

Euler's Partition Theorem

Lukas Bulwahn

March 17, 2025

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 ≤ b
assumes a > 0
shows (a :: nat) ^ b div a ^ c = a ^ (b - c)
by (metis assms nonzero-mult-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 ∀a. a ∈ A ⇒ (b::'b::euclidean-semiring) dvd f a
  shows (∑ a∈A. f a) div b = (∑ a∈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 ∀a. a ∈ A ⇒ f a mod b = (0::'b::unique-euclidean-semiring)
  shows (∑ a∈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 ≤ (n::nat)}
proof -
  have {i::nat. 2 ^ i ≤ n} ⊆ {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 ⇒ nat set
where
  bitset n = {i. odd (n div (2 ^ i))}

lemma in-bitset-bound:
  b ∈ bitset n ⇒ 2 ^ b ≤ n
  unfolding bitset-def using not-less by fastforce

lemma in-bitset-bound-weak:
  b ∈ bitset n ⇒ b ≤ n
  by (meson order.trans in-bitset-bound self-le-ge2-pow[OF order-refl])

lemma finite-bitset:
  finite (bitset n)
proof -

```

```

have bitset n ⊆ {..n} by (auto dest: in-bitset-bound-weak)
from this show ?thesis using finite-subset by auto
qed

lemma bitset-0:
  bitset 0 = {}
unfolding bitset-def by auto

lemma bitset-2n: bitset (2 * n) = Suc ` (bitset n)
proof (rule set-eqI)
  fix x
  show (x ∈ bitset (2 * n)) = (x ∈ Suc ` bitset n)
    unfolding bitset-def by (cases x) auto
qed

lemma bitset-Suc:
  assumes even n
  shows bitset (n + 1) = insert 0 (bitset n)
proof (rule set-eqI)
  fix x
  from assms show (x ∈ bitset (n + 1)) = (x ∈ insert 0 (bitset n))
    unfolding bitset-def by (cases x) (auto simp add: div-mult2-eq)
qed

lemma bitset-2n1:
  bitset (2 * n + 1) = insert 0 (Suc ` (bitset n))
by (subst bitset-Suc) (auto simp add: bitset-2n)

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

lemma binarysum-div:

```

```

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

lemma odd-iff:
assumes finite B
shows odd ( $\sum_{i \in B} \text{if } i < x \text{ then } (0::nat) \text{ else } 2^{i-x}$ ) = ( $x \in B$ ) (is odd ( $\sum_{i \in -.} ?s i$ ) = -)
proof -
  from assms have even: even ( $\sum_{i \in B} \text{if } i \neq x \text{ then } 1 \text{ else } 0$ ) by (subst dvd-sum) auto
  show ?thesis
proof
  assume odd ( $\sum_{i \in B} ?s i$ )
  from this even show x ∈ B by (cases x ∈ B) auto
next
  assume x ∈ B
  from assms this have ( $\sum_{i \in B} ?s i$ ) = 1 + ( $\sum_{i \in B \setminus \{x\}} ?s i$ )
  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:
  assumes finite B
  shows 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
  | 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
  | 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 (oddpark n)  $\longleftrightarrow$  n  $\neq$  0
  by (induct n rule: index.induct) auto

lemma index-oddpark-decomposition:
  n =  $2^{\lceil \text{index } n \rceil} * \text{oddpark } n$ 
proof (induct n rule: index.induct)
  case (2 n)
  from this show Suc n =  $2^{\lceil \text{index } (\text{Suc } n) \rceil} * \text{oddpark } (\text{Suc } n)$ 
    by (simp add: mult.assoc)
  qed (simp)

lemma oddpart-leq:
  oddpart n  $\leq$  n
  by (induct n rule: index.induct) (simp, metis div-le-dividend le-Suc-eq le-trans odd-

```

```

part.simps(2))

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

```

1.4 Partitions With Only Distinct and Only Odd Parts

```

definition odd-of-distinct :: (nat  $\Rightarrow$  nat)  $\Rightarrow$  nat  $\Rightarrow$  nat
where
  odd-of-distinct p = ( $\lambda$ i. if odd i then ( $\sum$ j | p (2 ^ j * i) = 1. 2 ^ j) else 0)

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

lemma odd:
  odd-of-distinct p i  $\neq$  0  $\Longrightarrow$  odd i
  unfolding odd-of-distinct-def by auto

lemma distinct-distinct-of-odd:
  distinct-of-odd p i  $\leq$  1
  unfolding distinct-of-odd-def by auto

lemma odd-of-distinct:
  assumes odd-of-distinct p i  $\neq$  0
  assumes  $\bigwedge$ i. p i  $\neq$  0  $\Longrightarrow$  i  $\leq$  n
  shows 1  $\leq$  i  $\wedge$  i  $\leq$  n
proof
  from assms(1) odd have odd i
  by simp
  then show 1  $\leq$  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  $i \leq 2^j * i$   $2^j * i \leq n$ 
  by simp-all
  then show  $i \leq n$ 
    by (rule order-trans)
qed

lemma distinct-of-odd:
assumes  $\bigwedge i. p(i) \neq 0 \Rightarrow \text{odd } i$ 
assumes distinct-of-odd  $p(i) \neq 0$ 
shows  $1 \leq i \wedge i \leq n$ 
proof
  from assms(3) have index: 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 zero:  $i = 0$ 
    from assms(2) have  $p(0) = 0$  by auto
    from index zero this show False by (auto simp add: bitset-0)
  qed
  from this show  $1 \leq i$  by auto
  from assms(1) have leq-n:  $p(\text{oddpart } i) * \text{oddpart } i \leq n$  by auto
  from index have  $2^j * \text{index } i \leq p(\text{oddpart } i)$  by (rule in-bitset-bound)
  from this leq-n show  $i \leq n$ 
    by (subst index-oddpart-decomposition[of i]) (meson dual-order.trans eq-imp-le
mult-le-mono)
qed

lemma odd-distinct:
assumes  $\bigwedge i. p(i) \neq 0 \Rightarrow \text{odd } i$ 
shows odd-of-distinct (distinct-of-odd  $p) = p$ 
using assms unfolding odd-of-distinct-def distinct-of-odd-def
by (auto simp add: fun-eq-iff index-oddpart sum-bitset)

lemma distinct-odd:
assumes  $\bigwedge i. p(i) \neq 0 \Rightarrow 1 \leq i \wedge i \leq n \wedge \bigwedge i. p(i) \leq 1$ 
shows distinct-of-odd (odd-of-distinct  $p) = p$ 
proof -
  from assms have  $\{i. p(i) = 1\} \subseteq \{..n\}$  by auto
  from this have finite: finite  $\{i. p(i) = 1\}$  by (simp add: finite-subset)
  have  $\bigwedge x. x > 0 \Rightarrow p(2^j * \text{oddpart } x) = 1 \Rightarrow$ 
    index  $(2^j * \text{oddpart } x) \in \text{index} \{i. p(i) = 1 \wedge \text{oddpart } x = \text{oddpart } i\}$ 
    by (rule imageI) (auto intro: imageI simp add: index-oddpart odd-oddpart)
  from this have eq:  $\bigwedge x. x > 0 \Rightarrow \{j. p(2^j * \text{oddpart } x) = 1\} = \text{index} \{i.$ 
 $p(i) = 1 \wedge \text{oddpart } x = \text{oddpart } i\}$ 
    by (auto simp add: index-oddpart odd-oddpart index-oddpart-decomposition[symmetric])
  from finite have all-finite:  $\bigwedge x. x > 0 \Rightarrow \text{finite} \{j. p(2^j * \text{oddpart } x) = 1\}$ 
    unfolding eq by auto
  show ?thesis

```

```

proof
  fix  $x$ 
  from  $\text{assms}(1)$  have  $p 0 = 0$  by auto
  show distinct-of-odd (odd-of-distinct  $p$ )  $x = p x$ 
  proof (cases  $x > 0$ )
    case False
    from this  $p 0$  show ?thesis
      unfolding odd-of-distinct-def distinct-of-odd-def
      by (auto simp add: odd-oddpart bitset-0)
    next
    case True
    from  $p 0$   $\text{assms}(2)[\text{of } x]$  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
qed

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

```

```

have set-eq: {i. 1 ≤ i ∧ i ≤ n} = ∪((λm. {i. ∃k. i = (2 ^ k) * m ∧ i ≤ n}) ` {m. m ≤ n ∧ odd m})
  proof -
    {
      fix x
      assume 1 ≤ x x ≤ n
      from this oddpart-leq[of x] have oddpart x ≤ n ∧ odd (oddpart x) ∧ (∃k. 2 ^ index x * oddpart x = 2 ^ k * oddpart x)
        by (auto simp add: odd-oddpart)
      from this have ∃m≤n. odd m ∧ (∃k. x = 2 ^ k * m)
        by (auto simp add: index-oddpart-decomposition[symmetric])
    }
    from this show ?thesis by (auto simp add: Suc-leI odd-pos)
  qed
  let ?S = (λm. {i. ∃k. i = 2 ^ k * m ∧ i ≤ n}) ` {m. m ≤ n ∧ odd m}
  have no-overlap: ∀A∈?S. ∀B∈?S. A ≠ B → A ∩ B = {}
    by (auto simp add: index-oddpart-unique)
  have inj: inj-on (λm. {i. (∃k. i = 2 ^ k * m) ∧ i ≤ n}) {m. m ≤ n ∧ odd m}
    unfolding inj-on-def by auto (force simp add: index-oddpart-unique)
  have reindex: ∏F. (∑i | 1 ≤ i ∧ i ≤ n. F i) = (∑m | m ≤ n ∧ odd m. (∑i | ∃k. i = 2 ^ k * m ∧ i ≤ n. F i))
    unfolding set-eq by (subst sum.Union-disjoint) (auto simp add: no-overlap
      intro: sum.reindex-cong[OF inj])
  have (∑i≤n. distinct-of-odd p i * i) = (∑i | 1 ≤ i ∧ i ≤ n. distinct-of-odd p i * i)
    by (auto intro: sum.mono-neutral-right)
  also have ... = (∑m | m ≤ n ∧ odd m. ∑i | ∃k. i = 2 ^ k * m ∧ i ≤ n. distinct-of-odd p i * i)
    by (simp only: reindex)
  also have ... = (∑i | i ≤ n ∧ odd i. p i * i)
    by (rule sum.cong[OF refl]; subst inner-eq) auto
  also have ... = (∑i≤n. p i * i)
    using assms(3) by (auto intro: sum.mono-neutral-left)
  finally show ?thesis .
qed

lemma leq-n:
  assumes ∀i. 0 < p i → 1 ≤ i ∧ i ≤ (n::nat)
  assumes (∑i≤n. p i * i) = n
  shows p i * i ≤ n
proof (rule ccontr)
  assume ¬ p i * i ≤ n
  from this have gr-n: p i * i > n by auto
  from this assms(1) have 1 ≤ i ∧ i ≤ n by force
  from this have (∑j≤n. p j * j) = p i * i + (∑j | j ≤ n ∧ j ≠ 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 ∈ {p. p partitions n ∧ (∀ i. p i ≠ 0 → odd i)}
shows distinct-of-odd p ∈ {p. p partitions n ∧ (∀ i. p i ≤ 1)}
proof
have distinct-of-odd p partitions n
proof (rule partitionsI)
fix i assume distinct-of-odd p i ≠ 0
from this assms show 1 ≤ i ∧ i ≤ n
unfolding partitions-def
by (rule-tac distinct-of-odd) (auto simp add: leq-n)
next
from assms show (∑ i≤n. distinct-of-odd p i * i) = n
by (subst sum-distinct-of-odd) (auto simp add: distinct-distinct-of-odd leq-n
elim: partitionsE)
qed
moreover have ∀ i. distinct-of-odd p i ≤ 1
by (intro allI distinct-distinct-of-odd)
ultimately show distinct-of-odd p partitions n ∧ (∀ i. distinct-of-odd p i ≤ 1)
by simp
qed

lemma odd-of-distinct-in-odd-partitions:
assumes p ∈ {p. p partitions n ∧ (∀ i. p i ≤ 1)}
shows odd-of-distinct p ∈ {p. p partitions n ∧ (∀ i. p i ≠ 0 → odd i)}
proof
from assms have distinct: ∏ i. p i = 0 ∨ p i = 1
using le-imp-less-Suc less-Suc-eq-0-disj by fastforce
from assms have set-eq: {x. p x = 1} = {x ∈ {..n}. p x = 1}
unfolding partitions-def by auto
from assms have sum: (∑ i≤n. p i * i) = n
unfolding partitions-def by auto
{
fix i
assume i: odd (i :: nat)
have 3: inj-on index {x. p x = 1 ∧ oddpart x = i}
unfolding inj-on-def by auto (metis index-oddpart-decomposition)
{
fix j assume p (2 ^ j * i) = 1
from this i have j ∈ index ` {x. p x = 1 ∧ oddpart x = i}
by (auto simp add: index-oddpart(1, 2) intro!: image-eqI[where x=2 ^ j *
i])
}
from i this have {j. p (2 ^ j * i) = 1} = index ` {x. p x = 1 ∧ oddpart x = i}
by (auto simp add: index-oddpart-decomposition[symmetric])
from 3 this have (∑ j | p (2 ^ j * i) = 1. 2 ^ j) * i = (∑ x | p x = 1 ∧
oddpart x = i. 2 ^ index x) * i
by (auto intro: sum.reindex-cong[where l = index])
also have ... = (∑ x | p x = 1 ∧ oddpart x = i. 2 ^ index x * oddpart x)
by (auto simp add: sum-distrib-right)

```

```

also have ... = ( $\sum x \mid p x = 1 \wedge \text{oddpart } x = i. x$ )
  by (simp only: index-oddpart-decomposition[symmetric])
also have ...  $\leq (\sum x \mid p x = 1. x)$ 
  using set-eq by (intro sum-mono2) auto
also have ... = ( $\sum_{x \leq n.} p x * x$ )
  using distinct by (subst set-eq) (force intro!: sum.mono-neutral-cong-left)
also have ... = n using sum .
finally have ( $\sum j \mid p (2^j * i) = 1. 2^j * i \leq n$  .
}
from this have less-n:  $\bigwedge i. \text{odd-of-distinct } p i * i \leq n$ 
  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  $\neq 0$ 
  from this assms show  $1 \leq i \wedge i \leq n$ 
    by (elim CollectE conjE partitionsE odd-of-distinct) auto
next
  have ( $\sum_{i \leq n.} \text{odd-of-distinct } p i * i = (\sum_{i \leq n.} \text{distinct-of-odd } (\text{odd-of-distinct } p) i * i)$ 
    using assms less-n by (subst sum-distinct-of-odd) (auto elim!: partitionsE odd-of-distinct simp only: odd)
  also have ... = ( $\sum_{i \leq n.} p i * i$ ) using assms
    by (auto elim!: partitionsE simp only:) (subst distinct-odd, auto)
  also with assms have ... = n by (auto elim: partitionsE)
  finally show ( $\sum_{i \leq n.} \text{odd-of-distinct } p i * i = n$  .
qed
moreover have  $\forall i. \text{odd-of-distinct } p i \neq 0 \longrightarrow \text{odd } i$ 
  by (intro allI impI odd)
ultimately show odd-of-distinct p partitions n  $\wedge (\forall i. \text{odd-of-distinct } p i \neq 0 \longrightarrow \text{odd } i)$  by simp
qed

```

1.5 Euler's Partition Theorem

```

theorem Euler-partition-theorem:
  card {p. p partitions n  $\wedge (\forall i. p i \leq 1)}$ } = card {p. p partitions n  $\wedge (\forall i. p i \neq 0 \longrightarrow \text{odd } i)}$ }
  (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  $\subseteq$  ?odd-partitions by auto
  moreover from distinct-of-odd-in-distinct-partitions show
    distinct-of-odd `?odd-partitions  $\subseteq$  ?distinct-partitions by auto
  moreover have  $\forall p \in ?distinct-partitions. \text{distinct-of-odd } (\text{odd-of-distinct } p) = p$ 
    by auto (subst distinct-odd; auto simp add: partitions-def)
  moreover have  $\forall p \in ?odd-partitions. \text{odd-of-distinct } (\text{distinct-of-odd } p) = p$ 
    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

```

```

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

end

```

References

- [1] J. Harrison. Euler's partition theorem and other elementary partition theorems. <https://github.com/jrh13/hol-light/blob/master/100/euler.ml>.
- [2] G. Musiker. Course 18.312: Algebraic combinatorics, 2009. http://ocw.mit.edu/courses/mathematics/18-312-algebraic-combinatorics-spring-2009/readings-and-lecture-notes/MIT18_312S09_lec10_Patitio.pdf.