Euler's Partition Theorem

Lukas Bulwahn

October 11, 2017

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

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 $(a :: nat) \hat{} b \; div \; a \hat{} c = a \hat{} (b - c)$
by (*metis assms nonzero-mult-div-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 $\bigwedge a.\ a \in A \Longrightarrow (b::'b::semiring\text{-}div) \; dvd \; f \; a$
  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 $\bigwedge a.\ a \in A \Longrightarrow f\ a \; mod \; b = (0::'b::\{semiring\text{-}div\})$
  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 Set-Interval Theory

lemma *geometric-sum-2nat*:
  $(\sum i{<}n.\ (2::nat) \hat{} i) = (2 \hat{} n - 1)$
by (*induct n*) *auto*

1.1.4 Additions to Nat Theory or Power Theory

lemma *n-leq-2-pow-n*:
  $n \leq 2 \hat{} n$
proof (*induct n*)
  case (*Suc n*)
  from *this* **have** *approx*: $2 * n \leq 2 \hat{} Suc\ n$ **by** *auto*
  show *?case*
  proof (*cases n*)
    case *0*
    from *this* **show** *?thesis* **by** *simp*
  next
    case *Suc*
    from *this* **show** *?thesis* **by** (*intro le-trans*[*OF - approx*]) *simp*
  qed
qed (*simp*)

1.1.5 Additions to Finite-Set Theory

lemma *finite-exponents*:
  finite $\{i.\ 2\ \hat{}\ i \leq (n::nat)\}$
proof −
  have $\{i::nat.\ 2\ \hat{}\ i \leq n\} \subseteq \{..n\}$
    using *dual-order.trans n-leq-2-pow-n* **by** *auto*
  from *this* **show** *?thesis* **by** (*simp add*: *finite-subset*)
qed

1.2 Binary Encoding of Natural Numbers

definition *bitset* :: *nat* ⇒ *nat set*
where
  bitset n = $\{i.\ odd\ (n\ div\ (2\ \hat{}\ i))\}$

lemma *in-bitset-bound*:
  $b \in bitset\ n \Longrightarrow 2\ \hat{}\ b \leq n$
unfolding *bitset-def* **using** *not-less* **by** *fastforce*

lemma *in-bitset-bound-weak*:
  $b \in bitset\ n \Longrightarrow b \leq n$
by (*auto dest*: *in-bitset-bound intro*: *le-trans n-leq-2-pow-n*)

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 *binary-induct* [*case-names zero even odd*]:
  assumes *P* (*0* :: *nat*)
  assumes $\bigwedge n.\ P\ n \Longrightarrow P\ (2 * n)$
  assumes $\bigwedge n.\ P\ n \Longrightarrow P\ (2 * n + 1)$
  shows $\bigwedge n.\ P\ n$
using *assms parity-induct* **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: Divides.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: binary-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 .
qed

lemma binarysum-div:
  assumes finite B
  shows (∑ i∈B. (2::nat) ^ i) div 2 ^ j = (∑ i∈B. if i < j then 0 else 2 ^ (i −
j))
  (is - = (∑ i∈-. ?f i))
proof −
  have split-B: B = {i∈B. i < j} ∪ {i∈B. j ≤ i} by auto
  have bound: (∑ i | i ∈ B ∧ i < j. (2::nat) ^ i) < 2 ^ j
  proof (rule order.strict-trans1)
    show (∑ i | i ∈ B ∧ i < j. (2::nat) ^ i) ≤ (∑ i<j. 2 ^ i) by (auto intro:
sum-mono2)
    show ... < 2 ^ j by (simp add: geometric-sum-2nat)
  qed
  from this have zero: (∑ i | i ∈ B ∧ i < j. (2::nat) ^ i) div (2 ^ j) = 0 by
(elim div-less)
  from assms have mod0: (∑ i | i ∈ B ∧ j ≤ i. (2::nat) ^ i) mod 2 ^ j = 0
    by (auto intro!: sum-mod simp add: le-imp-power-dvd)
```

from *assms* **have** $(\sum i{\in}B.\ (2{::}nat)\ \hat{}\ i)\ div\ (2\ \hat{}\ j) = ((\sum i \mid i \in B \wedge i < j.\ 2\ \hat{}\ i) + (\sum i \mid i \in B \wedge j \leq i.\ 2\ \hat{}\ i))\ div\ 2\ \hat{}\ j$
by (*subst sum.union-disjoint*[*symmetric*]) (*auto simp add*: *split-B*[*symmetric*])
also have $... = (\sum i \mid i \in B \wedge j \leq i.\ 2\ \hat{}\ i)\ div\ 2\ \hat{}\ j$
by (*simp add*: *div-add1-eq zero mod0*)
also have $... = (\sum i \mid i \in B \wedge j \leq i.\ 2\ \hat{}\ i\ div\ 2\ \hat{}\ j)$
using *assms* **by** (*subst sum-div*) (*auto simp add*: *sum-div le-imp-power-dvd*)
also have $... = (\sum i \mid i \in B \wedge j \leq i.\ 2\ \hat{}\ (i - j))$
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 $odd\ (\sum i{\in}B.\ if\ i < x\ then\ (0{::}nat)\ else\ 2\ \hat{}\ (i - x)) = (x \in B)$ (**is** $odd\ (\sum i{\in}\text{-}.\ ?s\ i) = \text{-}$)
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*:
assumes *finite B*
shows $bitset\ (\sum i{\in}B.\ 2\ \hat{}\ i) = B$
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) ∈ {(x, y). x < y} by simp
qed

function (sequential) oddpart :: nat ⇒ 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) ∈ {(x, y). x < y} by simp
qed

lemma odd-oddpart:
  odd (oddpart n) ⟷ n ≠ 0
by (induct n rule: index.induct) auto

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

lemma oddpart-leq:
  oddpart n ≤ 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') ⟷ (i = i' ∧ 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** $m'{=}oddpart\ (2\ \hat{}\ i * m)$ **and** $i'{=}index\ (2\ \hat{}\ i * m)$]
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 = $(\lambda i.\ if\ odd\ i\ then\ (\sum j \mid p\ (2\ \hat{}\ j * i) = 1.\ 2\ \hat{}\ j)\ else\ 0)$

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*:
$odd\text{-}of\text{-}distinct\ p\ i \neq 0 \Longrightarrow odd\ i$
unfolding *odd-of-distinct-def* **by** *auto*

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

lemma *odd-of-distinct*:
assumes $odd\text{-}of\text{-}distinct\ p\ i \neq 0$
assumes $\bigwedge i.\ p\ i \neq 0 \Longrightarrow i \leq n$
shows $1 \leq i \wedge i \leq n$
proof
from *assms*(*1*) **show** $1 \leq i$ **by** (*metis One-nat-def Suc-leI even-zero gr0I odd*)
next
from *assms*(*1*) **obtain** *j* **where** $p\ (2\ \hat{}\ j * i) \neq 0$
unfolding *odd-of-distinct-def* **by** (*auto split*: *if-split-asm*) *fastforce*
from *assms*(*2*)[*OF this*] **show** $i \leq n$
by (*metis div-le-dividend nonzero-mult-div-cancel-left le-trans power-not-zero zero-not-eq-two*)
qed

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

lemma odd-distinct:
  assumes ⋀i. p i ≠ 0 ⟹ 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 ⋀i. p i ≠ 0 ⟹ 1 ≤ i ∧ i ≤ n ⋀i. p i ≤ 1
  shows distinct-of-odd (odd-of-distinct p) = p
proof −
  from assms have {i. p i = 1} ⊆ {..n} by auto
  from this have finite: finite {i. p i = 1} by (simp add: finite-subset)
  have ⋀x j. x > 0 ⟹ p (2 ^ j * oddpart x) = 1 ⟹
    index (2 ^ j * oddpart x) ∈ index ‘ {i. p i = 1 ∧ oddpart x = oddpart i}
    by (rule imageI) (auto intro: imageI simp add: index-oddpart odd-oddpart)
  from this have eq: ⋀x. x > 0 ⟹ {j. p (2 ^ j * oddpart x) = 1} = index ‘ {i.
p i = 1 ∧ oddpart x = oddpart i}
   by (auto simp add: index-oddpart odd-oddpart index-oddpart-decomposition[symmetric])
  from finite have all-finite: ⋀x. x > 0 ⟹ finite {j. p (2 ^ j * oddpart x) = 1}
    unfolding eq by auto
  show ?thesis
  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
qed

lemma sum-distinct-of-odd:
  assumes ⋀i. p i ≠ 0 ⟹ 1 ≤ i ∧ i ≤ n
```

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

have *set-eq*: $\{i.\ 1 \leq i \wedge i \leq n\} = \bigcup((\lambda m.\ \{i.\ \exists k.\ i = (2\ \hat{}\ k) * m \wedge i \leq n\})\ `\ \{m.\ m \leq n \wedge odd\ m\})$
proof −
{
fix x
assume $1 \leq x\ x \leq n$
from *this oddpart-leq*[*of x*] **have** $oddpart\ x \leq n \wedge odd\ (oddpart\ x) \wedge (\exists k.\ 2\ \hat{}\ index\ x * oddpart\ x = 2\ \hat{}\ k * oddpart\ x)$
by (*auto simp add*: *odd-oddpart*)
from *this* **have** $\exists m{\leq}n.\ odd\ m \wedge (\exists k.\ x = 2\ \hat{}\ k * m)$
by (*auto simp add*: *index-oddpart-decomposition*[*symmetric*])
}
from *this* **show** *?thesis* **by** (*auto simp add*: *Suc-leI odd-pos*)
qed
let $?S = (\lambda m.\ \{i.\ \exists k.\ i = 2\ \hat{}\ k * m \wedge i \leq n\})\ `\ \{m.\ m \leq n \wedge odd\ m\}$
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*: $inj\text{-}on\ (\lambda m.\ \{i.\ (\exists k.\ i = 2\ \hat{}\ k * m) \wedge i \leq n\})\ \{m.\ m \leq n \wedge odd\ m\}$
unfolding *inj-on-def* **by** *auto* (*force simp add*: *index-oddpart-unique*)
have *reindex*: $\bigwedge F.\ (\sum i \mid 1 \leq i \wedge i \leq n.\ F\ i) = (\sum m \mid m \leq n \wedge odd\ m.\ (\sum i \mid \exists k.\ i = 2\ \hat{}\ k * m \wedge i \leq n.\ F\ i))$

unfolding *set-eq* **by** (*subst sum.Union-disjoint*) (*auto simp add*: *no-overlap intro*: *sum.reindex-cong*[*OF inj*])
have $(\sum i{\le}n.\ \mathit{distinct\text{-}of\text{-}odd}\ p\ i * i) = (\sum i \mid 1 \le i \wedge i \le n.\ \mathit{distinct\text{-}of\text{-}odd}\ p\ i * i)$
by (*auto intro*: *sum.mono-neutral-right*)
also have ... $= (\sum m \mid m \le n \wedge \mathit{odd}\ m.\ \sum i \mid \exists k.\ i = 2\ \hat{}\ k * m \wedge i \le n.\ \mathit{distinct\text{-}of\text{-}odd}\ p\ i * i)$
by (*simp only*: *reindex*)
also have ... $= (\sum i \mid i \le n \wedge \mathit{odd}\ i.\ p\ i * i)$
by (*rule sum.cong*[*OF refl*]; *subst inner-eq*) *auto*
also have ... $= (\sum i{\le}n.\ p\ i * i)$
using *assms*(*3*) **by** (*auto intro*: *sum.mono-neutral-left*)
finally show *?thesis* **.**
qed

lemma *leq-n*:
assumes $\forall i.\ 0 < p\ i \longrightarrow 1 \le i \wedge i \le (n::nat)$
assumes $(\sum i{\le}n.\ p\ i * i) = n$
shows $p\ i * i \le n$
proof (*rule ccontr*)
assume $\neg\ p\ i * i \le n$
from *this* **have** *gr-n*: $p\ i * i > n$ **by** *auto*
from *this assms*(*1*) **have** $1 \le i \wedge i \le n$ **by** *force*
from *this* **have** $(\sum j{\le}n.\ p\ j * j) = p\ i * i + (\sum j \mid j \le n \wedge j \neq i.\ p\ j * j)$
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 $p \in \{p.\ p\ \mathit{partitions}\ n \wedge (\forall i.\ p\ i \neq 0 \longrightarrow \mathit{odd}\ i)\}$
shows $\mathit{distinct\text{-}of\text{-}odd}\ p \in \{p.\ p\ \mathit{partitions}\ n \wedge (\forall i.\ p\ i \le 1)\}$
proof
have *distinct-of-odd p partitions n*
proof (*rule partitionsI*)
fix *i* **assume** $\mathit{distinct\text{-}of\text{-}odd}\ p\ i \neq 0$
from *this assms* **show** $1 \le i \wedge i \le n$
unfolding *partitions-def*
by (*rule-tac distinct-of-odd*) (*auto simp add*: *leq-n*)
next
from *assms* **show** $(\sum i{\le}n.\ \mathit{distinct\text{-}of\text{-}odd}\ p\ i * i) = n$
by (*subst sum-distinct-of-odd*) (*auto simp add*: *distinct-distinct-of-odd leq-n elim*: *partitionsE*)
qed
moreover have $\forall i.\ \mathit{distinct\text{-}of\text{-}odd}\ p\ i \le 1$
by (*intro allI distinct-distinct-of-odd*)
ultimately show $\mathit{distinct\text{-}of\text{-}odd}\ p\ \mathit{partitions}\ n \wedge (\forall i.\ \mathit{distinct\text{-}of\text{-}odd}\ p\ i \le 1)$
by *simp*
qed

lemma *odd-of-distinct-in-odd-partitions*:
assumes $p \in \{p.\ p \text{ partitions } n \wedge (\forall i.\ p\ i \leq 1)\}$
shows *odd-of-distinct* $p \in \{p.\ p \text{ partitions } n \wedge (\forall i.\ p\ i \neq 0 \longrightarrow \text{odd } i)\}$
proof
from *assms* **have** *distinct*: $\bigwedge i.\ p\ i = 0 \vee p\ i = 1$
using *le-imp-less-Suc less-Suc-eq-0-disj* **by** *fastforce*
from *assms* **have** *set-eq*: $\{x.\ p\ x = 1\} = \{x \in \{..n\}.\ p\ x = 1\}$
unfolding *partitions-def* **by** *auto*
from *assms* **have** *sum*: $(\sum i{\leq}n.\ p\ i * i) = n$
unfolding *partitions-def* **by** *auto*
{
fix *i*
assume *i*: *odd* (*i* :: *nat*)
have *3*: *inj-on index* $\{x.\ p\ x = 1 \wedge \text{oddpart } x = i\}$
unfolding *inj-on-def* **by** *auto* (*metis index-oddpart-decomposition*)
{
fix *j* **assume** $p\ (2\ \hat{}\ j * i) = 1$
from *this i* **have** $j \in \text{index}\ `\ \{x.\ p\ x = 1 \wedge \text{oddpart } x = i\}$
by (*auto simp add*: *index-oddpart*(*1*, *2*) *intro*!: *image-eqI*[**where** *x*=*2* ^ *j* * *i*])
}
from *i this* **have** $\{j.\ p\ (2\ \hat{}\ j * i) = 1\} = \text{index}\ `\ \{x.\ p\ x = 1 \wedge \text{oddpart } x = i\}$
by (*auto simp add*: *index-oddpart-decomposition*[*symmetric*])
from *3 this* **have** $(\sum j \mid p\ (2\ \hat{}\ j * i) = 1.\ 2\ \hat{}\ j) * i = (\sum x \mid p\ x = 1 \wedge \text{oddpart } x = i.\ 2\ \hat{}\ \text{index } x) * i$
by (*auto intro*: *sum.reindex-cong*[**where** *l* = *index*])
also have ... $= (\sum x \mid p\ x = 1 \wedge \text{oddpart } x = i.\ 2\ \hat{}\ \text{index } x * \text{oddpart } x)$
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 $(\sum j \mid p\ (2\ \hat{}\ j * i) = 1.\ 2\ \hat{}\ j) * i \leq n$ **.**
}
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 $(\sum i{\leq}n.$ *odd-of-distinct p i* $*$ *i*$) = n$ **.**
qed
moreover have $\forall\, i.$ *odd-of-distinct p i* $\neq$ *0* $\longrightarrow$ *odd i*
by (*intro allI impI odd*)
ultimately show *odd-of-distinct p partitions n* $\wedge$ $(\forall\, i.$ *odd-of-distinct p i* $\neq$ *0* $\longrightarrow$ *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* $\longrightarrow$ *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*. *distinct-of-odd* (*odd-of-distinct p*) = *p*
by *auto* (*subst distinct-odd*; *auto simp add*: *partitions-def*)
moreover have $\forall\, p{\in}$*?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
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.