Involutions2Squares

Maksym Bortin

March 17, 2025

Abstract

This theory contains the involution-based proof of the 'two squares' theorem from THE BOOK.

Contents

theory *Involutions2Squares*
imports *Main*
begin

1 A few basic properties

lemma *nat-sqr* :
shows $nat(n^2) = (nat(abs\ n))^2$
by(*rule int-int-eq*[*THEN iffD1*], *simp*)

lemma *nat-mod-int* :
assumes $n\ mod\ m = k$
shows $int\ n\ mod\ int\ m = int\ k$
by (*metis assms of-nat-mod*)

lemma *sqr-geq-nat* :
shows $(n::nat) \leq n^2$
using *power2-nat-le-imp-le* **by** *simp*

lemma *sqr-geq-abs* :
  shows $abs(n::int) \leq n^2$
proof(*rule nat-le-eq-zle*[*THEN iffD1*], *simp*)
  show $nat\ |n| \leq nat\ (n^2)$
    using *nat-sqr sqr-geq-nat* **by** *presburger*
qed

lemma *sqr-fix-nat* :
  assumes $(n::nat) = n^2$
  shows $n = 0 \lor n = 1$
  using *assms numeral-2-eq-2* **by** *fastforce*

lemma *card1* :
  shows $(card\{a,\ b\} = Suc\ 0) = (a = b)$
  using *singleton-insert-inj-eq′* **by** *fastforce*

lemma *card2* :
  shows $card\{a,\ b\} \geq Suc\ 0 \land card\{a,\ b\} \leq 2$
  by (*simp add*: *card-insert-if*)

2 The relevant properties of involutions

definition *involution-on* $A\ \varphi = (\varphi\ `\ A \subseteq A \land (\forall x{\in}A.\ \varphi(\varphi\ x) = x))$

lemma *involution-bij* :
  assumes *involution-on* $A\ \varphi$
  shows *bij-betw* $\varphi\ A\ A$
  using *assms bij-betw-byWitness involution-on-def* **by** *fast*

lemma *involution-sub-bij* :
  assumes *involution-on* $A\ \varphi$
      and $S \subseteq A$
      and $\forall x{\in}A.\ (x \in S) = (\varphi\ x \notin S)$
 shows *bij-betw* $\varphi\ S\ (A - S)$
proof(*simp add*: *bij-betw-def*, *rule conjI*)
  show *inj-on* $\varphi\ S$
    by (*meson assms bij-betw-def inj-on-subset involution-bij*)
next
  show $\varphi\ `\ S = A - S$
  proof(*rule set-eqI*, *clarsimp*)
    fix x **show** $(x \in \varphi\ `\ S) = (x \in A \land x \notin S)$ (**is** *?L* = *?R*)
    proof
      assume *?L* **thus** *?R*
          by(*metis assms bij-betw-imp-surj-on f-inv-into-f image-eqI inv-into-into involution-bij subset-iff*)

```
    next
      assume ?R thus ?L
        by (metis assms(1) assms(3) image-iff involution-on-def)
    qed
  qed
qed
```

```
lemma involution-sub-card :
  assumes involution-on A φ
      and finite A
      and S ⊆ A
      and ∀x∈A. (x ∈ S) = (φ x ∉ S)
    shows 2 ∗ card S = card A
proof −
  have card S = card (A − S)
    using assms bij-betw-same-card involution-sub-bij by blast

  also have ... = card A − card S
    by (meson assms card-Diff-subset rev-finite-subset)

  finally show ?thesis by simp
qed
```

2.1 Unions of preimage/image sets, fixed points

```
definition preimg-img-on A φ = (⋃x∈A. {{x, φ x}})
definition fixpoints-on A φ = {x∈A. φ x = x}
```

```
lemma preimg-img-on-Union :
  assumes φ ‘ A ⊆ A
  shows A = ⋃(preimg-img-on A φ)
  using assms by(fastforce simp: preimg-img-on-def)

lemma preimg-img-on-finite :
  assumes finite A
  shows finite (preimg-img-on A φ)
  by(simp add: assms preimg-img-on-def)

lemma fixpoints-on-finite :
  assumes finite A
  shows finite (fixpoints-on A φ)
  by(simp add: assms fixpoints-on-def)

lemma preimg-img-on-card :
  assumes x ∈ preimg-img-on A φ
  shows 1 ≤ card x ∧ card x ≤ 2
```

using *assms* **by**(*fastforce simp*: *preimg-img-on-def card2*)

corollary *preimg-img-on-eq* :
shows *preimg-img-on* $A\ \varphi = \{x \in$ *preimg-img-on* $A\ \varphi.$ *card* $x = 1\} \cup$
$\{x \in$ *preimg-img-on* $A\ \varphi.$ *card* $x = 2\}$
proof(*rule equalityI*[*rotated 1*], *clarsimp*+)
fix x **assume** *card* $x \neq 2$ **and** $x \in$ *preimg-img-on* $A\ \varphi$
thus *card* $x =$ *Suc 0*
using *preimg-img-on-card* **by** *fastforce*
qed

lemma *fixpoints-on-card-eq* :
shows *card*(*fixpoints-on* $A\ \varphi$) $=$ *card* $\{x \in$ *preimg-img-on* $A\ \varphi.$ *card* $x = 1\}$
proof −
have *bij-betw* $(\lambda x.\ \{x\})$ (*fixpoints-on* $A\ \varphi$)
$\{x.\ x \in$ *preimg-img-on* $A\ \varphi \wedge$ *card* $x = 1\}$
by(*fastforce simp*: *bij-betw-def fixpoints-on-def preimg-img-on-def card1*)
thus *?thesis* **by**(*rule bij-betw-same-card*)
qed

lemma *preimg-img-on-disjoint* :
assumes *involution-on* $A\ \varphi$
shows *pairwise disjnt* (*preimg-img-on* $A\ \varphi$)
proof(*clarsimp simp*: *pairwise-def disjnt-def preimg-img-on-def*)
fix $u\ v$ **assume** b: $u \in A$ **and** c: $v \in A$ **and** d: $\{u, \varphi\ u\} \neq \{v, \varphi\ v\}$
hence e: $u \neq v$ **by** *clarsimp*
with b **and** c **have** f: $\varphi\ u \neq \varphi\ v$ **by** (*metis assms involution-on-def*)
have $(\varphi\ v = u) = (v = \varphi\ u)$ **by** (*metis assms b c involution-on-def*)
with d **have** $\varphi\ v \neq u \wedge v \neq \varphi\ u$ **by** *fastforce*
with e **and** f **show** $v \neq u \wedge v \neq \varphi\ u \wedge \varphi\ v \neq u \wedge \varphi\ v \neq \varphi\ u$ **by** *simp*
qed

theorem *involution-dom-card-sum* :
assumes *involution-on* $A\ \varphi$
and *finite* A
shows *card* $A =$ *card* (*fixpoints-on* $A\ \varphi$) $+$
$2 *$ *card* $\{x \in$ *preimg-img-on* $A\ \varphi.$ *card* $x = 2\}$
proof −
have *eq*: $\{x \in$ *preimg-img-on* $A\ \varphi.$ *card* $x =$ *Suc 0*$\} \cap$
$\{x \in$ *preimg-img-on* $A\ \varphi.$ *card* $x = 2\} = \{\}$
by *fastforce*

have *f1* : *finite* $\{x \in$ *preimg-img-on* $A\ \varphi.$ *card* $x = 1\}$

by (*simp add*: *assms preimg-img-on-finite*)
have *f2* : *finite* $\{x \in$ *preimg-img-on A* φ. *card* $x = 2\}$
by (*simp add*: *assms preimg-img-on-finite*)

have *card A* = *card* ($\bigcup$(*preimg-img-on A* φ))
by (*metis assms*(*1*) *involution-on-def preimg-img-on-Union*)

also have ... = *sum card* (*preimg-img-on A* φ)
by (*metis assms*(*1*) *card-Union-disjoint card-eq-0-iff not-one-le-zero preimg-img-on-card*

preimg-img-on-disjoint)

also have ... = *sum card* ($\{x \in$ *preimg-img-on A* φ. *card* $x = 1\}$ $\cup$
$\{x \in$ *preimg-img-on A* φ. *card* $x = 2\}$)
by (*metis preimg-img-on-eq*)

also have ... = *sum card* $\{x \in$ *preimg-img-on A* φ. *card* $x = 1\}$ +
sum card $\{x \in$ *preimg-img-on A* φ. *card* $x = 2\}$
by (*metis* (*no-types*, *lifting*) *Collect-cong One-nat-def eq f1 f2 sum.union-disjoint*)

also have ... = *card* (*fixpoints-on A* φ) +
2 $*$ *card* $\{x \in$ *preimg-img-on A* φ. *card* $x = 2\}$
by(*simp add*: *fixpoints-on-card-eq*)

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

corollary *involution-dom-fixpoints-parity* :
assumes *involution-on A* φ
and *finite A*
shows *odd*(*card A*) = *odd*(*card*(*fixpoints-on A* φ))
using *assms involution-dom-card-sum* **by** *fastforce*

3 Primes and the two squares theorem

definition *is-prime* (n :: *nat*) = ($n > 1 \wedge (\forall d.\ d$ *dvd* $n \longrightarrow d = 1 \vee d = n$))

lemma *prime-factors* :
assumes *is-prime p*
and $p = n * m$
shows $(n = 1 \wedge m = p) \vee (n = p \wedge m = 1)$
using *assms* **proof**(*clarsimp simp*: *is-prime-def*)
assume $\forall d.\ d$ *dvd* $n * m \longrightarrow d =$ *Suc 0* $\vee\ d = n * m$
hence *a*: $n =$ *Suc 0* $\vee\ n = n * m \wedge m =$ *Suc 0* $\vee\ m = n * m$
by (*meson dvd-triv-left dvd-triv-right*)
assume $0 < n \wedge$ *Suc 0* $\neq m \vee m \neq$ *Suc 0* **and** *Suc 0* $< n*m$
with *a* **show** $n =$ *Suc 0* $\wedge$ ($m = 0 \vee$ *Suc 0* $= n$) **by** *fastforce*
qed

lemma *prime-not-sqr* :
  assumes *is-prime p*
  shows $p \neq n^2$
  by (*metis assms is-prime-def order-less-irrefl power2-eq-square prime-factors*)

lemma *int-prime-not-sqr* :
  assumes *is-prime p*
  shows *int* $p \neq n^2$
  by (*metis assms nat-int nat-sqr prime-not-sqr*)

lemma *prime-gr4* :
  assumes *is-prime p*
    and p *mod* $4 = 1$
  shows $p > 4$
proof(*rule ccontr*, *drule leI*)
  assume $p \leq 4$
  thus *False*
   by (*metis assms dvd-imp-mod-0 dvd-triv-left is-prime-def mod-less mult.right-neutral*

     order-less-le zero-neq-one)
qed

theorem *two-squares* :
  assumes *a*: *is-prime p*
    and *b*: p *mod* $4 = 1$
  shows $\exists\, n\ m.\ p = n^2 + m^2$
proof −
  let $?S = \{(u, v, w).\ int\ p = 4 * u * v + w^2 \wedge u > 0 \wedge v > 0\}$
  let $?T = ?S \cap \{(u, v, w).\ w > 0\}$
  let $?U = ?S \cap \{(u, v, w).\ u - v + w > 0\}$
  let $?f = \lambda(u, v, w).\ (v, u, -w)$
  let $?g = \lambda(u, v, w).\ (u - v + w, v, 2 * v - w)$
  let $?h = \lambda(u, v, w).\ (v, u, w)$

  have *w-neq0* : $\forall\, u\ v\ w.\ (u, v, w) \in ?S \longrightarrow w \neq 0$
  proof *clarsimp*
    fix *u v* **assume** *int* $p = 4 * u * v$
    hence 4 *dvd int p* **by** *simp*
    hence 4 *dvd p* **by** *presburger*
    with *b* **show** *False* **by** *simp*
  qed

  have *fin-S* : *finite ?S*
  proof −
    have *uv-comm* : $\forall\, u\ v\ w.\ (u, v, w) \in ?S \longrightarrow (v, u, w) \in ?S$ **by** *simp*

```
  have bound1 : ∀ u v w. (u, v, w) ∈ ?S ⟶ u ≤ (int p − 1) div 4
  proof clarsimp
     fix u v w assume 1: int p = 4 ∗ u ∗ v + w² and 2: (0::int) < v and 3:
(0::int) < u
    with 2 and 3 have 4: 4 ∗ u ≤ 4 ∗ u ∗ v by simp
    have w ≠ (0::int) using 1 2 3 w-neq0 by simp
    hence 1 ≤ w²
      by (metis add-0 linorder-not-le power2-less-eq-zero-iff zless-imp-add1-zle)
    with 4 have 5: 4 ∗ u ≤ 4 ∗ u ∗ v + w² − 1 by simp
    note zdiv-mono1[OF 5, where b=4::int, simplified]
    thus u ≤ (4 ∗ u ∗ v + w² − 1) div 4 .
  qed

  have bound2 : ∀ u v w. (u, v, w) ∈ ?S ⟶ v ≤ (int p − 1) div 4
    using bound1 uv-comm by blast

  have bound3 : ∀ u v w. (u, v, w) ∈ ?S ⟶ |w| ≤ int p
  proof clarsimp
    fix u v w::int
    have 1: |w| ≤ w² by(rule sqr-geq-abs)
    assume (0::int) < u and (0::int) < v
    hence 0 < u ∗ v by(rule mult-pos-pos)
    hence 0 ≤ u ∗ v by simp
    hence w² ≤ 4 ∗ u ∗ v + w² by simp
    with 1 show |w| ≤ 4 ∗ u ∗ v + w² by simp
  qed

  let ?prod = {1..(int p − 1) div 4} × {1..(int p − 1) div 4} × {− int p..int p}

  have prod: ∀ u v w. (u, v, w) ∈ ?S ⟶ (u, v, w) ∈ ?prod
  proof(intro allI)
    fix u v w show (u, v, w) ∈ ?S ⟶ (u, v, w) ∈ ?prod
    proof(rule impI)
      assume 1: (u, v, w) ∈ ?S
      note bound1[rule-format, OF 1] and
           bound2[rule-format, OF 1]
      with 1 show (u, v, w) ∈ ?prod
      proof simp
        have |w| ≤ int p by(rule bound3[rule-format, OF 1])
        hence w ≤ int p ∧ −w ≤ int p by(rule abs-le-iff[THEN iffD1])
        with 1 show − (4 ∗ u ∗ v) − w² ≤ w ∧ w ≤ 4 ∗ u ∗ v + w² by simp
      qed
    qed
  qed

  show ?thesis
  proof(rule-tac B=?prod in finite-subset)
    show ?S ⊆ ?prod using prod by fast
```

```
    next
      show finite ?prod by simp
    qed
  qed

  have inv1 : involution-on ?S ?f
    by(clarsimp simp: involution-on-def)

  have inv2 : involution-on ?U ?g
    by(fastforce simp: involution-on-def int-distrib power2-diff power2-eq-square)

  have inv3 : involution-on ?T ?h
    by(fastforce simp: involution-on-def)

  have part1 : ∀x∈?S. (x ∈ ?T) = (?f x ∉ ?T)
  proof clarsimp
     fix u v w assume 1: int p = 4 ∗ u ∗ v + w² and 2: (0::int) < v and 3:
(0::int) < u
    have w ≠ 0
    proof(rule w-neq0[rule-format])
      from 1 2 3 show (u, v, w) ∈ ?S by simp
    qed
    thus (0 < w) = (¬ w < 0) by fastforce
  qed

  have part2 : ∀x∈?S. (x ∈ ?U) = (?f x ∉ ?U)
  proof clarsimp
    fix u v w assume 1: int p = 4 ∗ u ∗ v + w² and 2: u > 0 and 3: v > 0
    show (0 < u − v + w) = (¬ w < v − u) (is ?L = ?R)
    proof
      assume ?L with 2 and 3 show ?R by fastforce
    next
      assume ?R hence 4: v − u ≤ w by simp
      show ?L
      proof(rule ccontr)
        assume ¬ ?L with 4 have w = v − u by fastforce
        with 1 have int p = 4 ∗ u ∗ v + (v − u)² by fast
        then have sqr : int p = (v + u)² by(simp add: power2-diff power2-sum)
        with int-prime-not-sqr[OF a] show False ..
      qed
    qed
  qed

  have card-eq : card ?T = card ?U
  proof −
    have 1: 2∗card ?T = card ?S
      by (smt (verit, ccfv-SIG) Int-iff fin-S inv1 involution-sub-card part1 subsetI)
    have 2∗card ?U = card ?S
      by (smt (verit, ccfv-SIG) Int-iff fin-S inv1 involution-sub-card part2 subsetI)
```

```
    with 1 show ?thesis by simp
  qed

  have fixp: fixpoints-on ?U ?g = {((int p - 1) div 4, 1, 1)} (is ?L = ?R)
  proof
    show ?L ⊆ ?R
    proof(clarsimp simp: fixpoints-on-def)
      fix u v assume 1: int p = 4 * u * v + v^2 and 2: 0 < u and 3: 0 < v
      then have 4: int p = v * (4 * u + v)
        by(simp add: power2-eq-square int-distrib)
      have 5: p = nat v * (4 * nat u + nat v)
      proof(rule int-int-eq[THEN iffD1])
        show int p = int (nat v * (4 * nat u + nat v))
          using 2 3 proof(simp add: 4)
        qed
      qed
      note prime-factors[OF a 5]
      then show u = (4 * u * v + v^2 - 1) div 4 ∧ v = 1
      proof
        assume nat v = 1 ∧ 4 * nat u + nat v = p
        with 2 and 3 have v = 1 ∧ 4 * u + v = int p by fastforce
        thus ?thesis by simp
      next
        assume nat v = p ∧ 4 * nat u + nat v = 1
        with 2 have False by fastforce
        thus ?thesis ..
      qed
    qed
  next
    show ?R ⊆ ?L
    proof(clarsimp simp: fixpoints-on-def, rule conjI)
      show int p = 4 * ((int p - 1) div 4) + 1
      proof(subst dvd-mult-div-cancel)
        show 4 dvd int p - 1
        proof(subst mod-eq-dvd-iff[THEN sym])
         show int p mod 4 = 1 mod 4 by(simp add: nat-mod-int[OF b, simplified])
        qed
      next
        show int p = int p - 1 + 1 by simp
      qed
    next
      show 0 < (int p - 1) div 4
        using a b prime-gr4 by fastforce
    qed
  qed

  have cardS1 : odd(card ?T)
  proof(subst card-eq)
    show odd(card ?U)
```

using *add-diff-cancel-right′ fin-S fixp inv2 involution-dom-fixpoints-parity* **by** *fastforce*
qed

have *fixp-ex* : $\exists\, x.\ x \in$ *fixpoints-on ?T ?h*
proof(*rule ccontr*)
assume $\neg$ *?thesis* **hence** *1*: *fixpoints-on ?T ?h* = {} **by** *fast*
note *involution-dom-card-sum*[*OF inv3*, *simplified 1*]
hence *even*(*card ?T*) **by** (*simp add*: *fin-S*)
with *cardS1* **show** *False* **..**
qed

note *fixp-ex* **then have** $\exists\, u\ w.\ u > 0 \wedge w > 0 \wedge int\ p = 4 * u * u + w^2$
by(*clarsimp simp*: *fixpoints-on-def*, *fast*)
then obtain *u w* **where** *c*: $u > 0 \wedge w > 0 \wedge int\ p = (2 * u)^2 + w^2$
by(*fastforce simp*: *power2-eq-square*)
hence $p = (nat(2 * u))^2 + (nat\ w)^2$
by (*smt* (*verit*) *int-nat-eq nat-int nat-int-add of-nat-power*)
thus *?thesis* **by** *fast*

qed

end