Euler’s Partition Theorem

Lukas Bulwahn

September 13, 2023

Abstract

Euler’s Partition Theorem states that the number of partitions with only distinct parts is equal to the number of partitions with only odd parts. The combinatorial proof follows John Harrison’s pre-existing HOL Light formalization [1]. To understand the rough idea of the proof, I read the lecture notes of the MIT course 18.312 on Algebraic Combinatorics [2] by Gregg Musiker. This theorem is the 45th theorem of the Top 100 Theorems list.

Contents

1 Euler’s Partition Theorem 1
  1.1 Preliminaries .......................... 1
    1.1.1 Additions to Divides Theory 1
    1.1.2 Additions to Groups-Big Theory 2
    1.1.3 Additions to Finite-Set Theory 2
  1.2 Binary Encoding of Natural Numbers 2
  1.3 Decomposition of a Number into a Power of Two and an Odd Number 5
  1.4 Partitions With Only Distinct and Only Odd Parts 6
  1.5 Euler’s Partition Theorem 11

1 Euler’s Partition Theorem

theory Euler-Partition
imports
  Main
  Card-Number-Partitions.Number-Partition
begin

1.1 Preliminaries

1.1.1 Additions to Divides Theory

lemma power-div-nat:
assumes $c \leq b$
assumes $a > 0$
shows $(a :: \text{nat}) \sim b \div a \sim c = a \sim (b - c)$
by (metis assms nonzero-mult-cancel-right le-add-diff-inverse2 less-not-refl2 power-add power-not-zero)

1.1.2 Additions to Groups-Big Theory

lemma sum-div:
assumes finite $A$
assumes $\forall a. a \in A \Longrightarrow (b :: \text{'b::euclidean-semiring}) \ dvd f a$
shows $(\sum a \in A. f a) \ div \ b = (\sum a \in A. (f a) \ div \ b)$
using assms
proof (induct)
case insert from this show ?case by auto (subst div-add; auto intro !: dvd-sum)
qed (auto)

lemma sum-mod:
assumes finite $A$
assumes $\forall a. a \in A \Longrightarrow f a \ mod \ b = (\theta :: \text{'b::unique-euclidean-semiring})$
shows $(\sum a \in A. f a) \ mod \ b = 0$
using assms by induct (auto simp add: mod-add-eq [symmetric])

1.1.3 Additions to Finite-Set Theory

lemma finite-exponents:
finite \{ i. 2 ^ i \leq (n :: \text{nat})\}
proof -
  have \{i :: \text{nat}. 2 ^ i \leq n\} \subseteq \{0..n\}
    using dual-order.trans by fastforce
  from finite-subset[OF this] show ?thesis by simp
qed

1.2 Binary Encoding of Natural Numbers

definition bitset :: nat \Rightarrow nat set
where
  \text{bitset } n = \{i. \text{odd (n div (2} \sim i))\}

lemma in-bitset-bound:
  \text{b} \in \text{bitset } n \Longrightarrow 2 ^ \sim b \leq n
unfolding bitset-def using not-less by fastforce

lemma in-bitset-bound-weak:
  b \in \text{bitset } n \Longrightarrow b \leq n
by (meson order.trans in-bitset-bound self-le-ge2-pow[OF order-refl])

lemma finite-bitset:
finite \{\text{bitset } n\}
proof -
have \( \{ n \} \subseteq \ldots \) by (auto dest: in-bitset-bound-weak)

from this show \( \text{thesis using finite-subset by auto} \)

qed

lemma bitset-0:
  \( \text{bitset 0} = \{ \} \)

unfolding bitset-def by auto

lemma bitset-2n: \( \text{bitset} (2 * n) = \text{Suc ' (bitset n)} \)

proof (rule set-eqI)
  fix \( x \)
  show \( (x \in \text{bitset} (2 * n)) = (x \in \text{Suc ' bitset n}) \)
    unfolding bitset-def by (cases \( x \)) auto
  qed

lemma bitset-Suc:
  assumes even \( n \)

  shows \( \text{bitset} (n + 1) = \text{insert 0 (bitset n)} \)

proof (rule set-eqI)
  fix \( x \)
  from assms show \( (x \in \text{bitset} (n + 1)) = (x \in \text{insert 0 (bitset n)}) \)
    unfolding bitset-def by (cases \( x \)) (auto simp add: Divides.div-mult2-eq)
  qed

lemma bitset-2n1:
  \( \text{bitset} (2 * n + 1) = \text{insert 0 (Suc ' (bitset n))} \)

by (subst bitset-Suc) (auto simp add: bitset-2n)

lemma sum-bitset:
  \( (\sum_{i\in\text{bitset n}} 2 ^ i) = n \)

proof (induct rule: nat-bit-induct)
  case zero
  show \( \text{?case by (auto simp add: bitset-0)} \)
  next
  case (even \( n \))
  from this show \( \text{?case by (simp add: bitset-Suc)} \)
  next
  case (odd \( n \))
  have \( (\sum_{i\in\text{bitset (2 * n + 1)} \ldots (2 ^ i)}) = (\sum_{i\in\text{insert 0 (Suc ' bitset n)} \ldots (2 ^ i)}) \)
    by (simp only: bitset-2n1)
  also have \( \ldots = 2 ^ 0 + (\sum_{i\in\text{Suc ' bitset n} \ldots (2 ^ i)}) \)
    by (subst sum.insert) (auto simp add: finite-bitset)
  also have \( \ldots = 2 * n + 1 \)
    using odd by (simp add: sum.reindex sum-distrib-left[symmetric])
  finally show \( \text{?case by simp} \)
  qed

lemma binarysum-div:

have
assumes finite B 
shows \((\sum i \in B. (2::nat) \sim i) \div 2 \sim j = (\sum i \in B. \text{if } i < j \text{ then } 0 \text{ else } 2 \sim (i - j))\)
(is - = (\sum i \in -. ?f i))
proof - 
  have split-B: \(B = \{i \in B. i < j\} \cup \{i \in B. j \leq i\}\) by auto 
  have bound: \((\sum i \mid i \in B \land i < j. (2::nat) \sim i) < 2 \sim j\) 
  proof (rule order.strict-trans1) 
    show \((\sum i \mid i \in B \land i < j. (2::nat) \sim i) \leq (\sum i < j. 2 \sim i)\) by (auto intro: sum mono2)
    show ... < 2 \sim j using sum power2 by (simp add: atLeast0LessThan)
  qed 
from this have zero: \((\sum i \mid i \in B \land i < j. (2::nat) \sim i) \div (2 \sim j) = 0\) by (elim div less)
from assms have mod0: \((\sum i \mid i \in B \land j \leq i. (2::nat) \sim i) \mod 2 \sim j = 0\) 
  by (auto intro!: sum mod simp add: le imp power dvd)
from assms have \((\sum i \in B. (2::nat) \sim i) \div (2 \sim j) = ((\sum i \mid i \in B \land i < j. 2 \sim i) + (\sum i \mid i \in B \land j \leq i. 2 \sim i)) \div 2 \sim j\) 
  by (subst sum union disjoint[symmetric]) (auto simp add: split B[symmetric])
also have ... = \((\sum i \mid i \in B \land j \leq i. 2 \sim i) \div 2 \sim j\) 
  by (simp add: div add1 eq zero mod0)
also have ... = \((\sum i \mid i \in B \land j \leq i. 2 \sim i \div 2 \sim j)\) 
  using assms by (subst sum div) (auto simp add: sum div le imp power dvd)
also have ... = \((\sum i \mid i \in B \land j \leq i. 2 \sim (i - j))\) 
  by (rule sum cong[OF refl]) (auto simp add: power div nat)
also have ... = \((\sum i \in B. ?s i)\) 
  using assms by (subst split B; subst sum union disjoint) auto 
finally show thesis .
qed

lemma odd iff:
assumes finite B
shows odd \((\sum i \in B. \text{if } i < x \text{ then } (0::nat) \text{ else } 2 \sim (i - x)) = (x \in B)\) (is odd \((\sum i \in -. ?s i) = -)\)
proof - 
  from assms have even: even \((\sum i \in B - \{x\}. ?s i)\)
    by (subst dvd sum) auto 
  show thesis 
  proof 
    assume odd \((\sum i \in B. ?s i)\)
    from this even show \(x \in B\) by (cases x \in B) auto 
  next 
    assume \(x \in B\)
    from assms this have \((\sum i \in B. ?s i) = 1 + (\sum i \in B - \{x\}. ?s i)\) 
      by (auto simp add: sum remove)
    from assms this even show odd \((\sum i \in B. ?s i)\) by auto 
  qed 
qed
lemma bitset-sum:
asssumes finite B
showss bitset (\(\sum_{i\in B} 2^i\)) = B
using assms unfolding bitset-def by (simp add: binarysum-div odd-iff)

1.3 Decomposition of a Number into a Power of Two and an Odd Number

function (sequential) index :: nat \(\Rightarrow\) nat
where
index 0 = 0
\mid index n = (if odd n then 0 else Suc (index (n div 2)))
by (pat-completeness) auto

termination
proof
  show wf {\((x::nat, y). \ x < \ y\)} by (simp add: wf)
next
  fix n show (Suc n div 2, Suc n) \(\in\) \{\((x, y). \ x < \ y\)\} by simp
qed

function (sequential) oddpart :: nat \(\Rightarrow\) nat
where
oddpart 0 = 0
\mid oddpart n = (if odd n then n else oddpart (n div 2))
by pat-completeness auto

termination
proof
  show wf {\((x::nat, y). \ x < \ y\)} by (simp add: wf)
next
  fix n show (Suc n div 2, Suc n) \(\in\) \{\((x, y). \ x < \ y\)\} by simp
qed

lemma odd-oddpart:
  odd (oddpart n) \(\longleftrightarrow\) n \(\neq\) 0
by (induct n rule: index.induct) auto

lemma index-oddpart-decomposition:
  n = 2^\(\sim\) (index n) * oddpart n
proof (induct n rule: index.induct)
case (2 n)
  from this show Suc n = 2^\(\sim\) index (Suc n) * oddpart (Suc n)
    by (simp add: mult.assoc)
qed (simp)

lemma oddpart-leq:
  oddpart n \(\leq\) n
by (induct n rule: index.induct) (simp, metis die-le-dividend le-Suc-eq le-trans odd-
part.simps(2))

lemma index-oddpart-unique:
  assumes odd (m :: nat) odd m'
  shows \((2 ^ i \cdot m = 2 ^ i' \cdot m') \iff (i = i' \land m = m')\)
proof (induct i arbitrary: i')
  case 0
    from assms show ?case by auto
next
  case (Suc - i')
    from assms this show ?case by (cases i') auto
qed

lemma index-oddpart:
  assumes odd m
  shows index (2 ^ i * m) = oddpart (2 ^ i * m) = m
using index-oddpart-unique[where i'=i and m=m and m'=oddpart (2 ^ i * m)]
  using index-oddpart-decomposition by force+

1.4 Partitions With Only Distinct and Only Odd Parts

definition odd-of-distinct :: (nat ⇒ nat) ⇒ nat ⇒ nat
where
  odd-of-distinct p = (λi. if odd i then (∑ j \in p \(2 ^ j \cdot i) = 1. 2 ^ j\) else 0)

definition distinct-of-odd :: (nat ⇒ nat) ⇒ nat ⇒ nat
where
  distinct-of-odd p = (λi. if index i \in bitset (p (oddpart i)) then 1 else 0)

lemma odd:
  odd-of-distinct p i ≠ 0 \implies odd i
unfolding odd-of-distinct-def by auto

lemma distinct-distinct-of-odd:
  distinct-of-odd p i ≤ 1
unfolding distinct-of-odd-def by auto

lemma odd-of-distinct:
  assumes odd-of-distinct p i ≠ 0
  assumes \(i. p i ≠ 0 \implies i ≤ n\)
  shows 1 ≤ i \land i ≤ n
proof
  from assms(1) odd have odd i
    by simp
  then show 1 ≤ i
    by (auto elim: oddE)
next
  from assms(1) obtain j where p (2 ^ j * i) > 0
by (auto simp add: odd-of-distinct-def split: if-splits) fastforce

with assms(2) have \( i \leq 2^j \cdot i \leq n \)
  by simp-all
then show \( i \leq n \)
  by (rule order-trans)
qued

lemma distinct-of-odd:
  assumes \( \forall i. \ p \ i \cdot i \leq n \)
  assumes \( \forall i. \ p \ i \neq 0 \implies \text{odd } i \)
  shows \( 1 \leq i \land i \leq n \)
proof
  from assms(3) have \( \text{index } i \in \text{bitset } (p \ (\text{oddpart } i)) \)
    unfolding distinct-of-odd-def by (auto split: if-split-asm)
  have \( i \neq 0 \)
  proof
    assume \( i = 0 \)
    from assms(2) have \( p \ 0 = 0 \)
      by auto
    from index zero this show False (auto simp add: bitset-0)
  qed
  from this show \( 1 \leq i \) by auto
  from \( \{i. \ p \ i = 1\} \subseteq \{..n\} \)
    by auto
  from this have \( \text{finite } \{\, p \ i = 1\,\} \)
    by (simp add: finite-subset)
  have \( \forall x. \ x > 0 \implies p \ (2^j \cdot \text{oddpart } x) = 1 \implies \)
    \( \text{index } (2^j \cdot \text{oddpart } x) \in \text{index } \{\, p \ i = 1 \land \text{oddpart } x = \text{oddpart } i\,\} \)
  proof
    from imageI (auto intro: imageI simp add: index-oddpart odd-oddpart)
  qed
  from this have eq: \( \forall x. \ x > 0 \implies \{\, p \ (2^j \cdot \text{oddpart } x) = 1\,\} = \text{index } \{\, i. \ p \ i = 1 \land \text{oddpart } x = \text{oddpart } i\,\} \)
  unfolding eq by auto
  show \( \theta \)
proof
  fix x
  from assms(1) have p0: p 0 = 0 by auto
  show distinct-of-odd (odd-of-distinct p) x = p x
    proof (cases x > 0)
      case False
      from this p0 show ?thesis
        unfolding odd-of-distinct-def distinct-of-odd-def
        by (auto simp add: odd-oddpart bitset-0)
    next
      case True
      from p0 assms(2)[of x] all-finite[OF True] show ?thesis
        unfolding odd-of-distinct-def distinct-of-odd-def
        by (auto simp add: odd-oddpart bitset-0 bitset-sum index-oddpart-decomposition[ symmetric])
    qed
  qed

lemma sum-distinct-of-odd:
  assumes \( \forall i. p i \neq 0 \Rightarrow 1 \leq i \land i \leq n \)\)
  assumes \( \forall i. p i \leq n \)\)
  assumes \( \forall i. p i \neq 0 \Rightarrow \text{odd } i \)\)
  shows \( (\sum_{i \leq n.} \text{distinct-of-odd } p i \ast i) = (\sum_{i \leq n.} p i \ast i) \)
proof –
  { fix m
    assume odd: odd (m :: nat)
    have finite: finite \{k. 2 \^ k \ast m \leq n \land k \in \text{bitset } (p m)\} by (simp add: finite-bitset)
      have \( (\sum i \mid \exists k. i = 2 \^ k \ast m \land i \leq n. \text{distinct-of-odd } p i \ast i) = \\
      (\sum i \mid \exists k. i = 2 \^ k \ast m \land i \leq n. \text{if index } i \in \text{bitset } (p (\text{oddpart } i)) \text{ then } i \)
      else 0 \)
      unfolding distinct-of-odd-def by (auto intro: sum.cong)
    also have \( = (\sum i \mid \exists k. i = 2 \^ k \ast m \land k \in \text{bitset } (p m) \land i \leq n. ) \)
      using odd by (intro sum.mono-neutral-cong-right) (auto simp add: index-oddpart)
    also have \( = (\sum k \in \text{bitset } (p m). 2 \^ k \ast m) \)
      using odd by (auto intro!: sum.reindex-cong[OF refl inj-onI])
    also have \( = (\sum k \in \text{bitset } (p m). 2 \^ k \ast m) \)
      using assms(2)[of m] finite dual-order.trans in-bitset-bound
      by (fastforce intro!: sum.mono-neutral-cong-right)
    also have \( = (\sum k \in \text{bitset } (p m). 2 \^ k) \ast m \)
      by (subst sum-distrib-right) auto
    also have \( = p m \ast m \)
      by (auto simp add: sum-bitset)
    finally have \( (\sum i \mid \exists k. i = 2 \^ k \ast m \land i \leq n. \text{distinct-of-odd } p i \ast i) = p m \ast m \).
  } note inner-eq = this

8
have set-eq: \{i. 1 \leq i \land i \leq n\} = \bigcup((\lambda m. \{i. \exists k. i = (2 ^ k) * m \land i \leq n\}) \setminus \{m. m \leq n \land odd m\})
proof -
{ 
  fix x
  assume 1 \leq x x \leq n
  from this oddpart-leg[of x] have oddpart x \leq n \land odd (oddpart x) \land (\exists k. 2 ^ index x * oddpart x = 2 ^ k * oddpart x)
    by (auto simp add: odd-oddpart)
  from this have \exists m\leq n. odd m \land (\exists k. x = 2 ^ k * m)
    by (auto simp add: index-oddpart-decomposition[symmetric])
}
from this show ?thesis by (auto simp add: Suc-le1 odd-pos)
qed

let \$S = (\lambda m. \{i. \exists k. i = 2 ^ k * m \land i \leq n\}) \setminus \{m. m \leq n \land odd m\}
have no-overlap: \forall A\in\$S. \forall B\in\$S. A \neq B \longrightarrow A \cap B = {}
  by (auto simp add: index-oddpart-unique)
have inj: inj-on (\lambda m. \{i. (\exists k. i = 2 ^ k * m) \land i \leq n\}) \{m. m \leq n \land odd m\}
  unfolding inj-on-def by auto (force simp add: index-oddpart-unique)
have reindex: \bigwedge F. (\sum i \mid 1 \leq i \land i \leq n. F i) = (\sum m \mid m \leq n \land odd m. (\sum i \mid \exists k. i = 2 ^ k * m \land i \leq n. F i))
  unfolding set-eq by (subst sum.Union-disjoint) (auto simp add: no-overlap intro: sum.reindex-cong[OF inj])
have (\sum i\leq n. distinct-of-odd p i * i) = (\sum i \mid 1 \leq i \land i \leq n. distinct-of-odd p i * i)
  by (auto intro: sum.mono-neutral-right)
also have ... = (\sum m \mid m \leq n \land odd m. \sum i \mid \exists k. i = 2 ^ k * m \land i \leq n. distinct-of-odd p i * i)
  by (simp only: reindex)
also have ... = (\sum i \mid i \leq n \land odd i. p i * i)
  by (rule sum.cong[OF refl[1]; subst inner-eq]) auto
also have ... = (\sum i\leq n. p i * i)
  using assms(3) by (auto intro: sum.mono-neutral-left)
finally show ?thesis .
qed

lemma leq-n:
  assumes \forall i. 0 < p i \longrightarrow 1 \leq i \land i \leq (n::nat)
  assumes (\sum i\leq n. p i * i) = n
  shows p i * i \leq n
proof (rule contr)
  assume \neg p i * i \leq n
  from this have gr-n: p i * i > n by auto
  from this assms(1) have 1 \leq i \land i \leq n by force
  from this have (\sum j\leq n. p j * j) = p i * i + (\sum j \mid j \leq n \land j \neq i. p j * j)
    by (subst sum.insert[symmetric]) (auto intro: sum.cong simp del: sum.insert)
  from this gr-n assms(2) show False by simp
qed
lemma distinct-of-odd-in-distinct-partitions:
assumes \( p \in \{ p. \; p \text{ partitions } n \land (\forall i. \; p \neq 0 \longrightarrow \text{odd } i) \} \)
shows \( \text{distinct-of-odd } p \in \{ p. \; p \text{ partitions } n \land (\forall i. \; p \leq 1) \} \)
proof
have \( \text{distinct-of-odd } p \text{ partitions } n \) 
proof (rule partitionsI)
  fix \( i \) assume \( \text{distinct-of-odd } p \; i \neq 0 \)
  from this \( \text{assms show } 1 \leq i \land i \leq n \) 
  unfolding partitions-def 
  by (rule_tac \( \text{distinct-of-odd} \) \( \text{auto simp add: leq-n} \)) 
next 
  from \( \text{assms} \) show \( (\sum i \leq n. \; \text{distinct-of-odd } p \; i \ast i) = n \) 
  by (subst sum-distinct-of-odd \( \text{auto simp add: distinct-distinct-of-odd leq-n} \)) 
  elim: partitionsE) 
qed 
moreover have \( \forall i. \; \text{distinct-of-odd } p \; i \leq 1 \) 
  by (intro allI \( \text{distinct-distinct-of-odd} \)) 
ultimately show \( \text{distinct-of-odd } p \text{ partitions } n \land (\forall i. \; \text{distinct-of-odd } p \; i \leq 1) \) 
by simp 
qed 

lemma odd-of-distinct-in-odd-partitions:
assumes \( p \in \{ p. \; p \text{ partitions } n \land (\forall i. \; p \leq 1) \} \)
shows \( \text{odd-of-distinct } p \in \{ p. \; p \text{ partitions } n \land (\forall i. \; p \neq 0 \longrightarrow \text{odd } i) \} \)
proof 
  from \( \text{assms} \) have \( \text{distinct} \): \( \bigwedge i. \; p \; i = 0 \lor p \; i = 1 \) 
  using le-imp-less-Suc less-Suc-eq-0-disj \( \text{by fastforce} \) 
  from \( \text{assms}\) have \( \text{set-eq}: \{ x. \; p \; x = 1 \} = \{ x \in \{..n\}. \; p \; x = 1 \} \) 
  unfolding partitions-def \( \text{by auto} \) 
  from \( \text{assms} \) have \( \text{sum}: (\sum i \leq n. \; p \; i \ast i) = n \) 
  unfolding partitions-def \( \text{by auto} \) 
  \{ 
    fix \( i \) 
    assume \( i: \; \text{odd } (i :: \text{nat}) \) 
    have \( 3: \; \text{inj-on index} \{ x. \; p \; x = 1 \land \text{oddpart } x = i \} \) 
      unfolding inj-on-def \( \text{by auto (metis index-oddpart-decomposition)} \) 
      \{ 
        fix \( j \) assume \( p \; (2 \ast j \ast i) = 1 \) 
        from this \( i \) have \( j \in \text{index} \{ x. \; p \; x = 1 \land \text{oddpart } x = i \} \) 
          by (auto simp add: index-oddpart(1, 2) intro!: image-eqI[where \( x=2 \sim j \ast i \)] 
        \} 
    from \( i \) this have \( \{ j, \; p \; (2 \ast j \ast i) = 1 \} = \text{index} \{ x. \; p \; x = 1 \land \text{oddpart } x = i \} \) 
      by (auto simp add: index-oddpart-decomposition[symmetric]) 
    from \( 3 \) this have \( (\sum j \mid p \; (2 \ast j \ast i) = 1. \; 2 \ast j \ast i = (\sum x \mid p \; x = 1 \land \text{oddpart } x = i. \; 2 \sim \text{index } x \ast i) \) 
      by (auto intro: sum.reindex-cong[where \( l = \text{index} \)]) 
    also have \( ... = (\sum x \mid p \; x = 1 \land \text{oddpart } x = i. \; 2 \sim \text{index } x \ast \text{oddpart } x) \) 
      by (auto simp add: sum-distrib-right) 
  \} 
10
also have ... = (∑ x | p x = 1 ∧ oddpart x = i, x)
  by (simp only: index-oddpart-decomposition[symmetric])
also have ... ≤ (∑ x | p x = 1. x)
  using set-eq by (intro sum-mono2) auto
also have ... = (∑ x≤n. p x * x)
  using distinct by (subst set-eq) (force intro: sum.mono-neutral-cong-left)
also have ... = n using sum .
finally have (∑ j | p (2 ^ j * i) = 1. 2 ^ j) * i ≤ n .
}
from this have less-n: ∨ i. odd-of-distinct p i * i ≤ n
  unfolding odd-of-distinct-def by auto
have odd-of-distinct p partitions n
proof (rule partitionsI)
  fix i assume odd-of-distinct p i ≠ 0
  from this assms show 1 ≤ i ∧ i ≤ n
  by (elim CollectE conjE partitionsE odd-of-distinct) auto
next
  have (∑ i≤n. odd-of-distinct p i * i) = (∑ i≤n. distinct-of-odd (odd-of-distinct p) i * i)
    using assms less-n by (subst sum-distinct-of-odd) (auto elim!: partitionsE simp only: odd)
  also have ... = (∑ i≤n. p i * i) using assms
    by (auto elim!: partitionsE simp only:)
  also with assms have ... = n by (auto elim: partitionsE)
  finally show (∑ i≤n. odd-of-distinct p i * i) = n .
qed
moreover have ∀ i. odd-of-distinct p i ≠ 0 −→ odd i
  by (intro allI impI odd)
ultimately show odd-of-distinct p partitions n ∧ (∀ i. odd-of-distinct p i ≠ 0
  −→ odd i) by simp
qed

1.5 Euler’s Partition Theorem

theorem Euler-partition-theorem:
  card { p; p partitions n ∧ (∀ i. p i ≤ 1)} = card { p; p partitions n ∧ (∀ i. p i ≠ 0
  −→ odd i)}
(is card ?distinct-partitions = card ?odd-partitions)
proof (rule card-bij-eq)
from odd-of-distinct-in-odd-partitions show
  odd-of-distinct ′ ?distinct-partitions ⊆ ?odd-partitions by auto
moreover from distinct-of-odd-in-distinct-partitions show
  distinct-of-odd ′ ?odd-partitions ⊆ ′?distinct-partitions by auto
moreover have ∀ p∈?distinct-partitions. distinct-of-odd (odd-of-distinct p) = p
  by auto (subst distinct-odd; auto simp add: partitions-def)
morerover have ∀ p∈?odd-partitions. odd-of-distinct (distinct-of-odd p) = p
  by auto (subst odd-distinct; auto simp add: partitions-def)
ultimately show inj-on odd-of-distinct ′?distinct-partitions
  inj-on distinct-of-odd ′?odd-partitions

11
by (intro bij-betw-imp-inj-an bij-betw-byWitness; auto)+
show finite ?distinct-partitions finite ?odd-partitions
  by (simp add: finite-partitions)+
qed
end

References
