Cardinality of Set Partitions

Lukas Bulwahn

September 13, 2023

Abstract

The theory's main theorem states that the cardinality of set partitions of size k on a carrier set of size n is expressed by Stirling numbers of the second kind. In Isabelle, Stirling numbers of the second kind are defined in the AFP entry 'Discrete Summation' [1] through their well-known recurrence relation. The main theorem relates them to the alternative definition as cardinality of set partitions. The proof follows the simple and short explanation in Richard P. Stanley's 'Enumerative Combinatorics: Volume 1' [2] and Wikipedia [3], and unravels the full details and implicit reasoning steps of these explanations.

Contents

1 Set Partitions

theory *Set-Partition*
imports
HOL−Library.Disjoint-Sets

$HOL\text{−}Library.FuncSet$
begin

1.1 Useful Additions to Main Theories

lemma *set-eqI′*:
  assumes $\bigwedge x.\ x \in A \Longrightarrow x \in B$
  assumes $\bigwedge x.\ x \in B \Longrightarrow x \in A$
  shows $A = B$
using *assms* **by** *auto*

lemma *comp-image*:
  $((`)\ f \circ (`)\ g) = (`)\ (f\ o\ g)$
by *rule auto*

1.2 Introduction and Elimination Rules

The definition of *partition-on* is in $HOL\text{−}Library.Disjoint\text{-}Sets$.

lemma *partition-onI*:
  assumes $\bigwedge p.\ p \in P \Longrightarrow p \neq \{\}$
  assumes $\bigcup P = A$
  assumes $\bigwedge p\ p'.\ p \in P \Longrightarrow p' \in P \Longrightarrow p \neq p' \Longrightarrow p \cap p' = \{\}$
  shows *partition-on* $A\ P$
using *assms* **unfolding** *partition-on-def disjoint-def* **by** *blast*

lemma *partition-onE*:
  assumes *partition-on* $A\ P$
  obtains $\bigwedge p.\ p \in P \Longrightarrow p \neq \{\}$
    $\bigcup P = A$
    $\bigwedge p\ p'.\ p \in P \Longrightarrow p' \in P \Longrightarrow p \neq p' \Longrightarrow p \cap p' = \{\}$
using *assms* **unfolding** *partition-on-def disjoint-def* **by** *blast*

1.3 Basic Facts on Set Partitions

lemma *partition-onD4*: *partition-on* $A\ P \Longrightarrow p \in P \Longrightarrow q \in P \Longrightarrow x \in p \Longrightarrow x \in q \Longrightarrow p = q$
  by (*auto simp*: *partition-on-def disjoint-def*)

lemma *partition-subset-imp-notin*:
  assumes *partition-on* $A\ P\ X \in P$
  assumes $X' \subset X$
  shows $X' \notin P$
proof
  assume $X' \in P$
  from ‹$X' \in P$› ‹*partition-on* $A\ P$› **have** $X' \neq \{\}$
    using *partition-onD3* **by** *blast*
  moreover from ‹$X' \in P$› ‹$X \in P$› ‹*partition-on* $A\ P$› ‹$X' \subset X$› **have** *disjnt* $X\ X'$
    by (*metis disjnt-def disjointD inf.strict-order-iff partition-onD2*)

moreover note ‹$X' \subset X$›
ultimately show *False*
by (*meson all-not-in-conv disjnt-iff psubsetD*)
qed

lemma *partition-on-Diff*:
assumes *P*: *partition-on A P* **shows** $Q \subseteq P \Longrightarrow$ *partition-on* $(A - \bigcup Q)$ $(P - Q)$
using *P P*[*THEN partition-onD4*] **by** (*auto simp*: *partition-on-def disjoint-def*)

lemma *partition-on-UN*:
assumes *A*: *partition-on A B* **and** *B*: $\bigwedge b.\ b \in B \Longrightarrow$ *partition-on b* $(P\ b)$
shows *partition-on A* $(\bigcup b{\in}B.\ P\ b)$
proof (*rule partition-onI*)
show $\bigcup (\bigcup b{\in}B.\ P\ b) = A$
using *B*[*THEN partition-onD1*] *A*[*THEN partition-onD1*] **by** *blast*
next
show $p \neq \{\}$ **if** $p \in (\bigcup b{\in}B.\ P\ b)$ **for** *p*
using *B*[*THEN partition-onD3*] *that* **by** *auto*
next
fix *p q* **assume** $p \in (\bigcup i{\in}B.\ P\ i)$ $q \in (\bigcup i{\in}B.\ P\ i)$ **and** $p \neq q$
then obtain *i j* **where** *i*: $p \in P\ i\ i \in B$ **and** *j*: $q \in P\ j\ j \in B$
by *auto*
show $p \cap q = \{\}$
proof *cases*
assume $i = j$ **then show** *?thesis*
using *i j* ‹$p \neq q$› *B*[*THEN partition-onD2*, *of i*] **by** (*simp add*: *disjointD*)
next
assume $i \neq j$
then have *disjnt i j*
using *i j A*[*THEN partition-onD2*] **by** (*auto simp*: *pairwise-def*)
moreover have $p \subseteq i$ $q \subseteq j$
using *B*[*THEN partition-onD1*, *of i*, *symmetric*] *B*[*THEN partition-onD1*, *of j*, *symmetric*] *i j* **by** *auto*
ultimately show *?thesis*
by (*auto simp*: *disjnt-def*)
qed
qed

lemma *partition-on-notemptyI*:
assumes *partition-on A P*
assumes $A \neq \{\}$
shows $P \neq \{\}$
using *assms* **by** (*auto elim*: *partition-onE*)

lemma *partition-on-disjoint*:
assumes *partition-on A P*
assumes *partition-on B Q*
assumes $A \cap B = \{\}$

shows $P \cap Q = \{\}$
using *assms* **by** (*fastforce elim*: *partition-onE*)

lemma *partition-on-eq-implies-eq-carrier*:
assumes *partition-on* A Q
assumes *partition-on* B Q
shows $A = B$
using *assms* **by** (*fastforce elim*: *partition-onE*)

lemma *partition-on-insert*:
assumes *partition-on* A P
assumes *disjnt* A X $X \neq \{\}$
assumes $A \cup X = A'$
shows *partition-on* A' (*insert* X P)
using *assms* **by** (*auto simp*: *partition-on-def disjoint-def disjnt-def*)

An alternative formulation of $[\![$*partition-on* $?A$ $?P$; *disjnt* $?A$ $?X$; $?X \neq \{\}$; $?A \cup ?X = ?A']\!] \Longrightarrow$ *partition-on* $?A'$ (*insert* $?X$ $?P$)

lemma *partition-on-insert'*:
assumes *partition-on* $(A - X)$ P
assumes $X \subseteq A$ $X \neq \{\}$
shows *partition-on* A (*insert* X P)
proof −
have *disjnt* $(A - X)$ X **by** (*simp add*: *disjnt-iff*)
from *assms*(*1*) *this assms*(*3*) **have** *partition-on* $((A - X) \cup X)$ (*insert* X P)
by (*auto intro*: *partition-on-insert*)
from *this* ‹$X \subseteq A$› **show** *?thesis*
by (*metis Diff-partition sup-commute*)
qed

lemma *partition-on-insert-singleton*:
assumes *partition-on* A P $a \notin A$ *insert* a $A = A'$
shows *partition-on* A' (*insert* $\{a\}$ P)
using *assms* **by** (*auto simp*: *partition-on-def disjoint-def disjnt-def*)

lemma *partition-on-remove-singleton*:
assumes *partition-on* A P $X \in P$ $A - X = A'$
shows *partition-on* A' $(P - \{X\})$
using *assms partition-on-Diff* **by** (*metis Diff-cancel Diff-subset cSup-singleton insert-subset*)

1.4 The Unique Part Containing an Element in a Set Partition

lemma *partition-on-partition-on-unique*:
assumes *partition-on* A P
assumes $x \in A$
shows $\exists!X.\ x \in X \wedge X \in P$
proof −

from ‹*partition-on* A P› **have** $\bigcup P = A$
  by (*auto elim*: *partition-onE*)
from *this* ‹$x \in A$› **obtain** X **where** X: $x \in X \wedge X \in P$ **by** *blast*
{
  fix Y
  assume $x \in Y \wedge Y \in P$
  from *this* **have** $X = Y$
    using X ‹*partition-on* A P› **by** (*meson partition-onE disjoint-iff-not-equal*)
}
from *this* X **show** *?thesis* **by** *auto*
qed

lemma *partition-on-the-part-mem*:
  assumes *partition-on* A P
  assumes $x \in A$
  shows ($THE\ X.\ x \in X \wedge X \in P) \in P$
proof −
  from ‹$x \in A$› **have** $\exists!X.\ x \in X \wedge X \in P$
    using ‹*partition-on* A P› **by** (*simp add*: *partition-on-partition-on-unique*)
  from *this* **show** ($THE\ X.\ x \in X \wedge X \in P) \in P$
    by (*metis* (*no-types*, *lifting*) *theI*)
qed

lemma *partition-on-in-the-unique-part*:
  assumes *partition-on* A P
  assumes $x \in A$
  shows $x \in (THE\ X.\ x \in X \wedge X \in P)$
proof −
  from *assms* **have** $\exists!X.\ x \in X \wedge X \in P$
    by (*simp add*: *partition-on-partition-on-unique*)
  from *this* **show** *?thesis*
    by (*metis* (*mono-tags*, *lifting*) *theI′*)
qed

lemma *partition-on-the-part-eq*:
  assumes *partition-on* A P
  assumes $x \in X$ $X \in P$
  shows ($THE\ X.\ x \in X \wedge X \in P) = X$
proof −
  from ‹$x \in X$› ‹$X \in P$› **have** $x \in A$
    using ‹*partition-on* A P› **by** (*auto elim*: *partition-onE*)
  from *this* **have** $\exists!X.\ x \in X \wedge X \in P$
    using ‹*partition-on* A P› **by** (*simp add*: *partition-on-partition-on-unique*)
  from ‹$x \in X$› ‹$X \in P$› *this* **show** ($THE\ X.\ x \in X \wedge X \in P) = X$
    by (*auto intro*!: *the1-equality*)
qed

lemma *the-unique-part-alternative-def*:

assumes *partition-on* A P
assumes $x \in A$
shows $(THE\ X.\ x \in X \wedge X \in P) = \{y.\ \exists X{\in}P.\ x \in X \wedge y \in X\}$
proof
show $(THE\ X.\ x \in X \wedge X \in P) \subseteq \{y.\ \exists X{\in}P.\ x \in X \wedge y \in X\}$
proof
fix y
assume $y \in (THE\ X.\ x \in X \wedge X \in P)$
moreover from ‹$x \in A$› **have** $x \in (THE\ X.\ x \in X \wedge X \in P)$
using ‹*partition-on* A P› *partition-on-in-the-unique-part* **by** *force*
moreover from ‹$x \in A$› **have** $(THE\ X.\ x \in X \wedge X \in P) \in P$
using ‹*partition-on* A P› *partition-on-the-part-mem* **by** *force*
ultimately show $y \in \{y.\ \exists X{\in}P.\ x \in X \wedge y \in X\}$ **by** *auto*
qed
next
show $\{y.\ \exists X{\in}P.\ x \in X \wedge y \in X\} \subseteq (THE\ X.\ x \in X \wedge X \in P)$
proof
fix y
assume $y \in \{y.\ \exists X{\in}P.\ x \in X \wedge y \in X\}$
from *this* **obtain** X **where** $x \in X$ **and** $y \in X$ **and** $X \in P$ **by** *auto*
from ‹$x \in X$› ‹$X \in P$› **have** $(THE\ X.\ x \in X \wedge X \in P) = X$
using ‹*partition-on* A P› *partition-on-the-part-eq* **by** *force*
from *this* ‹$y \in X$› **show** $y \in (THE\ X.\ x \in X \wedge X \in P)$ **by** *simp*
qed
qed

lemma *partition-on-all-in-part-eq-part*:
assumes *partition-on* A P
assumes $X' \in P$
shows $\{x \in A.\ (THE\ X.\ x \in X \wedge X \in P) = X'\} = X'$
proof
show $\{x \in A.\ (THE\ X.\ x \in X \wedge X \in P) = X'\} \subseteq X'$
using *assms*(*1*) *partition-on-in-the-unique-part* **by** *force*
next
show $X' \subseteq \{x \in A.\ (THE\ X.\ x \in X \wedge X \in P) = X'\}$
proof
fix x
assume $x \in X'$
from ‹$x \in X'$› ‹$X' \in P$› **have** $x \in A$
using ‹*partition-on* A P› **by** (*auto elim*: *partition-onE*)
moreover from ‹$x \in X'$› ‹$X' \in P$› **have** $(THE\ X.\ x \in X \wedge X \in P) = X'$
using ‹*partition-on* A P› *partition-on-the-part-eq* **by** *fastforce*
ultimately show $x \in \{x \in A.\ (THE\ X.\ x \in X \wedge X \in P) = X'\}$ **by** *auto*
qed
qed

lemma *partition-on-part-characteristic*:
assumes *partition-on* A P
assumes $X \in P$ $x \in X$

shows $X = \{y.\ \exists X{\in}P.\ x \in X \wedge y \in X\}$
proof $-$
from ‹$x \in X$› ‹$X \in P$› **have** $x \in A$
using ‹*partition-on A P*› *partition-onE* **by** *blast*
from ‹$x \in X$› ‹$X \in P$› **have** $X = (\mathit{THE}\ X.\ x \in X \wedge X \in P)$
using ‹*partition-on A P*› **by** (*simp add*: *partition-on-the-part-eq*)
also from ‹$x \in A$› **have** $(\mathit{THE}\ X.\ x \in X \wedge X \in P) = \{y.\ \exists X{\in}P.\ x \in X \wedge y \in X\}$
using ‹*partition-on A P*› *the-unique-part-alternative-def* **by** *force*
finally show *?thesis* **.**
qed

lemma *partition-on-no-partition-outside-carrier*:
assumes *partition-on A P*
assumes $x \notin A$
shows $\{y.\ \exists X{\in}P.\ x \in X \wedge y \in X\} = \{\}$
using *assms* **unfolding** *partition-on-def* **by** *auto*

1.5 Cardinality of Parts in a Set Partition

lemma *partition-on-le-set-elements*:
assumes *finite A*
assumes *partition-on A P*
shows $\mathit{card}\ P \le \mathit{card}\ A$
using *assms*
proof (*induct A arbitrary*: *P*)
case *empty*
from *this* **show** $\mathit{card}\ P \le \mathit{card}\ \{\}$ **by** (*simp add*: *partition-on-empty*)
next
case (*insert a A*)
show *?case*
proof (*cases* $\{a\} \in P$)
case *True*
have *prop-partition-on*: $\forall p{\in}P.\ p \neq \{\}$ $\bigcup P = \mathit{insert}\ a\ A$
$\forall p{\in}P.\ \forall p'{\in}P.\ p \neq p' \longrightarrow p \cap p' = \{\}$
using ‹*partition-on* (*insert a A*) *P*› **by** (*fastforce elim*: *partition-onE*)+
from *this*(*2*, *3*) ‹$a \notin A$› ‹$\{a\} \in P$› **have** *A-eq*: $A = \bigcup(P - \{\{a\}\})$
by *auto* (*metis Int-iff UnionI empty-iff insert-iff*)
from *prop-partition-on A-eq* **have** *partition-on*: *partition-on A* $(P - \{\{a\}\})$
by (*intro partition-onI*) *auto*
from *insert.hyps*(*3*) *this* **have** $\mathit{card}\ (P - \{\{a\}\}) \le \mathit{card}\ A$ **by** *simp*
from *this insert*(*1*, *2*, *4*) ‹$\{a\} \in P$› **show** *?thesis*
using *finite-elements*[*OF* ‹*finite A*› *partition-on*] **by** *simp*
next
case *False*
from ‹*partition-on* (*insert a A*) *P*› **obtain** *p* **where** *p-def*: $p \in P$ $a \in p$
by (*blast elim*: *partition-onE*)
from ‹*partition-on* (*insert a A*) *P*› *p-def* **have** *a-notmem*: $\forall p'{\in}\ P - \{p\}.\ a \notin p'$

    by (*blast elim*: *partition-onE*)
  from ‹*partition-on* (*insert a A*) *P*› *p-def* **have** $p - \{a\} \notin P$
    unfolding *partition-on-def disjoint-def*
    by (*metis Diff-insert-absorb Diff-subset inf.orderE mk-disjoint-insert*)
  let $?P' = insert\ (p - \{a\})\ (P - \{p\})$
  have *partition-on A ?P'*
  proof (*rule partition-onI*)
    from ‹*partition-on* (*insert a A*) *P*› **have** $\forall p{\in}P.\ p \neq \{\}$ **by** (*auto elim*: *partition-onE*)
    from *this p-def* ‹$\{a\} \notin P$› **show** $\bigwedge p'.\ p'{\in}insert\ (p - \{a\})\ (P - \{p\}) \Longrightarrow p' \neq \{\}$
      by (*simp*; *metis* (*no-types*) *Diff-eq-empty-iff subset-singletonD*)
  next
    from ‹*partition-on* (*insert a A*) *P*› **have** $\bigcup P = insert\ a\ A$ **by** (*auto elim*: *partition-onE*)
    from *p-def this* ‹$a \notin A$› *a-notmem* **show** $\bigcup(insert\ (p - \{a\})\ (P - \{p\})) = A$ **by** *auto*
  next
    show $\bigwedge pa\ pa'.\ pa{\in}insert\ (p - \{a\})\ (P - \{p\}) \Longrightarrow pa'{\in}insert\ (p - \{a\})\ (P - \{p\}) \Longrightarrow pa \neq pa' \Longrightarrow pa \cap pa' = \{\}$
      using ‹*partition-on* (*insert a A*) *P*› *p-def a-notmem*
      unfolding *partition-on-def disjoint-def*
      by (*metis disjoint-iff-not-equal insert-Diff insert-iff*)
  qed
  have *finite P* **using** ‹*finite A*› ‹*partition-on A ?P'*› *finite-elements* **by** *fastforce*
  have $card\ P = Suc\ (card\ (P - \{p\}))$
    using *p-def* ‹*finite P*› *card.remove* **by** *fastforce*
  also have $\ldots = card\ ?P'$ **using** ‹$p - \{a\} \notin P$› ‹*finite P*› **by** *simp*
  also have $\ldots \le card\ A$ **using** ‹*partition-on A ?P'*› *insert.hyps*(*3*) **by** *simp*
  also have $\ldots \le card\ (insert\ a\ A)$ **by** (*simp add*: *card-insert-le* ‹*finite A*›)
  finally show *?thesis* **.**
 qed
qed

1.6 Operations on Set Partitions

lemma *partition-on-union*:
  assumes $A \cap B = \{\}$
  assumes *partition-on A P*
  assumes *partition-on B Q*
  shows *partition-on* $(A \cup B)\ (P \cup Q)$
proof (*rule partition-onI*)
  fix *X*
  assume $X \in P \cup Q$
  from *this* ‹*partition-on A P*› ‹*partition-on B Q*› **show** $X \neq \{\}$
    by (*auto elim*: *partition-onE*)
next
  show $\bigcup(P \cup Q) = A \cup B$
    using ‹*partition-on A P*› ‹*partition-on B Q*› **by** (*auto elim*: *partition-onE*)

next
  fix $X\ Y$
  assume $X \in P \cup Q\ Y \in P \cup Q\ X \neq Y$
  from *this assms* **show** $X \cap Y = \{\}$
    by (*elim UnE partition-onE*) *auto*
qed

lemma *partition-on-split1*:
  assumes *partition-on* $A\ (P \cup Q)$
  shows *partition-on* $(\bigcup P)\ P$
proof (*rule partition-onI*)
  fix p
  assume $p \in P$
  from *this assms* **show** $p \neq \{\}$
    using *Un-iff partition-onE* **by** *auto*
next
  show $\bigcup P = \bigcup P$ **..**
next
  fix $p\ p'$
  assume a: $p \in P\ p' \in P\ p \neq p'$
  from *this assms* **show** $p \cap p' = \{\}$
    using *partition-onE subsetCE sup-ge1* **by** *blast*
qed

lemma *partition-on-split2*:
  assumes *partition-on* $A\ (P \cup Q)$
  shows *partition-on* $(\bigcup Q)\ Q$
using *assms partition-on-split1 sup-commute* **by** *metis*

lemma *partition-on-intersect-on-elements*:
  assumes *partition-on* $(A \cup C)\ P$
  assumes $\forall X \in P.\ \exists x.\ x \in X \cap C$
  shows *partition-on* $C\ ((\lambda X.\ X \cap C)$ ‘ $P)$
proof (*rule partition-onI*)
  fix p
  assume $p \in (\lambda X.\ X \cap C)$ ‘ P
  from *this assms* **show** $p \neq \{\}$ **by** *auto*
next
  have $\bigcup P = A \cup C$
    using *assms* **by** (*auto elim*: *partition-onE*)
  from *this* **show** $\bigcup((\lambda X.\ X \cap C)$ ‘ $P) = C$ **by** *auto*
next
  fix $p\ p'$
  assume $p \in (\lambda X.\ X \cap C)$ ‘ $P\ p' \in (\lambda X.\ X \cap C)$ ‘ $P\ p \neq p'$
  from *this assms*(*1*) **show** $p \cap p' = \{\}$
    by (*blast elim*: *partition-onE*)
qed

lemma *partition-on-insert-elements*:

assumes $A \cap B = \{\}$
assumes *partition-on* B P
assumes $f \in A \rightarrow_E P$
shows *partition-on* $(A \cup B)$ $((\lambda X.\ X \cup \{x \in A.\ f\,x = X\})$ ‘ $P)$ (**is** *partition-on* - *?P*)
proof (*rule partition-onI*)
fix X
assume $X \in$ *?P*
from *this* ‹*partition-on* B P› **show** $X \neq \{\}$
by (*auto elim*: *partition-onE*)
next
show $\bigcup$ *?P* $= A \cup B$
using ‹*partition-on* B P› ‹$f \in A \rightarrow_E P$› **by** (*auto elim*: *partition-onE*)
next
fix X Y
assume $X \in$ *?P* $Y \in$ *?P* $X \neq Y$
from ‹$X \in$ *?P*› **obtain** X' **where** X': $X = X' \cup \{x \in A.\ f\,x = X'\}$ $X' \in P$ **by** *auto*
from ‹$Y \in$ *?P*› **obtain** Y' **where** Y': $Y = Y' \cup \{x \in A.\ f\,x = Y'\}$ $Y' \in P$ **by** *auto*
from ‹$X \neq Y$› X' Y' **have** $X' \neq Y'$ **by** *auto*
from *this* X' Y' **have** $X' \cap Y' = \{\}$
using ‹*partition-on* B P› **by** (*auto elim*!: *partition-onE*)
from X' Y' **have** $X' \subseteq B$ $Y' \subseteq B$
using ‹*partition-on* B P› **by** (*auto elim*!: *partition-onE*)
from *this* ‹$X' \cap Y' = \{\}$› X' Y' ‹$X' \neq Y'$› **show** $X \cap Y = \{\}$
using ‹$A \cap B = \{\}$› **by** *auto*
qed

lemma *partition-on-map*:
assumes *inj-on* f A
assumes *partition-on* A P
shows *partition-on* $(f$ ‘ $A)$ $((‘)$ f ‘ $P)$
proof −
{
fix X Y
assume $X \in P$ $Y \in P$ f ‘ $X \neq f$ ‘ Y
moreover from *assms* **have** $\forall p{\in}P.\ \forall p'{\in}P.\ p \neq p' \longrightarrow p \cap p' = \{\}$ **and** *inj-on* f $(\bigcup P)$
by (*auto elim*!: *partition-onE*)
ultimately have f ‘ $X \cap f$ ‘ $Y = \{\}$
unfolding *inj-on-def* **by** *auto* (*metis IntI empty-iff rev-image-eqI*)+
}
from *assms this* **show** *partition-on* $(f$ ‘ $A)$ $((‘)$ f ‘ $P)$
by (*auto intro*!: *partition-onI elim*!: *partition-onE*)
qed

lemma *set-of-partition-on-map*:
assumes *inj-on* f A

shows $(\text{`})\ ((\text{`})\ f)\ \text{`}\ \{P.\ \textit{partition-on}\ A\ P\} = \{P.\ \textit{partition-on}\ (f\ \text{`}\ A)\ P\}$
proof (*rule set-eqI′*)
  fix x
  assume $x \in (\text{`})\ ((\text{`})\ f)\ \text{`}\ \{P.\ \textit{partition-on}\ A\ P\}$
  from *this* ‹*inj-on f A*› **show** $x \in \{P.\ \textit{partition-on}\ (f\ \text{`}\ A)\ P\}$
    by (*auto intro*: *partition-on-map*)
next
  fix P
  assume $P \in \{P.\ \textit{partition-on}\ (f\ \text{`}\ A)\ P\}$
  from *this* **have** *partition-on* $(f\ \text{`}\ A)\ P$ **by** *auto*
  from *this* **have** *mem*: $\bigwedge X\ x.\ X \in P \Longrightarrow x \in X \Longrightarrow x \in f\ \text{`}\ A$
    by (*auto elim*!: *partition-onE*)
  have $(\text{`})\ (f \circ \textit{the-inv-into}\ A\ f)\ \text{`}\ P = (\text{`})\ f\ \text{`}\ (\text{`})\ (\textit{the-inv-into}\ A\ f)\ \text{`}\ P$
    by (*simp add*: *image-image cong*: *image-cong-simp*)
  moreover have $P = (\text{`})\ (f \circ \textit{the-inv-into}\ A\ f)\ \text{`}\ P$
  proof (*rule set-eqI′*)
    fix X
    assume X: $X \in P$
    moreover from X *mem* **have** *in-range*: $\forall x{\in}X.\ x \in f\ \text{`}\ A$ **by** *auto*
    moreover have $X = (f \circ \textit{the-inv-into}\ A\ f)\ \text{`}\ X$
    proof (*rule set-eqI′*)
      fix x
      assume $x \in X$
      show $x \in (f \circ \textit{the-inv-into}\ A\ f)\ \text{`}\ X$
      proof (*rule image-eqI*)
        from *in-range* ‹$x \in X$› *assms* **show** $x = (f \circ \textit{the-inv-into}\ A\ f)\ x$
          by (*auto simp add*: *f-the-inv-into-f*[*of f*])
        from ‹$x \in X$› **show** $x \in X$ **by** *assumption*
      qed
    next
      fix x
      assume $x \in (f \circ \textit{the-inv-into}\ A\ f)\ \text{`}\ X$
      from *this* **obtain** x' **where** x': $x' \in X \wedge x = f\ (\textit{the-inv-into}\ A\ f\ x')$ **by** *auto*
      from *in-range* x' **have** f: $f\ (\textit{the-inv-into}\ A\ f\ x') \in X$
        by (*subst f-the-inv-into-f*[*of f*]) (*auto intro*: ‹*inj-on f A*›)
      from x' ‹$X \in P$› f **show** $x \in X$ **by** *auto*
    qed
    ultimately show $X \in (\text{`})\ (f \circ \textit{the-inv-into}\ A\ f)\ \text{`}\ P$ **by** *auto*
  next
    fix X
    assume $X \in (\text{`})\ (f \circ \textit{the-inv-into}\ A\ f)\ \text{`}\ P$
    moreover
    {
      fix Y
      assume $Y \in P$
      from *this* ‹*inj-on f A*› *mem* **have** $\forall x{\in}Y.\ f\ (\textit{the-inv-into}\ A\ f\ x) = x$
        by (*auto simp add*: *f-the-inv-into-f*)
      from *this* **have** $(f \circ \textit{the-inv-into}\ A\ f)\ \text{`}\ Y = Y$ **by** *force*
    }

ultimately show $X \in P$ **by** *auto*
qed
ultimately have P: $P = (\text{`})\ f\ \text{`}\ (\text{`})\ (\textit{the-inv-into}\ A\ f)\ \text{`}\ P$ **by** *simp*
have *A-eq*: $A = \textit{the-inv-into}\ A\ f\ \text{`}\ f\ \text{`}\ A$ **by** (*simp add*: *assms*)
from ‹*inj-on f A*› **have** *inj-on* (*the-inv-into A f*) (f ‘ A)
using ‹*partition-on* (f ‘ A) P› **by** (*simp add*: *inj-on-the-inv-into*)
from *this* **have** (‘) (*the-inv-into A f*) ‘ $P \in \{P.\ \textit{partition-on}\ A\ P\}$
using ‹*partition-on* (f ‘ A) P› **by** (*subst A-eq*, *auto intro*!: *partition-on-map*)
from P *this* **show** $P \in (\text{`})\ ((\text{`})\ f)\ \text{`}\ \{P.\ \textit{partition-on}\ A\ P\}$ **by** (*rule image-eqI*)
qed

end

2 Combinatorial Basics

theory *Injectivity-Solver*
imports
HOL−Library.Disjoint-Sets
HOL−Library.Monad-Syntax
HOL−Eisbach.Eisbach
begin

2.1 Preliminaries

These lemmas shall be added to the Disjoint Set theory.

2.1.1 Injectivity and Disjoint Families

lemma *inj-on-impl-disjoint-family-on-singleton*:
assumes *inj-on f A*
shows *disjoint-family-on* ($\lambda x.\ \{f\ x\}$) A
using *assms disjoint-family-on-def inj-on-contraD* **by** *fastforce*

2.1.2 Cardinality Theorems for Set.bind

lemma *card-bind*:
assumes *finite S*
assumes $\forall X \in S.\ \textit{finite}\ (f\ X)$
assumes *disjoint-family-on f S*
shows $\textit{card}\ (S \gg\!= f) = (\sum x{\in}S.\ \textit{card}\ (f\ x))$
proof −
have $\textit{card}\ (S \gg\!= f) = \textit{card}\ (\bigcup (f\ \text{`}\ S))$
by (*simp add*: *bind-UNION*)
also have $\textit{card}\ (\bigcup (f\ \text{`}\ S)) = (\sum x{\in}S.\ \textit{card}\ (f\ x))$
using *assms* **unfolding** *disjoint-family-on-def* **by** (*simp add*: *card-UN-disjoint*)
finally show *?thesis* **.**
qed

lemma *card-bind-constant*:
  assumes *finite S*
  assumes $\forall X \in S.\ \mathit{finite}\ (f\ X)$
  assumes *disjoint-family-on f S*
  assumes $\bigwedge x.\ x \in S \Longrightarrow \mathit{card}\ (f\ x) = k$
  shows $\mathit{card}\ (S \ggg f) = \mathit{card}\ S * k$
using *assms* **by** (*simp add*: *card-bind*)

lemma *card-bind-singleton*:
  assumes *finite S*
  assumes *inj-on f S*
  shows $\mathit{card}\ (S \ggg (\lambda x.\ \{f\ x\})) = \mathit{card}\ S$
using *assms* **by** (*auto simp add*: *card-bind-constant inj-on-impl-disjoint-family-on-singleton*)

2.2 Third Version of Injectivity Solver

Here, we provide a third version of the injectivity solver. The original first version was provided in the AFP entry 'Spivey's Generalized Recurrence for Bell Numbers'. From that method, I derived a second version in the AFP entry 'Cardinality of Equivalence Relations'. At roughly the same time, Makarius improved the injectivity solver in the development version of the first AFP entry. This third version now includes the improvements of the second version and Makarius improvements to the first, and it further extends the method to handle the new cases in the cardinality proof of this AFP entry.

As the implementation of the injectivity solver only evolves in the development branch of the AFP, the submissions of the three AFP entries that employ the injectivity solver, have to create clones of the injectivity solver for the identified and needed method adjustments. Ultimately, these three clones should only remain in the stable branches of the AFP from Isabelle2016 to Isabelle2017 to work with their corresponding release versions. In the development version, I have now consolidated the three versions here. In the next step, I will move this version of the injectivity solver in the *HOL−Library.Disjoint-Sets* and it will hopefully only evolve further there.

lemma *disjoint-family-onI*:
  assumes $\bigwedge i\ j.\ i \in I \wedge j \in I \Longrightarrow i \neq j \Longrightarrow (A\ i) \cap (A\ j) = \{\}$
  shows *disjoint-family-on A I*
using *assms* **unfolding** *disjoint-family-on-def* **by** *auto*

lemma *disjoint-bind*: $\bigwedge S\ T\ f\ g.\ (\bigwedge s\ t.\ S\ s \wedge T\ t \Longrightarrow f\ s \cap g\ t = \{\}) \Longrightarrow (\{s.\ S\ s\} \ggg f) \cap (\{t.\ T\ t\} \ggg g) = \{\}$
by *fastforce*

lemma *disjoint-bind'*: $\bigwedge S\ T\ f\ g.\ (\bigwedge s\ t.\ s \in S \wedge t \in T \Longrightarrow f\ s \cap g\ t = \{\}) \Longrightarrow (S \ggg f) \cap (T \ggg g) = \{\}$
by *fastforce*

lemma *injectivity-solver-CollectE*:
  assumes $a \in \{x.\ P\ x\} \wedge a' \in \{x.\ P'\ x\}$
  assumes $(P\ a \wedge P'\ a') \Longrightarrow W$
  shows W
using *assms* **by** *auto*

lemma *injectivity-solver-prep-assms-Collect*:
  assumes $x \in \{x.\ P\ x\}$
  shows $P\ x \wedge P\ x$
using *assms* **by** *simp*

lemma *injectivity-solver-prep-assms*: $x \in A \Longrightarrow x \in A \wedge x \in A$
  by *simp*

lemma *disjoint-terminal-singleton*: $\bigwedge s\ t\ X\ Y.\ s \neq t \Longrightarrow (X = Y \Longrightarrow s = t) \Longrightarrow \{X\} \cap \{Y\} = \{\}$
by *auto*

lemma *disjoint-terminal-Collect*:
  assumes $s \neq t$
  assumes $\bigwedge x\ x'.\ S\ x \wedge T\ x' \Longrightarrow x = x' \Longrightarrow s = t$
  shows $\{x.\ S\ x\} \cap \{x.\ T\ x\} = \{\}$
using *assms* **by** *auto*

lemma *disjoint-terminal*:
  $s \neq t \Longrightarrow (\bigwedge x\ x'.\ x \in S \wedge x' \in T \Longrightarrow x = x' \Longrightarrow s = t) \Longrightarrow S \cap T = \{\}$
by *blast*

lemma *elim-singleton*:
  assumes $x \in \{s\} \wedge x' \in \{t\}$
  obtains $x = s \wedge x' = t$
using *assms* **by** *blast*

method *injectivity-solver* **uses** *rule* =
  insert method-facts,
  use nothing **in** ‹
   ((*drule injectivity-solver-prep-assms-Collect* | *drule injectivity-solver-prep-assms*)+)?;
    rule disjoint-family-onI;
    ((*rule disjoint-bind* | *rule disjoint-bind'*)+)?;
    (*erule elim-singleton*)?;
    (*erule disjoint-terminal-singleton* | *erule disjoint-terminal-Collect* | *erule disjoint-terminal*);
    (*elim injectivity-solver-CollectE*)?;
    rule rule;
    assumption+
  ›

end

3 Cardinality of Set Partitions

```
theory Card-Partitions
imports
  HOL−Combinatorics.Stirling
  Set-Partition
  Injectivity-Solver
begin

lemma set-partition-on-insert-with-fixed-card-eq:
  assumes finite A
  assumes a ∉ A
  shows {P. partition-on (insert a A) P ∧ card P = Suc k} = (do {
     P <− {P. partition-on A P ∧ card P = Suc k};
     p <− P;
     {insert (insert a p) (P − {p})}
  })
  ∪ (do {
    P <− {P. partition-on A P ∧ card P = k};
    {insert {a} P}
  }) (is ?S = ?T)
proof
  show ?S ⊆ ?T
  proof
    fix P
    assume P ∈ {P. partition-on (insert a A) P ∧ card P = Suc k}
    from this have partition-on (insert a A) P and card P = Suc k by auto
    show P ∈ ?T
    proof cases
      assume {a} ∈ P
      have partition-on A (P − {{a}})
        using ‹{a} ∈ P› ‹partition-on (insert a A) P›[THEN partition-on-Diff, of
{{a}}] ‹a ∉ A›
        by auto
      moreover from ‹{a} ∈ P› ‹card P = Suc k› have card (P − {{a}}) = k
        by (subst card-Diff-singleton) (auto intro: card-ge-0-finite)
      moreover from ‹{a} ∈ P› have P = insert {a} (P − {{a}}) by auto
      ultimately have P ∈ {P. partition-on A P ∧ card P = k} ≫= (λP. {insert
{a} P})
        by auto
      from this show ?thesis by auto
    next
      assume {a} ∉ P
      let ?p' = (THE X. a ∈ X ∧ X ∈ P)
      let ?p = (THE X. a ∈ X ∧ X ∈ P) − {a}
      let ?P' = insert ?p (P − {?p'})
      from ‹partition-on (insert a A) P› have a ∈ (THE X. a ∈ X ∧ X ∈ P)
        using partition-on-in-the-unique-part by fastforce
      from ‹partition-on (insert a A) P› have (THE X. a ∈ X ∧ X ∈ P) ∈ P
```

using *partition-on-the-part-mem* **by** *fastforce*
from *this* ‹*partition-on* (*insert a A*) *P*› **have** (*THE X*. $a \in X \wedge X \in P$) − $\{a\} \notin P$
using *partition-subset-imp-notin* ‹$a \in$ (*THE X*. $a \in X \wedge X \in P$)› **by** *blast*
have (*THE X*. $a \in X \wedge X \in P$) $\neq \{a\}$
using ‹(*THE X*. $a \in X \wedge X \in P$) $\in P$› ‹$\{a\} \notin P$› **by** *auto*
from ‹*partition-on* (*insert a A*) *P*› **have** (*THE X*. $a \in X \wedge X \in P$) $\subseteq$ *insert a A*
using ‹(*THE X*. $a \in X \wedge X \in P$) $\in P$› *partition-onD1* **by** *fastforce*
note *facts-on-the-part-of* = ‹$a \in$ (*THE X*. $a \in X \wedge X \in P$)› ‹(*THE X*. $a \in X \wedge X \in P$) $\in P$›
‹(*THE X*. $a \in X \wedge X \in P$) − $\{a\} \notin P$› ‹(*THE X*. $a \in X \wedge X \in P$) $\neq \{a\}$›
‹(*THE X*. $a \in X \wedge X \in P$) $\subseteq$ *insert a A*›
from ‹*partition-on* (*insert a A*) *P*› ‹*finite A*› **have** *finite P*
by (*meson finite.insertI finite-elements*)
from ‹*partition-on* (*insert a A*) *P*› ‹$a \notin A$› **have** *partition-on* (*A* − *?p*) (*P* − {*?p'*})
using *facts-on-the-part-of* **by** (*auto intro*: *partition-on-remove-singleton*)
from *this* **have** *partition-on A ?P'*
using *facts-on-the-part-of* **by** (*auto intro*: *partition-on-insert simp add*: *disjnt-iff*)
moreover have *card ?P'* = *Suc k*
proof −
from ‹*card P* = *Suc k*› **have** *card* (*P* − {*THE X*. $a \in X \wedge X \in P$}) = *k*
using ‹*finite P*› ‹(*THE X*. $a \in X \wedge X \in P$) $\in P$› **by** *simp*
from *this* **show** *?thesis*
using ‹*finite P*› ‹(*THE X*. $a \in X \wedge X \in P$) − $\{a\} \notin P$› **by** (*simp add*: *card-insert-if*)
qed
moreover have *?p* $\in$ *?P'* **by** *auto*
moreover have *P* = *insert* (*insert a ?p*) (*?P'* − {*?p*})
using *facts-on-the-part-of* **by** (*auto simp add*: *insert-absorb*)
ultimately have $P \in$ {*P*. *partition-on A P* $\wedge$ *card P* = *Suc k*} $\gg=$ (λP. *P* $\gg=$ (λp. {*insert* (*insert a p*) (*P* − {*p*})}))
by *auto*
then show *?thesis* **by** *auto*
qed
qed
next
show *?T* $\subseteq$ *?S*
proof
fix *P*
assume $P \in$ *?T* (**is** - $\in$ *?subexpr1* $\cup$ *?subexpr2*)
from *this* **show** $P \in$ *?S*
proof
assume $P \in$ *?subexpr1*
from *this* **obtain** *p P'* **where** *P* = *insert* (*insert a p*) (*P'* − {*p*})
and *partition-on A P'* **and** *card P'* = *Suc k* **and** $p \in P'$ **by** *auto*
from ‹$p \in P'$› ‹*partition-on A P'*› **have** *partition-on* (*A* − *p*) (*P'* − {*p*})

by (*simp add*: *partition-on-remove-singleton*)
from ‹*partition-on A P'*› ‹*finite A*› **have** *finite P*
using ‹$P = \text{-}$› *finite-elements* **by** *auto*
from ‹*partition-on A P'*› ‹$a \notin A$› **have** *insert a p* $\notin P' - \{p\}$
using *partition-onD1* **by** *fastforce*
from ‹$P = \text{-}$› *this* ‹*card* $P' =$ *Suc k*› ‹*finite P*› ‹$p \in P'$›
have *card P* $=$ *Suc k* **by** *auto*
moreover have *partition-on* (*insert a A*) *P*
using ‹*partition-on* $(A - p)$ $(P' - \{p\})$› ‹$a \notin A$› ‹$p \in P'$› ‹*partition-on A* P'› ‹$P = \text{-}$›
by (*auto intro*!: *partition-on-insert dest*: *partition-onD1 simp add*: *disjnt-iff*)
ultimately show $P \in$ *?S* **by** *auto*
next
assume $P \in$ *?subexpr2*
from *this* **obtain** P' **where** $P =$ *insert* $\{a\}$ P' **and** *partition-on A* P' **and** *card* $P' = k$ **by** *auto*
from ‹*partition-on A P'*› ‹*finite A*› **have** *finite P*
using ‹$P =$ *insert* $\{a\}$ P'› *finite-elements* **by** *auto*
from ‹*partition-on A P'*› ‹$a \notin A$› **have** $\{a\} \notin P'$
using *partition-onD1* **by** *fastforce*
from ‹$P =$ *insert* $\{a\}$ P'› ‹*card* $P' = k$› *this* ‹*finite P*› **have** *card P* $=$ *Suc k* **by** *auto*
moreover from ‹*partition-on A P'*› ‹$a \notin A$› **have** *partition-on* (*insert a A*) *P*
using ‹$P =$ *insert* $\{a\}$ P'› **by** (*simp add*: *partition-on-insert-singleton*)
ultimately show $P \in$ *?S* **by** *auto*
qed
qed
qed

lemma *injectivity-subexpr1*:
assumes $a \notin A$
assumes $X \in P \wedge X' \in P'$
assumes *insert* (*insert a X*) $(P - \{X\}) =$ *insert* (*insert a* X') $(P' - \{X'\})$
assumes (*partition-on A P* $\wedge$ *card P* $=$ *Suc* k') $\wedge$ (*partition-on A* P' $\wedge$ *card* P' $=$ *Suc* k')
shows $P = P'$ **and** $X = X'$
proof $-$
from *assms*(*1*, *2*, *4*) **have** $a \notin X$ $a \notin X'$
using *partition-onD1* **by** *auto*
from *assms*(*1*, *4*) **have** *insert a X* $\notin P$ *insert a* $X' \notin P'$
using *partition-onD1* **by** *auto*
from *assms*(*1*, *3*, *4*) **have** *insert a X* $=$ *insert a* X'
by (*metis Diff-iff insertE insertI1 mem-simps*(*9*) *partition-onD1*)
from *this* ‹$a \notin X'$› ‹$a \notin X$› **show** $X = X'$
by (*meson insert-ident*)
from *assms*(*2*, *3*) **show** $P = P'$
using ‹*insert a X* $=$ *insert a* X'› ‹*insert a X* $\notin P$› ‹*insert a* $X' \notin P'$›
by (*metis insert-Diff insert-absorb insert-commute insert-ident*)

qed

lemma *injectivity-subexpr2*:
  assumes $a \notin A$
  assumes $insert\ \{a\}\ P = insert\ \{a\}\ P'$
  assumes $(partition\text{-}on\ A\ P \wedge card\ P = k') \wedge partition\text{-}on\ A\ P' \wedge card\ P' = k'$
  shows $P = P'$
proof −
  from *assms*(*1*, *3*) **have** $\{a\} \notin P$ **and** $\{a\} \notin P'$
    using *partition-onD1* **by** *auto*
  from ‹$\{a\} \notin P$› **have** $P = insert\ \{a\}\ P - \{\{a\}\}$ **by** *simp*
  also from ‹$insert\ \{a\}\ P = insert\ \{a\}\ P'$› **have** $\ldots = insert\ \{a\}\ P' - \{\{a\}\}$ **by** *simp*
  also from ‹$\{a\} \notin P'$› **have** $\ldots = P'$ **by** *simp*
  finally show *?thesis* **.**
qed

theorem *card-partition-on*:
  assumes *finite A*
  shows $card\ \{P.\ partition\text{-}on\ A\ P \wedge card\ P = k\} = Stirling\ (card\ A)\ k$
using *assms*
proof (*induct A arbitrary*: *k*)
  case *empty*
    have *eq*: $\{P.\ P = \{\} \wedge card\ P = 0\} = \{\{\}\}$ **by** *auto*
    show *?case* **by** (*cases k*) (*auto simp add*: *partition-on-empty eq*)
next
  case (*insert a A*)
  from *this* **show** *?case*
  proof (*cases k*)
    case *0*
    from *insert*(*1*) **have** *empty*: $\{P.\ partition\text{-}on\ (insert\ a\ A)\ P \wedge card\ P = 0\} = \{\}$
      unfolding *partition-on-def* **by** (*auto simp add*: *card-eq-0-iff finite-UnionD*)
    from *0 insert* **show** *?thesis* **by** (*auto simp add*: *empty*)
  next
    case (*Suc k'*)
    let *?subexpr1* = *do* {
      $P <- \{P.\ partition\text{-}on\ A\ P \wedge card\ P = Suc\ k'\}$;
      $p <- P$;
      $\{insert\ (insert\ a\ p)\ (P - \{p\})\}$
    }
    let *?subexpr2* = *do* {
      $P <- \{P.\ partition\text{-}on\ A\ P \wedge card\ P = k'\}$;
      $\{insert\ \{a\}\ P\}$
    }
    let $?expr = ?subexpr1 \cup ?subexpr2$
    have $card\ \{P.\ partition\text{-}on\ (insert\ a\ A)\ P \wedge card\ P = k\} = card\ ?expr$
    using ‹*finite A*› ‹$a \notin A$› ‹$k = Suc\ k'$› **by** (*simp add*: *set-partition-on-insert-with-fixed-card-eq*)
    also have $card\ ?expr = Stirling\ (card\ A)\ k' + Stirling\ (card\ A)\ (Suc\ k') * Suc$

k'

proof $-$

have *finite ?subexpr1* $\wedge$ *card ?subexpr1* $=$ *Stirling* (*card A*) (*Suc k'*) $*$ *Suc k'*

proof $-$

from ‹*finite A*› **have** *finite* {*P*. *partition-on A P* $\wedge$ *card P* $=$ *Suc k'*}

by (*simp add*: *finitely-many-partition-on*)

moreover have $\forall$ *X*$\in${*P*. *partition-on A P* $\wedge$ *card P* $=$ *Suc k'*}. *finite* (*X* $\ggg$= (λ*p*. {*insert* (*insert a p*) (*X* $-$ {*p*})}))

using *finite-elements* ‹*finite A*› *finite-bind*

by (*metis* (*no-types*, *lifting*) *finite.emptyI finite-insert mem-Collect-eq*)

moreover have *disjoint-family-on* (λ*P*. *P* $\ggg$= (λ*p*. {*insert* (*insert a p*) (*P* $-$ {*p*})})) {*P*. *partition-on A P* $\wedge$ *card P* $=$ *Suc k'*}

by (*injectivity-solver rule*: *injectivity-subexpr1*(*1*)[*OF* ‹*a* $\notin$ *A*›])

moreover have *card* (*P* $\ggg$= (λ*p*. {*insert* (*insert a p*) (*P* $-$ {*p*})})) $=$ *Suc k'*

if *P* $\in$ {*P*. *partition-on A P* $\wedge$ *card P* $=$ *Suc k'*} **for** *P*

proof $-$

from *that* ‹*finite A*› **have** *finite P*

using *finite-elements* **by** *blast*

moreover have *inj-on* (λ*p*. *insert* (*insert a p*) (*P* $-$ {*p*})) *P*

using *that injectivity-subexpr1*(*2*)[*OF* ‹*a* $\notin$ *A*›] **by** (*simp add*: *inj-onI*)

moreover from *that* **have** *card P* $=$ *Suc k'* **by** *simp*

ultimately show *?thesis* **by** (*simp add*: *card-bind-singleton*)

qed

ultimately have *card ?subexpr1* $=$ *card* {*P*. *partition-on A P* $\wedge$ *card P* $=$ *Suc k'*} $*$ *Suc k'*

by (*subst card-bind-constant*) *simp+*

from *this* **have** *card ?subexpr1* $=$ *Stirling* (*card A*) (*Suc k'*) $*$ *Suc k'*

using *insert.hyps*(*3*) **by** *simp*

moreover have *finite ?subexpr1*

using ‹*finite* {*P*. *partition-on A P* $\wedge$ *card P* $=$ *Suc k'*}›

‹$\forall$ *X*$\in${*P*. *partition-on A P* $\wedge$ *card P* $=$ *Suc k'*}. *finite* (*X* $\ggg$= (λ*p*. {*insert* (*insert a p*) (*X* $-$ {*p*})}))›

by (*auto intro*: *finite-bind*)

ultimately show *?thesis* **by** *blast*

qed

moreover have *finite ?subexpr2* $\wedge$ *card ?subexpr2* $=$ *Stirling* (*card A*) *k'*

proof $-$

from ‹*finite A*› **have** *finite* {*P*. *partition-on A P* $\wedge$ *card P* $=$ *k'*}

by (*simp add*: *finitely-many-partition-on*)

moreover have *inj-on* (*insert* {*a*}) {*P*. *partition-on A P* $\wedge$ *card P* $=$ *k'*}

using *injectivity-subexpr2*[*OF* ‹*a* $\notin$ *A*›] **by** (*simp add*: *inj-on-def*)

ultimately have *card ?subexpr2* $=$ *card* {*P*. *partition-on A P* $\wedge$ *card P* $=$ *k'*}

by (*simp add*: *card-bind-singleton*)

also have ... $=$ *Stirling* (*card A*) *k'*

using *insert.hyps*(*3*) **.**

finally have *card ?subexpr2* $=$ *Stirling* (*card A*) *k'* **.**

moreover have *finite ?subexpr2*

```
        by (simp add: ‹finite {P. partition-on A P ∧ card P = k'}› finite-bind)
      ultimately show ?thesis by blast
    qed
    moreover have ?subexpr1 ∩ ?subexpr2 = {}
    proof −
      have ∀ P∈?subexpr1. {a} ∉ P
        using insert.hyps(2) by (force elim!: partition-onE)
      moreover have ∀ P∈?subexpr2. {a} ∈ P by auto
      ultimately show ?subexpr1 ∩ ?subexpr2 = {} by blast
    qed
    ultimately show ?thesis
      by (simp add: card-Un-disjoint)
  qed
  also have ... = Stirling (card (insert a A)) k
    using insert(1, 2) ‹k = Suc k'› by simp
  finally show ?thesis .
 qed
qed

theorem card-partition-on-at-most-size:
 assumes finite A
 shows card {P. partition-on A P ∧ card P ≤ k} = (∑j≤k. Stirling (card A) j)
proof −
 have card {P. partition-on A P ∧ card P ≤ k} = card (⋃j≤k. {P. partition-on
A P ∧ card P = j})
   by (rule arg-cong[where f=card]) auto
 also have ... = (∑j≤k. card {P. partition-on A P ∧ card P = j})
  by (subst card-UN-disjoint) (auto simp add: ‹finite A› finitely-many-partition-on)
 also have (∑j≤k. card {P. partition-on A P ∧ card P = j}) = (∑j≤k. Stirling
(card A) j)
   using ‹finite A› by (simp add: card-partition-on)
 finally show ?thesis .
qed

theorem partition-on-size1:
 assumes finite A
 shows {P. partition-on A P ∧ (∀X∈P. card X = 1)} = {(λa. {a}) ‘ A}
proof
 show {P. partition-on A P ∧ (∀X∈P. card X = 1)} ⊆ {(λa. {a}) ‘ A}
 proof
   fix P
   assume P: P ∈ {P. partition-on A P ∧ (∀X∈P. card X = 1)}
   have P = (λa. {a}) ‘ A
   proof
     show P ⊆ (λa. {a}) ‘ A
     proof
       fix X
       assume X ∈ P
       from P this obtain x where X = {x}
```

```
      by (auto simp add: card-Suc-eq)
    from this ‹X ∈ P› have x ∈ A
      using P unfolding partition-on-def by blast
    from this ‹X = {x}› show X ∈(λa. {a}) ‘ A by auto
  qed
next
  show (λa. {a}) ‘ A ⊆ P
  proof
    fix X
    assume X ∈ (λa. {a}) ‘ A
    from this obtain x where X: X = {x} x ∈ A by auto
    have ⋃ P = A
      using P unfolding partition-on-def by blast
    from this ‹x ∈ A› obtain X′ where x ∈ X′ and X′ ∈ P
      using UnionE by blast
    from ‹X′ ∈ P› have card X′ = 1
      using P unfolding partition-on-def by auto
    from this ‹x ∈ X′› have X′ = {x}
      using card-1-singletonE by blast
    from this X(1) ‹X′ ∈ P› show X ∈ P by auto
  qed
qed
from this show P ∈ {(λa. {a}) ‘ A} by auto
qed
next
  show {(λa. {a}) ‘ A} ⊆ {P. partition-on A P ∧ (∀ X∈P. card X = 1)}
  proof
    fix P
    assume P ∈ {(λa. {a}) ‘ A}
    from this have P: P = (λa. {a}) ‘ A by auto
    from this have partition-on A P by (auto intro: partition-onI)
    from P this show P ∈ {P. partition-on A P ∧ (∀ X∈P. card X = 1)} by auto
  qed
qed

theorem card-partition-on-size1:
  assumes finite A
  shows card {P. partition-on A P ∧ (∀ X∈P. card X = 1)} = 1
using assms partition-on-size1 by fastforce

lemma card-partition-on-size1-eq-1:
  assumes finite A
  assumes card A ≤ k
  shows card {P. partition-on A P ∧ card P ≤ k ∧ (∀ X∈P. card X = 1)} = 1
proof –
  {
    fix P
    assume partition-on A P ∀ X∈P. card X = 1
    from this have P ∈ {P. partition-on A P ∧ (∀ X∈P. card X = 1)} by simp
```

  from *this* **have** $P \in \{(\lambda a.\ \{a\})\ `\ A\}$
    using *partition-on-size1* ‹*finite A*› **by** *auto*
  from *this* **have** $P = (\lambda a.\ \{a\})\ `\ A$ **by** *auto*
  moreover from *this* **have** *card P* = *card A*
    by (*auto intro*: *card-image*)
 }
 from *this* **have** $\{P.\ \textit{partition-on}\ A\ P \wedge \textit{card}\ P \leq k \wedge (\forall X{\in}P.\ \textit{card}\ X = 1)\} = \{P.\ \textit{partition-on}\ A\ P \wedge (\forall X{\in}P.\ \textit{card}\ X = 1)\}$
  using ‹*card A* ≤ *k*› **by** *auto*
 from *this* **show** *?thesis*
  using ‹*finite A*› **by** (*simp only*: *card-partition-on-size1*)
qed

lemma *card-partition-on-size1-eq-0*:
 assumes *finite A*
 assumes *k* < *card A*
 shows $\textit{card}\ \{P.\ \textit{partition-on}\ A\ P \wedge \textit{card}\ P \leq k \wedge (\forall X{\in}P.\ \textit{card}\ X = 1)\} = 0$
proof −
 {
  fix *P*
  assume $\textit{partition-on}\ A\ P\ \forall X{\in}P.\ \textit{card}\ X = 1$
  from *this* **have** $P \in \{P.\ \textit{partition-on}\ A\ P \wedge (\forall X{\in}P.\ \textit{card}\ X = 1)\}$ **by** *simp*
  from *this* **have** $P \in \{(\lambda a.\ \{a\})\ `\ A\}$
    using *partition-on-size1* ‹*finite A*› **by** *auto*
  from *this* **have** $P = (\lambda a.\ \{a\})\ `\ A$ **by** *auto*
  from *this* **have** *card P* = *card A*
    by (*auto intro*: *card-image*)
 }
 from *this assms*(*2*) **have** $\{P.\ \textit{partition-on}\ A\ P \wedge \textit{card}\ P \leq k \wedge (\forall X{\in}P.\ \textit{card}\ X = 1)\} = \{\}$
  using *Collect-empty-eq leD* **by** *fastforce*
 from *this* **show** *?thesis* **by** (*simp only*: *card.empty*)
qed

end

References

[1] F. Haftmann. Discrete summation. *Archive of Formal Proofs*, Apr. 2014. http://isa-afp.org/entries/Discrete_Summation.shtml, Formal proof development.

[2] R. P. Stanley. *Enumerative Combinatorics: Volume 1*. Cambridge University Press, second edition, 2012.

[3] Wikipedia. Stirling numbers of the second kind — wikipedia, the free encyclopedia, 2015. https://en.wikipedia.org/w/index.php?

title=Stirling_numbers_of_the_second_kind&oldid=693800357, [Online; accessed 12-December-2015].