-on*)
qed

15.3 Cardinality

lemma *card-surjective-functions-domain-and-range-permutation*:
assumes *finite A finite B*
shows *card* ({*f* ∈ *A* $\rightarrow_E$ *B*. *f* ‘ *A* = *B*} // *domain-and-range-permutation A B*) = *Partition* (*card A*) (*card B*)
proof −
have *bij-betw* (*number-partition-of A B*) ({*f* ∈ *A* $\rightarrow_E$ *B*. *f* ‘ *A* = *B*} // *domain-and-range-permutation A B*) {*N*. *number-partition* (*card A*) *N* ∧ *size N* = *card B*}
using ‹*finite A*› ‹*finite B*› **by** (*rule bij-betw-number-partition-of*)
from *this* **have** *card* ({*f* ∈ *A* $\rightarrow_E$ *B*. *f* ‘ *A* = *B*} // *domain-and-range-permutation A B*) = *card* {*N*. *number-partition* (*card A*) *N* ∧ *size N* = *card B*}
by (*rule bij-betw-same-card*)
also have *card* {*N*. *number-partition* (*card A*) *N* ∧ *size N* = *card B*} = *Partition* (*card A*) (*card B*)
by (*rule card-partitions-with-k-parts*)
finally show *?thesis* **.**
qed

end

16 Cardinality of Bijections

theory *Card-Bijections*
imports
Twelvefold-Way-Entry2
Twelvefold-Way-Entry3
Twelvefold-Way-Entry5
Twelvefold-Way-Entry6
Twelvefold-Way-Entry8
Twelvefold-Way-Entry9
Twelvefold-Way-Entry11

Twelvefold-Way-Entry12
begin

16.1 Bijections from A to B

lemma *bij-betw-set-is-empty*:
  assumes *finite A finite B*
  assumes *card A ≠ card B*
  shows {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} = {}
using *assms bij-betw-same-card* **by** *blast*

lemma *card-bijections-eq-zero*:
  assumes *finite A finite B*
  assumes *card A ≠ card B*
  shows *card* {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} = *0*
using *bij-betw-set-is-empty*[*OF assms*] **by** (*simp only*: *card.empty*)

Two alternative proofs for the cardinality of bijections up to a permutation on A.

lemma
  assumes *finite A finite B*
  assumes *card A* = *card B*
  shows *card* {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} = *fact* (*card B*)
proof −
  have *card* {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} = *card* {*f* ∈ *A* $\to_E$ *B*. *inj-on f A*}
   using ‹*finite B*› ‹*card A* = *card B*› **by** (*metis bij-betw-implies-inj-on-and-card-eq*)
  also have ... = *fact* (*card B*)
   using ‹*finite A*› ‹*finite B*› ‹*card A* = *card B*› **by** (*simp add*: *card-extensional-funcset-inj-on*)
  finally show *?thesis* **.**
qed

lemma *card-bijections*:
  assumes *finite A finite B*
  assumes *card A* = *card B*
  shows *card* {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} = *fact* (*card B*)
proof −
  have *card* {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} = *card* {*f* ∈ *A* $\to_E$ *B*. *f* ‘ *A* = *B*}
    using ‹*finite A*› ‹*card A* = *card B*›
    by (*metis bij-betw-implies-surj-on-and-card-eq*)
  also have ... = *fact* (*card B*)
    using ‹*finite A*› ‹*finite B*› ‹*card A* = *card B*›
    by (*simp add*: *card-extensional-funcset-surj-on*)
  finally show *?thesis* **.**
qed

16.2 Bijections from A to B up to a Permutation on A

lemma *bij-betw-quotient-domain-permutation-eq-empty*:
  assumes *card A ≠ card B*

shows $\{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\}\ //\ \textit{domain-permutation}\ A\ B = \{\}$
using ‹*card A* ≠ *card B*› *bij-betw-same-card* **by** *auto*

lemma *card-bijections-domain-permutation-eq-0*:
assumes *card A* ≠ *card B*
shows $\textit{card}\ (\{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\}\ //\ \textit{domain-permutation}\ A\ B) = 0$
using *bij-betw-quotient-domain-permutation-eq-empty*[*OF assms*] **by** (*simp only*: *card.empty*)

Two alternative proofs for the cardinality of bijections up to a permutation on A.

lemma
assumes *finite A finite B*
assumes *card A* = *card B*
shows $\textit{card}\ (\{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\}\ //\ \textit{domain-permutation}\ A\ B) = 1$
proof −
from *assms* **have** $\{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\}\ //\ \textit{domain-permutation}\ A\ B$
$= \{f \in A \rightarrow_E B.\ \textit{inj-on}\ f\ A\}\ //\ \textit{domain-permutation}\ A\ B$
by (*metis* (*no-types*, *lifting*) *PiE-cong bij-betw-implies-inj-on-and-card-eq*)
from *this* **show** *?thesis*
using *assms* **by** (*simp add*: *card-injective-functions-domain-permutation*)
qed

lemma *card-bijections-domain-permutation-eq-1*:
assumes *finite A finite B*
assumes *card A* = *card B*
shows $\textit{card}\ (\{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\}\ //\ \textit{domain-permutation}\ A\ B) = 1$
proof −
from *assms* **have** $\{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\}\ //\ \textit{domain-permutation}\ A\ B$
$= \{f \in A \rightarrow_E B.\ f\ `\ A = B\}\ //\ \textit{domain-permutation}\ A\ B$
by (*metis* (*no-types*, *lifting*) *PiE-cong bij-betw-implies-surj-on-and-card-eq*)
from *this* **show** *?thesis*
using *assms* **by** (*simp add*: *card-surjective-functions-domain-permutation*)
qed

lemma *card-bijections-domain-permutation*:
assumes *finite A finite B*
shows $\textit{card}\ (\{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\}\ //\ \textit{domain-permutation}\ A\ B) = \textit{iverson}\ (\textit{card}\ A = \textit{card}\ B)$
using *assms card-bijections-domain-permutation-eq-0 card-bijections-domain-permutation-eq-1*
unfolding *iverson-def* **by** *auto*

16.3 Bijections from A to B up to a Permutation on B

lemma *bij-betw-quotient-range-permutation-eq-empty*:
assumes *card A* ≠ *card B*
shows $\{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\}\ //\ \textit{range-permutation}\ A\ B = \{\}$
using ‹*card A* ≠ *card B*› *bij-betw-same-card* **by** *auto*

lemma *card-bijections-range-permutation-eq-0*:
assumes *card A* ≠ *card B*
shows *card* ({*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} // *range-permutation A B*) = *0*
using *bij-betw-quotient-range-permutation-eq-empty*[*OF assms*] **by** (*simp only*: *card.empty*)

Two alternative proofs for the cardinality of bijections up to a permutation on B.

lemma
assumes *finite A finite B*
assumes *card A* = *card B*
shows *card* ({*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} // *range-permutation A B*) = *1*
proof −
from *assms* **have** {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} // *range-permutation A B* =
{*f* ∈ *A* $\to_E$ *B*. *inj-on f A*} // *range-permutation A B*
by (*metis* (*no-types*, *lifting*) *PiE-cong bij-betw-implies-inj-on-and-card-eq*)
from *this* **show** *?thesis*
using *assms* **by** (*simp add*: *iverson-def card-injective-functions-range-permutation*)
qed

lemma *card-bijections-range-permutation-eq-1*:
assumes *finite A finite B*
assumes *card A* = *card B*
shows *card* ({*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} // *range-permutation A B*) = *1*
proof −
from *assms* **have** {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} // *range-permutation A B* =
{*f* ∈ *A* $\to_E$ *B*. *f* ‘ *A* = *B*} // *range-permutation A B*
by (*metis* (*no-types*, *lifting*) *PiE-cong bij-betw-implies-surj-on-and-card-eq*)
from *this* **show** *?thesis*
using *assms* **by** (*simp add*: *card-surjective-functions-range-permutation*)
qed

lemma *card-bijections-range-permutation*:
assumes *finite A finite B*
shows *card* ({*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} // *range-permutation A B*) = *iverson* (*card A* = *card B*)
using *assms card-bijections-range-permutation-eq-0 card-bijections-range-permutation-eq-1*
unfolding *iverson-def* **by** *auto*

16.4 Bijections from A to B up to a Permutation on A and B

lemma *bij-betw-quotient-domain-and-range-permutation-eq-empty*:
assumes *card A* ≠ *card B*
shows {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} // *domain-and-range-permutation A B* = {}
using ‹*card A* ≠ *card B*› *bij-betw-same-card* **by** *auto*

lemma *card-bijections-domain-and-range-permutation-eq-0*:
assumes *card A* ≠ *card B*

shows *card* ({*f* ∈ *A* $\rightarrow_E$ *B*. *bij-betw f A B*} // *domain-and-range-permutation A B*) = *0*
using *bij-betw-quotient-domain-and-range-permutation-eq-empty*[*OF assms*] **by** (*simp only*: *card.empty*)

Two alternative proofs for the cardinality of bijections up to a permutation on A and B.

lemma
assumes *finite A finite B*
assumes *card A* = *card B*
shows *card* ({*f* ∈ *A* $\rightarrow_E$ *B*. *bij-betw f A B*} // *domain-and-range-permutation A B*) = *1*
proof −
from *assms* **have** {*f* ∈ *A* $\rightarrow_E$ *B*. *bij-betw f A B*} // *domain-and-range-permutation A B* =
{*f* ∈ *A* $\rightarrow_E$ *B*. *inj-on f A*} // *domain-and-range-permutation A B*
by (*metis* (*no-types*, *lifting*) *PiE-cong bij-betw-implies-inj-on-and-card-eq*)
from *this* **show** *?thesis*
using *assms* **by** (*simp add*: *iverson-def card-injective-functions-domain-and-range-permutation*)
qed

lemma *card-bijections-domain-and-range-permutation-eq-1*:
assumes *finite A finite B*
assumes *card A* = *card B*
shows *card* ({*f* ∈ *A* $\rightarrow_E$ *B*. *bij-betw f A B*} // *domain-and-range-permutation A B*) = *1*
proof −
from *assms* **have** {*f* ∈ *A* $\rightarrow_E$ *B*. *bij-betw f A B*} // *domain-and-range-permutation A B* =
{*f* ∈ *A* $\rightarrow_E$ *B*. *f* ‘ *A* = *B*} // *domain-and-range-permutation A B*
by (*metis* (*no-types*, *lifting*) *PiE-cong bij-betw-implies-surj-on-and-card-eq*)
from *this* **show** *?thesis*
using *assms* **by** (*simp add*: *card-surjective-functions-domain-and-range-permutation Partition-diag*)
qed

lemma *card-bijections-domain-and-range-permutation*:
assumes *finite A finite B*
shows *card* ({*f* ∈ *A* $\rightarrow_E$ *B*. *bij-betw f A B*} // *domain-and-range-permutation A B*) = *iverson* (*card A* = *card B*)
using *assms card-bijections-domain-and-range-permutation-eq-0 card-bijections-domain-and-range-permutatio*
unfolding *iverson-def* **by** *auto*

end

17 Direct Proofs for Cardinality of Bijections

theory *Card-Bijections-Direct*
imports

Equiv-Relations-on-Functions
Twelvefold-Way-Core
begin

17.1 Bijections from A to B up to a Permutation on A

17.1.1 Equivalence Class

lemma *bijections-in-domain-permutation*:
assumes *finite A finite B*
assumes *card A = card B*
shows $\{f \in A \rightarrow_E B.$ *bij-betw f A B*$\} \in \{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}$ // *domain-permutation A B*
proof −
from *assms* **obtain** *f* **where** *f*: $f \in \{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}$
by (*metis finite-same-card-bij-on-ext-funcset mem-Collect-eq*)
moreover have *proj-f*: $\{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}$ = *domain-permutation A B* ‘‘ $\{f\}$
proof
from *f* **show** $\{f \in A \rightarrow_E B.$ *bij-betw f A B*$\} \subseteq$ *domain-permutation A B* ‘‘ $\{f\}$
unfolding *domain-permutation-def*
by (*auto elim*: *obtain-domain-permutation-for-two-bijections*)
next
show *domain-permutation A B* ‘‘ $\{f\} \subseteq \{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}$
proof
fix f'
assume $f' \in$ *domain-permutation A B* ‘‘ $\{f\}$
have $(f', f) \in$ *domain-permutation A B*
using ‹$f' \in$ *domain-permutation A B* ‘‘ $\{f\}$› *equiv-domain-permutation*[*of A B*]
by (*simp add*: *equiv-class-eq-iff*)
from *this* **obtain** *p* **where** *p permutes A* $\forall x \in A.\ f'\ x = f\ (p\ x)$
unfolding *domain-permutation-def* **by** *auto*
from *this* **have** *bij-betw* $(f \circ p)$ *A B*
using *bij-betw-comp-iff f permutes-imp-bij* **by** *fastforce*
from *this* **have** *bij-betw* f' *A B*
using ‹$\forall x \in A.\ f'\ x = f\ (p\ x)$›
by (*metis* (*mono-tags*, *lifting*) *bij-betw-cong comp-apply*)
moreover have $f' \in A \rightarrow_E B$
using ‹$f' \in$ *domain-permutation A B* ‘‘ $\{f\}$›
unfolding *domain-permutation-def* **by** *auto*
ultimately show $f' \in \{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}$ **by** *simp*
qed
qed
ultimately show *?thesis* **by** (*simp add*: *quotientI*)
qed

lemma *bij-betw-quotient-domain-permutation-eq*:
assumes *finite A finite B*
assumes *card A = card B*

shows {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} // *domain-permutation A B* = {{*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*}}
proof
show {{*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*}} ⊆ {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} // *domain-permutation A B*
by (*simp add*: *bijections-in-domain-permutation*[*OF assms*])
next
show {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} // *domain-permutation A B* ⊆ {{*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*}}
proof
fix *F*
assume *F-in*: *F* ∈ {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} // *domain-permutation A B*
have {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} // *domain-permutation A B* = {*F* ∈ ((*A* $\to_E$ *B*) // *domain-permutation A B*). *univ* (λ*f*. *bij-betw f A B*) *F*}
using *equiv-domain-permutation*[*of A B*] *bij-betw-respects-domain-permutation*[*of A B*] **by** (*simp only*: *univ-preserves-predicate*)
from *F-in this* **have** *F* ∈ (*A* $\to_E$ *B*) // *domain-permutation A B*
and *univ* (λ*f*. *bij-betw f A B*) *F*
by *blast*+
have *F* = {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*}
proof
have ∀*f* ∈ *F*. *f* ∈ *A* $\to_E$ *B*
using ‹*F* ∈ (*A* $\to_E$ *B*) // *domain-permutation A B*›
by (*metis ImageE equiv-class-eq-iff equiv-domain-permutation quotientE*)
moreover have ∀*f* ∈ *F*. *bij-betw f A B*
using *univ-predicate-impl-forall*[*OF equiv-domain-permutation bij-betw-respects-domain-permutation*]
using ‹*F* ∈ (*A* $\to_E$ *B*) // *domain-permutation A B*› ‹*univ* (λ*f*. *bij-betw f A B*) *F*›
by *auto*
ultimately show *F* ⊆ {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} **by** *auto*
next
show {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*} ⊆ *F*
proof
fix *f'*
assume *f'* ∈ {*f* ∈ *A* $\to_E$ *B*. *bij-betw f A B*}
from *this* **have** *f'* ∈ *A* $\to_E$ *B bij-betw f' A B* **by** *auto*
obtain *f* **where** *f* ∈ *A* $\to_E$ *B* **and** *F* = *domain-permutation A B* `` {*f*}
using ‹*F* ∈ (*A* $\to_E$ *B*) // *domain-permutation A B*› **by** (*auto elim*: *quotientE*)
have *bij-betw f A B*
using *univ-commute'*[*OF equiv-domain-permutation bij-betw-respects-domain-permutation*]
using ‹*f* ∈ *A* $\to_E$ *B*› ‹*F* = *domain-permutation A B* `` {*f*}› ‹*univ* (λ*f*. *bij-betw f A B*) *F*›
by *auto*
obtain *p* **where** *p permutes A* ∀ *x*∈*A*. *f x* = *f'* (*p x*)
using *obtain-domain-permutation-for-two-bijections*
using ‹*bij-betw f A B*› ‹*bij-betw f' A B*› **by** *blast*
from *this* ‹*f* ∈ *A* $\to_E$ *B*› ‹*f'* ∈ *A* $\to_E$ *B*›
have (*f*, *f'*) ∈ *domain-permutation A B*

unfolding *domain-permutation-def* **by** *auto*
from *this* **show** $f' \in F$
using ‹$F = \textit{domain-permutation}\ A\ B$ ‘‘ $\{f\}$› **by** *simp*
qed
qed
from *this* **show** $F \in \{\{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\}\}$ **by** *simp*
qed
qed

17.1.2 Cardinality

lemma
assumes *finite A finite B*
assumes $\textit{card}\ A = \textit{card}\ B$
shows $\textit{card}\ (\{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\}\ //\ \textit{domain-permutation}\ A\ B) = 1$
using *bij-betw-quotient-domain-permutation-eq*[*OF assms*] **by** *auto*

17.2 Bijections from A to B up to a Permutation on B

17.2.1 Equivalence Class

lemma *bijections-in-range-permutation*:
assumes *finite A finite B*
assumes $\textit{card}\ A = \textit{card}\ B$
shows $\{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\} \in \{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\}\ //\ \textit{range-permutation}\ A\ B$
proof −
from *assms* **obtain** *f* **where** *f*: $f \in \{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\}$
by (*metis finite-same-card-bij-on-ext-funcset mem-Collect-eq*)
moreover have *proj-f*: $\{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\} = \textit{range-permutation}\ A\ B$ ‘‘ $\{f\}$
proof
from *f* **show** $\{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\} \subseteq \textit{range-permutation}\ A\ B$ ‘‘ $\{f\}$
unfolding *range-permutation-def*
by (*auto elim*: *obtain-range-permutation-for-two-bijections*)
next
show $\textit{range-permutation}\ A\ B$ ‘‘ $\{f\} \subseteq \{f \in A \rightarrow_E B.\ \textit{bij-betw}\ f\ A\ B\}$
proof
fix f'
assume $f' \in \textit{range-permutation}\ A\ B$ ‘‘ $\{f\}$
have $(f', f) \in \textit{range-permutation}\ A\ B$
using ‹$f' \in \textit{range-permutation}\ A\ B$ ‘‘ $\{f\}$› *equiv-range-permutation*[*of A B*]
by (*simp add*: *equiv-class-eq-iff*)
from *this* **obtain** *p* **where** *p permutes B* $\forall x \in A.\ f'\ x = p\ (f\ x)$
unfolding *range-permutation-def* **by** *auto*
from *this* **have** $\textit{bij-betw}\ (p \circ f)\ A\ B$
using *bij-betw-comp-iff f permutes-imp-bij* **by** *fastforce*
from *this* **have** $\textit{bij-betw}\ f'\ A\ B$
using ‹$\forall x \in A.\ f'\ x = p\ (f\ x)$›
by (*metis* (*mono-tags*, *lifting*) *bij-betw-cong comp-apply*)

moreover have $f' \in A \to_E B$
using ‹$f' \in$ *range-permutation* A B ‘‘ $\{f\}$›
unfolding *range-permutation-def* **by** *auto*
ultimately show $f' \in \{f \in A \to_E B.$ *bij-betw* f A $B\}$ **by** *simp*
qed
qed
ultimately show *?thesis* **by** (*simp add*: *quotientI*)
qed

lemma *bij-betw-quotient-range-permutation-eq*:
assumes *finite* A *finite* B
assumes *card* A = *card* B
shows $\{f \in A \to_E B.$ *bij-betw* f A $B\}$ // *range-permutation* A B = $\{\{f \in A \to_E B.$ *bij-betw* f A $B\}\}$
proof
show $\{\{f \in A \to_E B.$ *bij-betw* f A $B\}\} \subseteq \{f \in A \to_E B.$ *bij-betw* f A $B\}$ // *range-permutation* A B
by (*simp add*: *bijections-in-range-permutation*[*OF assms*])
next
show $\{f \in A \to_E B.$ *bij-betw* f A $B\}$ // *range-permutation* A $B \subseteq \{\{f \in A \to_E B.$ *bij-betw* f A $B\}\}$
proof
fix F
assume *F-in*: $F \in \{f \in A \to_E B.$ *bij-betw* f A $B\}$ // *range-permutation* A B
have $\{f \in A \to_E B.$ *bij-betw* f A $B\}$ // *range-permutation* A B = $\{F \in ((A \to_E B)$ // *range-permutation* A $B)$. *univ* ($\lambda f.$ *bij-betw* f A B) $F\}$
using *equiv-range-permutation*[*of* A B] *bij-betw-respects-range-permutation*[*of* A B] **by** (*simp only*: *univ-preserves-predicate*)
from *this F-in* **have** $F \in (A \to_E B)$ // *range-permutation* A B
and *univ* ($\lambda f.$ *bij-betw* f A B) F **by** *blast+*
have $F = \{f \in A \to_E B.$ *bij-betw* f A $B\}$
proof
have $\forall f \in F.\ f \in A \to_E B$
using ‹$F \in (A \to_E B)$ // *range-permutation* A B›
by (*metis ImageE equiv-class-eq-iff equiv-range-permutation quotientE*)
moreover have $\forall f \in F.$ *bij-betw* f A B
using *univ-predicate-impl-forall*[*OF equiv-range-permutation bij-betw-respects-range-permutation*]
using ‹$F \in (A \to_E B)$ // *range-permutation* A B› ‹*univ* ($\lambda f.$ *bij-betw* f A B) F›
by *auto*
ultimately show $F \subseteq \{f \in A \to_E B.$ *bij-betw* f A $B\}$ **by** *auto*
next
show $\{f \in A \to_E B.$ *bij-betw* f A $B\} \subseteq F$
proof
fix f'
assume $f' \in \{f \in A \to_E B.$ *bij-betw* f A $B\}$
from *this* **have** $f' \in A \to_E B$ *bij-betw* f' A B **by** *auto*
obtain f **where** $f \in A \to_E B$ **and** F = *range-permutation* A B ‘‘ $\{f\}$
using ‹$F \in (A \to_E B)$ // *range-permutation* A B› **by** (*auto elim*: *quotientE*)

have *bij-betw f A B*
using *univ-commute'*[*OF equiv-range-permutation bij-betw-respects-range-permutation*]
using ‹$f \in A \rightarrow_E B$› ‹F = *range-permutation A B* ‘‘ {f}› ‹*univ* (λf. *bij-betw f A B*) F›
by *auto*
obtain p **where** p *permutes* B $\forall x{\in}A.\ f\ x = p\ (f'\ x)$
using *obtain-range-permutation-for-two-bijections*
using ‹*bij-betw f A B*› ‹*bij-betw f' A B*› **by** *blast*
from *this* ‹$f \in A \rightarrow_E B$› ‹$f' \in A \rightarrow_E B$›
have $(f, f') \in$ *range-permutation A B*
unfolding *range-permutation-def* **by** *auto*
from *this* **show** $f' \in F$
using ‹F = *range-permutation A B* ‘‘ {f}› **by** *simp*
qed
qed
from *this* **show** $F \in$ {{$f \in A \rightarrow_E B$. *bij-betw f A B*}} **by** *simp*
qed
qed

17.2.2 Cardinality

lemma *card-bijections-range-permutation-eq-1*:
assumes *finite A finite B*
assumes *card A = card B*
shows *card* ({$f \in A \rightarrow_E B$. *bij-betw f A B*} // *range-permutation A B*) = *1*
using *bij-betw-quotient-range-permutation-eq*[*OF assms*] **by** *auto*

17.3 Bijections from A to B up to a Permutation on A and B

17.3.1 Equivalence Class

lemma *bijections-in-domain-and-range-permutation*:
assumes *finite A finite B*
assumes *card A = card B*
shows {$f \in A \rightarrow_E B$. *bij-betw f A B*} $\in$ {$f \in A \rightarrow_E B$. *bij-betw f A B*} // *domain-and-range-permutation A B*
proof −
from *assms* **obtain** f **where** f: $f \in$ {$f \in A \rightarrow_E B$. *bij-betw f A B*}
by (*metis finite-same-card-bij-on-ext-funcset mem-Collect-eq*)
moreover have *proj-f*: {$f \in A \rightarrow_E B$. *bij-betw f A B*} = *domain-and-range-permutation A B* ‘‘ {f}
proof
have *id permutes A* **by** (*simp add*: *permutes-id*)
from f *this* **show** {$f \in A \rightarrow_E B$. *bij-betw f A B*} $\subseteq$ *domain-and-range-permutation A B* ‘‘ {f}
unfolding *domain-and-range-permutation-def*
by (*fastforce elim*: *obtain-range-permutation-for-two-bijections*)
next

show *domain-and-range-permutation A B* ‘‘ $\{f\} \subseteq \{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}$
proof
fix f'
assume $f' \in$ *domain-and-range-permutation A B* ‘‘ $\{f\}$
have $(f', f) \in$ *domain-and-range-permutation A B*
using ‹$f' \in$ *domain-and-range-permutation A B* ‘‘ $\{f\}$› *equiv-domain-and-range-permutation*[*of A B*]
by (*simp add*: *equiv-class-eq-iff*)
from *this* **obtain** p_A p_B **where** p_A *permutes A* p_B *permutes B*
and $\forall x \in A.\ f'\ x = p_B\ (f\ (p_A\ x))$
unfolding *domain-and-range-permutation-def* **by** *auto*
from *this* **have** *bij-betw* $(p_B \circ f \circ p_A)$ *A B*
using *bij-betw-comp-iff f permutes-imp-bij*
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq*)
from *this* **have** *bij-betw* f' *A B*
using ‹$\forall x \in A.\ f'\ x = p_B\ (f\ (p_A\ x))$›
by (*auto intro*: *bij-betw-congI*)
moreover have $f' \in A \rightarrow_E B$
using ‹$f' \in$ *domain-and-range-permutation A B* ‘‘ $\{f\}$›
unfolding *domain-and-range-permutation-def* **by** *auto*
ultimately show $f' \in \{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}$ **by** *simp*
qed
qed
ultimately show *?thesis* **by** (*simp add*: *quotientI*)
qed

lemma *bij-betw-quotient-domain-and-range-permutation-eq*:
assumes *finite A finite B*
assumes *card A* = *card B*
shows $\{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}$ // *domain-and-range-permutation A B* = $\{\{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}\}$
proof
show $\{\{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}\}$
$\subseteq \{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}$ // *domain-and-range-permutation A B*
using *bijections-in-domain-and-range-permutation*[*OF assms*] **by** *auto*
next
show $\{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}$ // *domain-and-range-permutation A B* $\subseteq$ $\{\{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}\}$
proof
fix F
assume *F-in*: $F \in \{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}$ // *domain-and-range-permutation A B*
have $\{f \in A \rightarrow_E B.$ *bij-betw f A B*$\}$ // *domain-and-range-permutation A B* = $\{F \in ((A \rightarrow_E B)$ // *domain-and-range-permutation A B*). *univ* ($\lambda f.$ *bij-betw f A B*) $F\}$
using *equiv-domain-and-range-permutation*[*of A B*] *bij-betw-respects-domain-and-range-permutation*[*of A B*] **by** (*simp only*: *univ-preserves-predicate*)
from *F-in this* **have** $F \in (A \rightarrow_E B)$ // *domain-and-range-permutation A B*

and *univ* (λ*f*. *bij-betw f A B*) *F* **by** *blast+*
have *F* = {*f* ∈ *A* $\rightarrow_E$ *B*. *bij-betw f A B*}
proof
have ∀ *f* ∈ *F*. *f* ∈ *A* $\rightarrow_E$ *B*
using ‹*F* ∈ (*A* $\rightarrow_E$ *B*) // *domain-and-range-permutation A B*›
by (*metis ImageE equiv-class-eq-iff equiv-domain-and-range-permutation quotientE*)
moreover have ∀ *f* ∈ *F*. *bij-betw f A B*
using *univ-predicate-impl-forall*[*OF equiv-domain-and-range-permutation bij-betw-respects-domain-and-range-permutation*]
using ‹*F* ∈ (*A* $\rightarrow_E$ *B*) // *domain-and-range-permutation A B*› ‹*univ* (λ*f*. *bij-betw f A B*) *F*›
by *auto*
ultimately show *F* ⊆ {*f* ∈ *A* $\rightarrow_E$ *B*. *bij-betw f A B*} **by** *auto*
next
show {*f* ∈ *A* $\rightarrow_E$ *B*. *bij-betw f A B*} ⊆ *F*
proof
fix *f'*
assume *f'* ∈ {*f* ∈ *A* $\rightarrow_E$ *B*. *bij-betw f A B*}
from *this* **have** *f'* ∈ *A* $\rightarrow_E$ *B bij-betw f' A B* **by** *auto*
obtain *f* **where** *f* ∈ *A* $\rightarrow_E$ *B* **and** *F* = *domain-and-range-permutation A B* `` {*f*}
using ‹*F* ∈ (*A* $\rightarrow_E$ *B*) // *domain-and-range-permutation A B*› **by** (*auto elim*: *quotientE*)
have *bij-betw f A B*
using *univ-commute'*[*OF equiv-domain-and-range-permutation bij-betw-respects-domain-and-range-perm*
using ‹*f* ∈ *A* $\rightarrow_E$ *B*› ‹*F* = *domain-and-range-permutation A B* `` {*f*}› ‹*univ* (λ*f*. *bij-betw f A B*) *F*›
by *auto*
obtain *p* **where** *p permutes A* ∀ *x*∈*A*. *f x* = *f'* (*p x*)
using *obtain-domain-permutation-for-two-bijections*
using ‹*bij-betw f A B*› ‹*bij-betw f' A B*› **by** *blast*
moreover have *id permutes B* **by** (*simp add*: *permutes-id*)
moreover note ‹*f* ∈ *A* $\rightarrow_E$ *B*› ‹*f'* ∈ *A* $\rightarrow_E$ *B*›
ultimately have (*f*, *f'*) ∈ *domain-and-range-permutation A B*
unfolding *domain-and-range-permutation-def id-def* **by** *auto*
from *this* **show** *f'* ∈ *F*
using ‹*F* = *domain-and-range-permutation A B* `` {*f*}› **by** *simp*
qed
qed
from *this* **show** *F* ∈ {{*f* ∈ *A* $\rightarrow_E$ *B*. *bij-betw f A B*}} **by** *simp*
qed
qed

17.3.2 Cardinality

lemma *card-bijections-domain-and-range-permutation-eq-1*:
assumes *finite A finite B*
assumes *card A* = *card B*

shows *card* ({*f* ∈ *A* $\rightarrow_E$ *B*. *bij-betw f A B*} // *domain-and-range-permutation A B*) = *1*
using *bij-betw-quotient-domain-and-range-permutation-eq*[*OF assms*] **by** *auto*

end

18 The Twelvefold Way

theory *Twelvefold-Way*
imports
Preliminaries
Twelvefold-Way-Core
Equiv-Relations-on-Functions
Twelvefold-Way-Entry1
Twelvefold-Way-Entry2
Twelvefold-Way-Entry4
Twelvefold-Way-Entry5
Twelvefold-Way-Entry6
Twelvefold-Way-Entry7
Twelvefold-Way-Entry8
Twelvefold-Way-Entry9
Twelvefold-Way-Entry3
Twelvefold-Way-Entry10
Twelvefold-Way-Entry11
Twelvefold-Way-Entry12
Card-Bijections
Card-Bijections-Direct
begin

end

References

[1] K. P. Bogart. *Combinatorics Through Guided Discovery.* 2004.

[2] L. Bulwahn. Cardinality of set partitions. *Archive of Formal Proofs*, Dec. 2015. http://isa-afp.org/entries/Card_Partitions.shtml, Formal proof development.

[3] L. Bulwahn. Cardinality of multisets. *Archive of Formal Proofs*, June 2016. http://isa-afp.org/entries/Card_Multisets.shtml, Formal proof development.

[4] L. Bulwahn. Cardinality of number partitions. *Archive of Formal Proofs*, Jan. 2016. http://isa-afp.org/entries/Card_Number_Partitions.shtml, Formal proof development.

[5] R. P. Stanley. *Enumerative Combinatorics. Volume 1.* Cambridge studies in advanced mathematics. Cambridge University Press, Cambridge, New York, second edition, 2012.

[6] Wikipedia. Twelvefold way — wikipedia, the free encyclopedia, 2016. [Online; accessed 4-October-2016].