Abstract

This entry defines the Bell numbers [1] as the cardinality of set partitions for a carrier set of given size, and derives Spivey’s generalized recurrence relation for Bell numbers [2] following his elegant and intuitive combinatorial proof.

As the set construction for the combinatorial proof requires construction of three intermediate structures, the main difficulty of the formalization is handling the overall combinatorial argument in a structured way. The introduced proof structure allows us to compose the combinatorial argument from its subparts, and supports to keep track how the detailed proof steps are related to the overall argument. To obtain this structure, this entry uses set monad notation for the set construction’s definition, introduces suitable predicates and rules, and follows a repeating structure in its Isar proof.

Contents

1 Cardinality of Set Partitions
   1.1 Insertion of Elements into Set Partitions . . . . . . . . . . . . 2
   1.2 Cardinality of Set Partitions . . . . . . . . . . . . . . . . . . 6

2 Set Partitions
   2.1 Useful Additions to Main Theories . . . . . . . . . . . . . . . . 12
   2.2 Introduction and Elimination Rules . . . . . . . . . . . . . . . 12
   2.3 Basic Facts on Set Partitions . . . . . . . . . . . . . . . . . . 12
   2.4 The Unique Part Containing an Element in a Set Partition . . 13
   2.5 Cardinality of Parts in a Set Partition . . . . . . . . . . . . 15
   2.6 Operations on Set Partitions . . . . . . . . . . . . . . . . . . 16

3 Bell Numbers and Spivey’s Generalized Recurrence
   3.1 Preliminaries . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
      3.1.1 Additions to FuncSet . . . . . . . . . . . . . . . . . . . . . 20
      3.1.2 Additions for Injectivity Proofs . . . . . . . . . . . . . 20
      3.1.3 Disjointness under Function Application . . . . . . . . 22
      3.1.4 Cardinality Theorems for Set.bind . . . . . . . . . . . 22
1 Cardinality of Set Partitions

theory Card-Partitions

imports ~~/src/HOL/Library/Stirling
~~/src/HOL/Library/Disjoint-Sets

begin

1.1 Insertion of Elements into Set Partitions

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

lemma partition-on-Diff:
  assumes P: partition-on A P shows Q \subseteq P \implies partition-on (A - \bigcup Q) (P - Q)
using P P \[\text{THEN partition-onD4}\] by (auto simp: partition-on-def disjoint-def)

lemma partition-on-UN:
  assumes A: partition-on A B and B: \( \forall b. b \in B \implies \text{partition-on} b (P b) \)
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
  show \{\} \notin (\bigcup i \in B. P i)
  using B[THEN partition-onD3] by simp
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 disjnt p q
proof cases
    assume i = j then show \?thesis
    using i j \( p \neq q \) B[THEN partition-onD2, of i] by (auto simp: pairwise-def)
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


lemma partition-on-insert:
  partition-on \(A \ B\) \(\implies\) disjnt \(A \ A'\ \implies\) \(A' \neq \{}\) \(\implies\) partition-on \((A \cup A')\) (insert \(A' \ B\))
by (auto simp: partition-on-def disjoint-def disjnt-def)


lemma partition-on-insert-rewrite1:
  assumes \(a\) \(\notin\) \(A\)
  assumes \(A\) finite \(A\)
  shows \(\{P.\ partition-on\ (insert\ a\ A)\ P \land\ card\ P = Suc\ k \land\ \{a\} \in\ P\} = (\lambda P.\ insert\ \{a\}\ P) \cdot\ \{P.\ partition-on\ A\ P \land\ card\ P = k\}\)
(is \(?S = ?T\))
proof
  {  
    fix \(P\)
    assume partition-on: partition-on (insert \(a\) \(A\)) \(P\)
    and card: card \(P\) = Suc \(k\)
    and mem: \(\{a\} \in\ P\)
    from card mem have card \((P - \{\{a\}\})\) = \(k\)
    by (auto intro: card-ge-0-finite)
    moreover have partition-on \(A\ (P - \{\{a\}\})\)
      using partition-on-insert[of \(A\ P\) \(\{a\}\)] by (auto simp: disjnt-def)
    ultimately have card \((P - \{\{a\}\})\) = Suc \(k\) partition-on \(A\ (P - \{\{a\}\})\).
  }
  note \(P\)-without-\(a\) = this
  show \(?S \subseteq \ ?T\)
  proof
    fix \(p\)
    assume \(p\) \(\in\ \{P.\ partition-on\ (insert\ a\ A)\ P \land\ card\ P = Suc\ k \land\ \{a\} \in\ P\}\)
    from \(P\)-without-\(a\) this show \(p\) \(\in\ insert\ \{a\}\ \cdot\ \{P.\ partition-on\ A\ P \land\ card\ P = k\}\)
      by (intro image-eqI[where \(x = p - \{\{a\}\}\]) fast+
  qed
next
  {  
    fix \(P\)
    assume \(p\): partition-on \(A\ P\)
    assume \(c\): card \(P\) = \(k\)
    from \(a\ p\) have not-mem: \(\{a\} \notin\ P\)
      unfolding partition-on-def by auto
    from \(p \ A\) have finite \(P\) by (auto intro: finite-elements)
    from \(a\ c\) not-mem this have card (insert \(a\) \(P\)) = Suc \(k\)
      by (simp add: card-insert)
    moreover from \(a\ p\) have partition-on (insert \(a\) \(A\)) (insert \(\{a\}\) \(P\))
      using partition-on-insert[of \(A\ P\) \(\{a\}\)] by (auto simp: disjnt-def)
    ultimately have card (insert \(a\) \(P\)) = Suc \(k\) partition-on (insert \(a\) \(A\)) (insert
\[
\{a\} \in P.
\]  
\{from this show \( T \subseteq S \) by auto\}  
qed

**lemma** partition-on-insert-rewrite2:
\[\text{assumes } a \notin A\]
\[\text{shows } \{P, \text{ partition-on } (\text{insert } a A) P \land \text{card } P = \text{Suc } k \land \{a\} \notin P\} = \bigcup ((\lambda P. (\lambda p. \text{insert } (a p) (P - \{p\})) \cdot P) \cdot \{P, \text{ partition-on } A P \land \text{card } P = \text{Suc } k\})(is \ ?S = \ ?T)\]
\[\text{proof}\]
\{
fix \(P\)
assume \(p: \text{partition-on} (\text{insert } a A) P\)
assume \(c: \text{card } P = \text{Suc } k\)
assume \(a: \{a\} \notin P\)
from \(p\) obtain \(p\) where \(p:\text{-def}\) \(p: P a: p\)
unfolding partition-on-def by blast
from \(p\) \(\text{-def}\) have \(a:\text{-notmem}\): \(\forall p' \in P - \{p\}. a \notin p'\)
unfolding partition-on-def disjoint-def by blast
from \(p\) \(\text{-def}\) have \(p': P - \{a\} \notin P\)
unfolding partition-on-def disjoint-def
by (metis Diff-insert-absorb Diff-subset inf.orderE mk-disjoint-insert)
let \(?P' = \text{insert } (p - \{a\}) (P - \{p\})\)
from \(c\) have \(j: \text{finite } P\) by (simp add: card-ge-0-finite)
from \(c\) \(\text{-def}\) have \(?P' = \text{Suc } (\text{card } (P - \{p\}) - \{p - \{a\}\}))\)
by (simp add: card-insert)
also from \(j\) have \(j = \text{Suc } (\text{card } (P - \{p\}))\)
by (subst card-Diff-singleton-if) (simp add: \(p'\)+)
also from \(c\) \(\text{-def}\) have \(j = \text{Suc } k\)
by (subst card-Diff-singleton) (simp add: \(p\)-def)+
finally have \(2: \text{card } ?P' = \text{Suc } k\).
from \(p'\) \(\text{-def}\) have \(P = \text{insert } (\text{insert } a (P - \{a\}) (P - \{p\}))\)
by (simp add: insert-absorb)
from \(\text{this have}\) \(3: P \in (\lambda p. \text{insert } (\text{insert } a p) (?P' - \{p\})) \cdot (\text{insert } (p - \{a\}) (P - \{p\}))\)
by simp
have \(1: \text{partition-on } A (\text{insert } (p - \{a\}) (P - \{p\}))\)
proof -
from \(p\) have \(\{\} \notin P\)
unfolding partition-on-def by auto
from \(\text{this p-def a have}\) \(1: \{\} \notin \text{insert } (p - \{a\}) (P - \{p\})\)
using subset-singletonD[of \(p a\)] by auto
from \(p\) have \(\bigcup P = \text{insert } a A\)
unfolding partition-on-def by auto
from \(p\) \(\text{-def}\) this \(\text{assms a-notmem have}\) \(2: \bigcup \text{insert } (p - \{a\}) (P - \{p\}) = A\)
by auto
from \(p\) \(\text{-def}\) a-notmem have \(3: \forall p \in \text{insert } (p - \{a\}) (P - \{p\}). \forall p' \in \text{insert}
\[(p - \{a\}) (P - \{p\}), \ p_a \neq p' \rightarrow p_a \cap p' = \{\}\]

unfolding partition-on-def disjoint-def by (metis disjoint-iff-not-equal insert-Diff insert-iff)

from 1 2 3 show \(\text{thesis}\) unfolding partition-on-def disjoint-def by simp
qed

from 1 2 3 have \(\exists P'. \ \text{partition-on} \ A \ P' \land \text{card} \ P' = \text{Suc} \ k \land P \in (\lambda p. \text{insert} (\text{insert} a \ p) (P' - \{p\})) \cdot P'\) by blast

} from this show \(\text{thesis}\) unfolding partition-on-def disjoint-def by simp
next

\{ fix \(P \ p\)
assume partition-on: \text{partition-on} \ A \ P
assume c: \text{card} \ P = \text{Suc} \ k
assume p: \(p \in P\)
from partition-on \(p\) assms have have \(p2\): \(a \notin \{\} \ a \notin p \ \text{and} \ \text{non-empty}: \forall p\in P. \ p \neq \{\}\)
and a-notmem: \(\forall p\in P. \ a \notin p \ \text{and} \ \{a\} \notin P\)
unfolding partition-on-def by auto

from partition-on \(p\) have \(\bigcup\ (P - \{p\}) \subseteq A \ p \subseteq A \cup P = A\)
unfolding partition-on-def by auto

from this have \(\text{thesis}\) unfolding partition-on-def disjoint-def by auto
{ fix \(q \ q'\)
assume \(1: q \in \text{insert} (\text{insert} a \ p) (P - \{p\})\)
assume \(2: q' \in \text{insert} (\text{insert} a \ p) (P - \{p\})\)
assume noteq: \(q \neq q'\)
from 1 have \(q \cap q' = \{\}\)
proof
assume \(q: q = \text{insert} a \ p\)
from 2 show \(\text{thesis}\)
proof
assume \(q': q' = \text{insert} a \ p\)
from noteq this q show \(\text{thesis}\) by simp
next
assume \(q': q' \in P - \{p\}\)
from this \(p\) partition-on have \(\forall x\in p. \ x \notin q'\)
unfolding partition-on-def disjoint-def by auto
from this \(q q'\) a-notmem show \(\text{thesis}\) by auto
qed
next
assume \(q: q \in P - \{p\}\)
from 2 show \(\text{thesis}\)
proof
assume \(q': q' = \text{insert} a \ p\)
from \(q\ p\) partition-on have \(\forall x\in p. \ x \notin q\)
unfolding partition-on-def disjoint-def by auto
from this \(q q'\) a-notmem show \(\text{thesis}\) by auto
next
assume \( q' \in P - \{ p \} \)
from \( q \neq q' \) noteq partition-on show ?thesis
unfolding partition-on-def disjoint-def by auto
qed
qed
\}

from this have no-overlap: \( (\forall p \in \text{insert} \ (\text{insert} \ a) \ p) \ (P - \{ p \}) \). \( \forall p' \in \text{insert} \ (\text{insert} \ a) \ p) \ (P - \{ p \}) \). \( p \neq p' \rightarrow p \cap p' = \{ \} \)
by blast
from non-empty Un this have 1: partition-on \( (\text{insert} \ a) \) \( (P - \{ p \}) \)
unfolding partition-on-def disjoint-def by auto
from c p a notmem have 2: card \( (\text{insert} \ (\text{insert} \ a) \ p) \ (P - \{ p \}) \)
by (subst card.\text{insert}) (auto simp add: card-ge-0-finite)
from p2 p a have 3: \( \{ a \} \notin \text{insert} \ (\text{insert} \ a) \ p) \ (P - \{ p \}) \)
by auto
from this show ?thesis by blast
qed

1.2 Cardinality of Set Partitions

theorem card-partition-on:
assumes finite A
shows \( k \) card \( \{ \text{P}. \text{partition-on} \ A \ P \land \text{card} \ P = k \} = \text{Stirling} \ (\text{card} \ A) \ k \)
using assms
proof (induct A arbitrary: \( k \))
case empty
have eq: \( \{ \text{P}. \ P = \{ \} \land \text{card} \ P = 0 \} = \{ \} \)
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: \( \{ \text{P}. \ P = \{ \} \land \text{card} \ P = 0 \} = \{ \} \)
by auto
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 \( ?P1 = \{ \text{P}. \text{partition-on} \ (\text{insert} \ a) \ A \ P \land \text{card} \ P = \text{Suc} \ k \land \{ a \} \in \ P \} \)
let \( ?P2 = \{ \text{P}. \text{partition-on} \ (\text{insert} \ a) \ A \ P \land \text{card} \ P = \text{Suc} \ k \land \{ a \} \notin \ P \} \)
have fin1: \( \land \text{A} \ k. \text{finite} \ A \rightarrow \text{finite} \ \{ \text{P}. \text{partition-on} \ A \ P \land \text{card} \ P = k \} \)
and fin2: \( \land \text{A} \ k. \text{finite} \ A \rightarrow \text{finite} \ \{ \text{P}. \text{partition-on} \ A \ P \land \text{card} \ P = k \land \)
Q P}
  by (simp add: finitely-many-partition-on)+
have finite-elements: finite \( \bigcup (\lambda P. (\lambda p. \text{insert } a \ p) (P - \{p\})) \cup P \)
  using insert(1) by (auto intro: fin1 finite-elements)
from insert(2) have split: \{P. \text{partition-on } (\text{insert } a \ A) \land \text{card } P = \text{Suc } k\}
  \( ?P1 \cup ?P2 \)
  by fast

have inj: inj-on (insert \{a\}) \{P. \text{partition-on } A \ P \land \text{card } P = k\}
proof (rule inj-onI; clarify)
  fix p q
  assume part: \text{partition-on } A \ p \ \text{partition-on } A \ q
  assume i: \text{insert } \{a\} \ p = \text{insert } \{a\} \ q
  from insert(2) part have \{a\} \notin \ p \ \{a\} \notin \ q
    unfolding \text{partition-on-def} by auto
  from this i show p = q by (meson insert-ident)
qed
from insert(1, 2) inj have eq1: card \( ?P1 = \text{Stirling} \ (\text{card } A) \ k \)
  by (simp add: \text{partition-on-insert-rewrite1} \text{card-image insert}(3))

have inj2: \( \forall P. \text{partition-on } A \ P \ \Rightarrow \text{inj-on } (\lambda p. \text{insert } a \ p) (P - \{p\}) \)
proof (rule inj-onI)
  fix P p q
  assume a: \text{partition-on } A \ p
  assume P: \ p \in P \ q \in P
  and eq: \text{insert } (\text{insert } a \ p) (P - \{p\}) = \text{insert } (\text{insert } a \ q) (P - \{q\})
  from insert(2) a have \text{insert } a \ p \notin \ P \ \text{insert } a \ q \notin \ P
    unfolding \text{partition-on-def} by auto
  from this eq P have \text{P - \{p\} = P - \{q\} by \ (metis \ \text{Diff-insert-absorb \ \text{Set.set-insert} insertE insertl2)}
    from this P show p = q by blast
  qed

have inj1: inj-on (\lambda P. (\lambda p. \text{insert } a \ p) (P - \{p\})) \{P. \text{partition-on } A \ P \land \text{card } P = \text{Suc } k\}
proof -
  { fix P Q
    assume partition-on: \text{partition-on } A \ P \ \text{partition-on } A \ Q
    assume card: \text{card } P = \text{Suc } k \ \text{card } Q = \text{Suc } k
    assume eq: (\lambda p. \text{insert } a \ p) (P - \{p\}) \cup P = (\lambda p. \text{insert } a \ p) (Q - \{p\}) \cup Q
    have P = Q
      proof (rule ccontr)
        from partition-on insert(2) have a-notmem: \( \forall p \in P. \ a \notin p \ \forall q \in Q. \ a \notin q \)
          unfolding \text{partition-on-def} by auto
        assume P \neq Q
  qed
from this have \((\exists x. x \in P \land x \notin Q) \lor (\exists x. x \notin P \land x \in Q)\)
by auto

from this have \((\lambda p. \text{insert} \; (\text{insert} \; a \; p) \; (P - \{p\})) \; P \neq (\lambda p. \text{insert} \; (\text{insert} \; a \; p) \; (P - \{p\})) \; Q\)

proof
assume \((\exists x. x \in P \land x \notin Q)\)
from this obtain \(p\) where \(p\): \(p\) \:\(\notin P \land p \in Q\) by auto

from \(p\) this a-notmem have (\(\text{insert} \; (\text{insert} \; a \; p) \; (P - \{p\})\) \(\notin (\lambda p. \text{insert} \; (\text{insert} \; a \; p) \; (P - \{p\})) \; Q\))
by clarify (metis Diff-insert-absorb insertCI insertE insertI1 insert-Diff)

from \(p\) this show \((\lambda p. \text{insert} \; (\text{insert} \; a \; p) \; (P - \{p\})) \; P \neq (\lambda p. \text{insert} \; (\text{insert} \; a \; p) \; (P - \{p\})) \; Q\)
by blast

next
assume \((\exists x. x \notin P \land x \in Q)\)
from this obtain \(q\) where \(q\): \(q\) \:\(\notin Q \land q \in P\) by auto

from this a-notmem have (\(\text{insert} \; (\text{insert} \; a \; q) \; (Q - \{q\})\) \(\notin (\lambda p. \text{insert} \; (\text{insert} \; a \; p) \; (P - \{p\})) \; Q\))
by clarify (metis Diff-insert-absorb insertCI insertE insertI1 insert-Diff)

from \(q\) this show \((\lambda p. \text{insert} \; (\text{insert} \; a \; p) \; (P - \{p\})) \; P \neq (\lambda p. \text{insert} \; (\text{insert} \; a \; p) \; (P - \{p\})) \; Q\)
by blast

qed

from this eq show False by blast

qed

\}
from this show \(?thesis

unfolding inj-on-def by auto

qed

\{\n\fix c
\assume c \in (\lambda P. (\lambda p. \text{insert} \; (\text{insert} \; a \; p) \; (P - \{p\})) \; P) \; \{\; P. \text{partition-on} A P \land \text{card} \; P = \text{Suc} \; k\}\)
from this inj2 have \(\text{card} \; c = \text{Suc} \; k\)
by (auto simp add: card-image)
\} note \(\text{card} = \text{this}\)

\{\n\fix P \; Q
let \(\text{if} = (\lambda P. (\lambda p. \text{insert} \; (\text{insert} \; a \; p) \; (P - \{p\})) \; P) \; \{\; P. \text{partition-on} A P \land \text{card} \; P = \text{Suc} \; k\}\)
assume partition-on: partition-on A P partition-on A Q
assume neg: \(\text{if} \; P \; \text{\(\neq\)}\; \text{if} \; Q \; \text{\(\neq\)}\)
from insert(2) partition-on have \(a: \forall p \in P. \; a \notin p \forall q \in Q. \; a \notin q\)
unfolding partition-on-def by auto

have \(\text{if} \; P \; \cap \; \text{if} \; Q \; \text{\(\neq\)}\; \emptyset\)
proof (rule ccontr)
assume \(\text{if} \; P \; \cap \; \text{if} \; Q \; \text{\(\neq\)}\; \emptyset\)
from this obtain \(q\) where \(q\): \(q\) \:\(\in \text{if} \; P \; \cap \; \text{if} \; Q \; \text{\(\neq\)}\; \emptyset\) by auto

qed
from q(2) obtain p where p: insert a p ∈ q p ∈ Q by auto
from q(1) obtain p' where p': insert a p' ∈ q p' ∈ P by auto
from q p p' a have eq: p = p' by clarify (metis insert-Diff insert-ident
insert-iff)
\[\begin{array}{l}
  \text{from } q \ p \ a \ \text{have } q - \{\text{insert } a \ p\} = P - \{p\} \ q - \{\text{insert } a \ p\} = Q - \{p\} \\
  \text{by (clarify; metis (no-types, hide-lams) Diff-iff Diff-insert-absorb insert-iff)} + \\
  \text{from this } p \ p' \ \text{eq have } P = Q \text{ by auto} \\
  \text{from this neq show False by blast} \\
  \text{qed}
\end{array}\]

\[\begin{array}{l}
  \text{from insert(2) have } \text{card } \{P. \ \text{partition-on } (\text{insert } a \ A) \ P \land \text{card } P = \text{Suc } k \\
  \land \{a\} \notin P\} = \\
  \text{card } \{\bigcup ((\lambda P. (\lambda p. \ \text{insert } (a p) (P - \{p\})) \ P \ P) \ (\text{partition-on } A \ P \land \text{card } P = \text{Suc } k))\} \\
  \text{by (simp add: partition-on-insert-rewrite2)} \\
  \text{also have } \ldots = \text{Suc } k \ast \text{card } \{((\lambda P. (\lambda p. \ \text{insert } a p) (P - \{p\})) \ P \ P) \ (\text{partition-on } A \ P \land \text{card } P = \text{Suc } k)\} \\
  \text{using card insert(1) no-intersect} \\
  \text{by (subst card-partition[axiometric]) (force intro: fin1)+} \\
  \text{also have } \ldots = \text{Suc } k \ast \text{Stirling } \text{(card } A) \ (\text{Suc } k) \\
  \text{using inj1 insert(3) by (subst card-image) auto} \\
  \text{finally have eq2: card } \{P. \ \text{partition-on } (\text{insert } a \ A) \ P \land \text{card } P = \text{Suc } k \land \{a\} \notin P\} = \text{Suc } k \ast \text{Stirling } \text{(card } A) \ (\text{Suc } k). \\
  \text{have } \text{card } \{P. \ \text{partition-on } (\text{insert } a \ A) \ P \land \text{card } P = \text{Suc } k\} = \text{card } ?P1 + \text{card } ?P2 \\
  \text{by (subst split; subst card-Un-disjoint) (auto intro: fin2 insert(1))} \\
  \text{also have } \ldots = \text{Stirling } \text{(card } a \ A) \ (\text{Suc } k) \\
  \text{using insert(1, 2) by (simp add: eq1 eq2)} \\
  \text{finally show } ?\text{thesis using } \text{Suc by auto} \\
  \text{qed}
\end{array}\]

\[\begin{array}{l}
  \text{theorem card-partition-on-at-most-size:} \\
  \text{assumes finite } A \\
  \text{shows card } \{P. \ \text{partition-on } A \ P \land \text{card } P \leq k\} = (\sum j \leq k. \ \text{Stirling } \text{(card } A) \ j) \\
  \text{proof} - \\
  \text{have card } \{P. \ \text{partition-on } A \ P \land \text{card } P \leq k\} = \text{card } \{\bigcup j \leq k. \ (\text{partition-on } A \ P \land \text{card } P = j)\} \\
  \text{by (rule arg-cong[where } f=\text{card]) auto} \\
  \text{also have } \ldots = (\sum j \leq k. \ \text{card } (\text{partition-on } A \ P \land \text{card } P = j)) \\
  \text{by (subst card-Un-disjoint) (auto simp add: finite } A \ \text{finely-many-partition-on}) \\
  \text{also have } (\sum j \leq k. \ \text{card } (\text{partition-on } A \ P \land \text{card } P = j)) = (\sum j \leq k. \ \text{Stirling } \text{(card } A) \ j) \\
  \text{using finite } A \ \text{by (simp add: card-partition-on)} \\
  \text{finally show } ?\text{thesis} . \\
  \text{qed}
\end{array}\]
theorem partition-on-size1:
assumes finite A
shows \(\{ P. \text{partition-on} A P \land (\forall X \in P. \text{card} X = 1)\}\) = \(\{ (\lambda a. \{a\}) \, \cdot \, A\}\)
proof
  show \(\{ P. \text{partition-on} A P \land (\forall X \in P. \text{card} X = 1)\}\) \(\subseteq\) \(\{ (\lambda a. \{a\}) \, \cdot \, A\}\)
  proof
    fix P
    assume P: P \in \{ P. \text{partition-on} A P \land (\forall X \in P. \text{card} X = 1)\}
    have P = (\lambda a. \{a\}) \, \cdot \, A
    proof
      show P \subseteq (\lambda a. \{a\}) \, \cdot \, A
      proof
        fix X
        assume X \in P
        from P this obtain x where X = \{x\}
        by (auto simp add: card-Suc-eq)
        from this \(X \in P\) have x \in A
        using P unfolding partition-on-def by blast
        from this \(X = \{x\}\) show X \in(\lambda a. \{a\}) \, \cdot \, A by auto
      qed
    qed
next
  show (\lambda a. \{a\}) \, \cdot \, A \subseteq P
  proof
    fix X
    assume X \in (\lambda a. \{a\}) \, \cdot \, A
    from this obtain x where X = \{x\} x \in A by auto
    have \(\bigcup P = A\)
    using P unfolding partition-on-def by blast
    from this \(x \in A\) obtain X' where \(x \in X'\) and \(X' \in P\)
    using UnionE by blast
    from \(X' \in P\) have card X' = 1
    using P unfolding partition-on-def by auto
    from this \(x \in X'\) have X' = \{x\}
    using card-1-singletonE by blast
    from this X(1) \(X' \in P\) show X \in P by auto
  qed
  qed
next
  show \((\lambda a. \{a\}) \, \cdot \, A\) \subseteq \{ P. \text{partition-on} A P \land (\forall X \in P. \text{card} X = 1)\}\)
  proof
    fix P
    assume P \in \((\lambda a. \{a\}) \, \cdot \, A\)
    from this have P: P = (\lambda a. \{a\}) \, \cdot \, A by auto
    from this have partition-on A P by (auto intro: partition-onI)
    from P this show P \in \{ P. \text{partition-on} A P \land (\forall X \in P. \text{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 ∈ {[(λa. {a}) ' A]}
      using partition-on-size1 (finite A) by auto
    from this have P = (λa. {a}) ' A by auto
    moreover from this have card P = card A
      by (auto intro: card-image)
    } from this have {P. partition-on A P ∧ card P ≤ k ∧ (∀X∈P. card X = 1)} =
      {P. partition-on A P ∧ (∀X∈P. 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 card {P. partition-on A P ∧ card P ≤ k ∧ (∀X∈P. card X = 1)} = 0
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 ∈ {[(λa. {a}) ' A]}
      using partition-on-size1 (finite A) by auto
    from this have P = (λa. {a}) ' A by auto
    from this have card P = card A
      by (auto intro: card-image)
    } from this assms(2) have {P. partition-on A P ∧ card P ≤ k ∧ (∀X∈P. card
      X = 1)} = {} using Collect-empty-eq leD by fastforce
    from this show ?thesis by (simp only: card-empty)
  qed

11
2 Set Partitions

theory Set-Partition
imports
  ~~/src/HOL/Library/FuncSet
  ../Card-Partitions/Card-Partitions
begin

2.1 Useful Additions to Main Theories

lemma set-eqI:
  assumes \( \forall x. x \in A \implies x \in B \) 
  assumes \( \forall x. x \in B \implies x \in A \) 
  shows \( A = B \) 
  using assms by auto

lemma comp-image:
  \( (\circ f \circ g) = \circ (f \circ g) \) 
  by rule auto

2.2 Introduction and Elimination Rules

The definition of partition-on is in Disjoint-Sets.

lemma partition-onI:
  assumes \( \forall p. p \in P \implies p \neq {} \) 
  assumes \( \bigcup P = A \) 
  assumes \( \forall p, p'. p \in P \implies p' \in P \implies p \neq p' \implies p \cap p' = {} \) 
  shows partition-on \( P \) 
  using assms unfolding partition-on-def disjoint-def by blast

lemma partition-onE:
  assumes partition-on \( P \) 
  obtains \( \forall p. p \in P \implies p \neq {} \) 
  \( \bigcup P = A \) 
  \( \forall p, p'. p \in P \implies p' \in P \implies p \neq p' \implies p \cap p' = {} \) 
  using assms unfolding partition-on-def disjoint-def by blast

2.3 Basic Facts on Set Partitions

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 $A \cap B = \emptyset$
shows $P \cap Q = \emptyset$
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)

An alternative formulation of $\left[ \text{partition-on ?A ?B;} \quad \text{disjnt ?A ?A'}; \quad ?A' \neq \{\} \right] \implies \text{partition-on \ (\ ?A \cup ?A') \ (insert \ ?A'?B)}$

lemma partition-on-insert):
assumes partition-on $(A - X) P$
assumes $X \subseteq A \quad X \neq \{\}$
shows partition-on A $(\text{insert } X P)$
proof
have disjnt $(A - X) X$ by (simp add: disjnt-iff)
from assms(1) this assms(3) have partition-on $((A - X) \cup X) (\text{insert } X P)$
by (rule partition-on-insert)
from this $\langle X \subseteq A \rangle$ show ?thesis
by (metis Diff-partition sup-commute)
qed

2.4 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 $\exists ! X. \ x \in X \land X \in P$
proof
from \langle partition-on A P \rangle have $\bigcup P = A$
by (auto elim: partition-onE)
from this $\langle x \in A \rangle$ obtain X where X: $x \in X \land X \in P$ by blast
\{ fix Y
assume $x \in Y \land Y \in P$
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 $x \in A$

13
shows \((\text{THE } X. \, x \in X \land X \in P) \in P\)

proof –
from \(x \in A\): have \(\exists! X. \, x \in X \land X \in P\)
  using \(\text{partition-on } A \, P\): by (simp add: partition-on-partition-on-unique)
from this show \((\text{THE } X. \, x \in X \land X \in P) \in P\)
  by (metis (no-types, lifting) theI)
qed

lemma partition-on-in-the-unique-part:
assumes \(\text{partition-on } A \, P\)
assumes \(x \in A\)
shows \(x \in (\text{THE } X. \, x \in X \land X \in P)\)
proof –
from assms have \(\exists! X. \, x \in X \land X \in P\)
  by (simp add: partition-on-partition-on-unique)
from this show \(?\text{thesis}\)
  by (metis (mono-tags, lifting) theI')
qed

lemma partition-on-the-part-eq:
assumes \(\text{partition-on } A \, P\)
assumes \(X' \in P\)
shows \((\text{THE } X. \, x \in X \land X \in P) = X'\)
proof –
from \((x \in X) \, (X \in P)\): have \(x \in A\)
  using \(\text{partition-on } A \, P\): by (auto elim: partition-onE)
from this have \(\exists! X. \, x \in X \land X \in P\)
  using \(\text{partition-on } A \, P\): by (simp add: partition-on-partition-on-unique)
from \((x \in X) \, (X \in P)\): this show \((\text{THE } X. \, x \in X \land X \in P) = X\)
  by (auto intro!: the1-equality)
qed

lemma partition-on-all-in-part-eq-part:
assumes \(\text{partition-on } A \, P\)
assumes \(X' \in P\)
shows \(\{x \in A. \,(\text{THE } X. \, x \in X \land X \in P) = X'\} = X'\)
proof
show \(\{x \in A. \,(\text{THE } X. \, x \in X \land X \in P) = X'\} \subseteq X'\)
  using assms(1) \(\text{partition-on-in-the-unique-part}\) by force
next
show \(X' \subseteq \{x \in A. \,(\text{THE } X. \, x \in X \land X \in P) = X'\}\)
proof
  fix \(x\)
  assume \(x \in X'\)
  from \((x \in X') \, (X' \in P)\): have \(x \in A\)
    using \(\text{partition-on } A \, P\): by (auto elim: partition-onE)
  moreover from \((x \in X') \, (X' \in P)\): have \((\text{THE } X. \, x \in X \land X \in P) = X'\)
    using \(\text{partition-on } A \, P\): \(\text{partition-on-the-part-eq}\) by fastforce
  ultimately show \(x \in \{x \in A. \,(\text{THE } X. \, x \in X \land X \in P) = X'\}\) by auto
2.5 Cardinality of Parts in a Set Partition

lemma partition-on-le-set-elements:
  assumes finite A
  assumes partition-on A P
  shows card P ≤ card A
using assms
proof (induct A arbitrary: P)
case empty
from this show card P ≤ card {} by (simp add: partition-on-empty)
next
case (insert a A)
show ?case
proof (cases a ∈ P)
case True
have prop-partition-on:
  ∀ p ∈ P. p ≠ {} ∪ P = insert a A
  ∀ p ∈ P. ∀ p' ∈ P. p ≠ p' ⟹ p ∩ p' = {}
  using (partition-on (insert a A) P) by (fastforce elim: partition-onE)+
  from this 2, 3 have A-eq: A = P − {{a}}
  using ⟨partition-on (insert a A) P⟩ by (intro partition-onI auto)
  from insert.hyps(3) have card (P − {{a}}) ≤ card A by simp
  using finite-elements[OF ⟨finite A; partition-on (insert a A) P⟩] by simp
next
  from ⟨partition-on (insert a A) P⟩ have ∪ P = insert a A
    using ⟨partition-on (insert a A) P⟩ by (fastforce elim: partition-onE)
  let ?P' = insert (p − {{a}}) (P − {p})
  have partition-on A ?P'
    proof (rule partition-onI)
      from ⟨partition-on (insert a A) P⟩ have ∀ p ∈ P. p ≠ {} by (auto elim: partition-onE)
      from this p-def: {{a}} ∉ P, show ∃ p' ∈ insert (p − {{a}}) (P − {p}) = p''
        by (simp; metis (no-types) Diff-eq-empty-iff subset-singletonD)
    next
      from ⟨partition-on (insert a A) P⟩ have P' = insert a A by (auto elim: partition-onE)
    qed
  qed
qed

qed
from p-def this \( \langle a \notin A \rangle \) a-notmem show \( \bigcup \) insert \( (p - \{a\}) (P - \{p\}) = A \)

by auto

next

show \( \bigwedge \) pa pa' \( \in \) insert \( (p - \{a\}) \) \( (P - \{p\}) \) \( \Rightarrow \) pa' \( \in \) insert \( (p - \{a\}) \) \( (P - \{p\}) \)

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)

dqed

have finite P using \( \langle \) finite A \( \rangle \) \( \langle \) partition-on A ?P \( \prime \) \( \rangle \)

by (fastforce

have card P = Suc (card \( \langle \) P - \{p\} \( \rangle \))

using p-def \( \langle \) finite A \( \rangle \) \( \langle \) partition-on A ?P \( \prime \) \( \rangle \)

by (elim: partition-onE)

also have \dots = card ?P' using \( \langle \) p - \{a\} \notin P \( \rangle \) \( \langle \) finite P \( \rangle \) by simp

also have \dots \( \leq \) card A using \( \langle \) partition-on A ?P \( \prime \) \( \rangle \) \( \langle \) insert \( \rho \) hyps(3) \( \rangle \) by simp

also have \dots \( \leq \) card (insert a A) by (simp add: card-insert-le \( \langle \) finite A \( \rangle \))

finally show \?thesis.

dqed

dqed

2.6 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 \( \langle \) partition-on A P \( \rangle \) \( \langle \) partition-on B Q \( \rangle \) show X \( \neq \) \{\}

by (auto elim: partition-onE)

next

show \( \bigcup (P \cup Q) = A \cup B \)

using \( \langle \) partition-on A P \( \rangle \) \( \langle \) partition-on B Q \( \rangle \) 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

dqed

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

16
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 $\text{partition-on } A (P \cup Q)$
  shows $\text{partition-on } (\bigcup Q) Q$
  using assms partition-on-split1 sup-commute by metis

lemma partition-on-intersect-on-elements:
  assumes $\text{partition-on } (A \cup C) P$
  assumes $\forall X \in P. \exists x. x \in X \cap C$
  shows $\text{partition-on } C ((\lambda X. X \cap C) ' P)$

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 $\text{partition-on } B P$
  assumes $f \in A \to_E P$
  shows $\text{partition-on } (A \cup B) ((\lambda X. X \cup \{x \in A. f x = X\}) ' P)$ (is $\text{partition-on } - ?P$)

proof (rule partition-onI)
  fix $X$
  assume $X \in ?P$
  from this $\langle\text{partition-on } B P\rangle$ show $X \neq \{\}$
    by (auto elim: partition-onE)

next
  show $\bigcup ?P = A \cup B$
    using $\langle\text{partition-on } B P\rangle$ $\langle f \in A \to_E P\rangle$ by (auto elim: partition-onE)

next
  fix $X$ $Y$
assume $X \in ?P \ Y \in ?P \ X \neq Y$
from $\langle X \in ?P \rangle$ obtain $X'$ where $X' : X' \cup \{ x \in A.\ f x = X' \} \ X' \in P$ by auto
from $\langle Y \in ?P \rangle$ obtain $Y'$ where $Y' : Y' \cup \{ x \in A.\ f x = Y' \} \ Y' \in P$
by auto
from $\langle X \neq Y \rangle$ $X' \ Y'$ have $X' \neq Y'$ by auto
using $\langle\text{partition-on } B P \rangle$ by (auto elim!: partition-onE)
from $X' \ Y'$ have $X' \subseteq B \ Y' \subseteq B$
using $\langle\text{partition-on } B P \rangle$ by (auto elim!: partition-onE)
from this $\langle X' \cap Y' = \{}\rangle \ X' \ Y' \langle X' \neq Y' \rangle$ show $X \cap Y = \{}$
using $\langle\text{A } B \rangle$ by auto
qed

lemma partition-on-map:
assumes inj-on $f$ $A$
assumes partition-on $A$ $P$
shows partition-on $(f ^{\bigcirc} A)$ $(op ^{\bigcirc} f ^{\bigcirc} P)$
proof -
\{
  \{ fix $X$ $Y$
  assume $X \in P \ Y \in P f ^{\bigcirc} X \neq f ^{\bigcirc} Y$
  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 $f ^{\bigcirc} X \cap f ^{\bigcirc} Y = \{}$
  unfolding inj-on-def by auto (metis IntI empty_iff rev_image_eqI+)
  \}
from assms this show partition-on $(f ^{\bigcirc} A)$ $(op ^{\bigcirc} f ^{\bigcirc} P)$
  by (auto intro!: partition-onI elim!: partition-onE)
qed

lemma set-of-partition-on-map:
assumes inj-on $f$ $A$
shows $op ^{\bigcirc} \{ P.\ \text{partition-on } A P \} = \{ P.\ \text{partition-on } (f ^{\bigcirc} A) P \}$
proof (rule set-eqI')
fix $x$
assume $x \in op ^{\bigcirc} \{ P.\ \text{partition-on } A P \}$
from this (inj-on $f$ $A$) show $x \in \{ P.\ \text{partition-on } (f ^{\bigcirc} A) P \}$
  by (auto intro: partition-on-map)
next
fix $P$
assume $P \in \{ P.\ \text{partition-on } (f ^{\bigcirc} A) P \}$
from this have partition-on $(f ^{\bigcirc} A) P$ by auto
from this have mem: $\forall X.\ X \in P \Rightarrow x \in X \Rightarrow x \in f ^{\bigcirc} A$
  by (auto elim!: partition-onE)
have $op ^{\bigcirc} (f \circ \text{the-inv-into } A f) ^{\bigcirc} P = op ^{\bigcirc} f ^{\bigcirc} \langle op ^{\bigcirc} \text{the-inv-into } A f \rangle ^{\bigcirc} P$
  by (simp add: image-comp comp_image)
moreover have $P = op ^{\bigcirc} (f \circ \text{the-inv-into } A f) ^{\bigcirc} P$
\end{verbatim}
proof \((\text{rule set-eqI}^{'})\)
\[
\begin{align*}
\text{fix } X \\
\text{assume } X \in P \\
\text{moreover from } X \text{ mem have } \text{ in-range: } \forall x \in X, \; x \in f \circ A \text{ by auto} \\
\text{moreover have } X = (f \circ \text{the-inv-into } A f) \cdot X \\
\end{align*}
\]
proof \((\text{rule set-eqI}^{'})\)
\[
\begin{align*}
\text{fix } x \\
\text{assume } x \in X \\
\text{show } x \in (f \circ \text{the-inv-into } A f) \cdot X \\
\end{align*}
\]
proof \((\text{rule image-eqI}^{'})\)
\[
\begin{align*}
\text{from } \text{in-range } \langle x \in X \rangle \text{ assms show } x = (f \circ \text{the-inv-into } A f) x \\
\text{by } (\text{auto simp add: f-the-inv-into-f} [of f]) \\
\text{from } \langle x \in X \rangle \text{ show } x \in X \text{ by assumption} \\
\text{qed}
\end{align*}
\]
next
\[
\begin{align*}
\text{fix } x \\
\text{assume } x \in (f \circ \text{the-inv-into } A f) \cdot X \\
\text{from } \text{this obtain } x' \text{ where } x': x' \in X \land x = f (\text{the-inv-into } A f x') \text{ by auto} \\
\text{from } \text{in-range } x' \text{ have } f: f (\text{the-inv-into } A f x') \in X \\
\text{by } (\text{subst f-the-inv-into-f} [of f]) (\text{auto intro: inj-on f A}) \\
\text{from } x' \langle X \in P \rangle \text{ f show } x \in X \text{ by auto} \\
\text{qed}
\end{align*}
\]
ultimately show \(X \in \text{op } (f \circ \text{the-inv-into } A f) \cdot P \text{ by auto}\)
next
\[
\begin{align*}
\text{fix } X \\
\text{assume } X \in \text{op } (f \circ \text{the-inv-into } A f) \cdot P \\
\text{moreover} \\
\{ \\
\text{fix } Y \\
\text{assume } Y \in P \\
\text{from } \text{this } \langle \text{inj-on } f A \rangle \text{ mem have } \forall x \in Y, \; f (\text{the-inv-into } A f x) = x \\
\text{by } (\text{auto simp add: f-the-inv-into-f}) \\
\text{from } \text{this have } (f \circ \text{the-inv-into } A f) \cdot Y = Y \text{ by force} \\
\}
\text{ultimately show } X \in P \text{ by auto} \\
\text{qed}
\end{align*}
\]
ultimately have \(P: P = \text{op } (f \circ \text{the-inv-into } A f) \cdot P \text{ by simp}\)
have \(A\text{-eq: } A = \text{the-inv-into } A f \cdot f \cdot A \text{ by (simp add: assms)}\)
from \(\langle \text{inj-on } f A \rangle \) have inj-on \((\text{the-inv-into } A f) (f \cdot A)\)
using \((\text{partition-on } (f \cdot A) P) \) by (simp add: inj-on-the-inv-into)
from this have \(op \cdot (\text{the-inv-into } A f) \cdot P \in \{P, \text{partition-on } A P\}\)
using \((\text{partition-on } (f \cdot A) P) \) by (subst A-eq, auto intro!: partition-on-map)
from \(P \) this show \(P \in \text{op } \{(op \cdot f \cdot P) \cdot \{P, \text{partition-on } A P\}\} \text{ by (rule image-eqI)}\)
qed
end
imports Pure
keywords
  method :: thy-decl and
  conclusion
  declares
  methods
  | ⇒
  uses
begin

ML-file parse-tools.ML
ML-file method-closure.ML
ML-file eisbach-rule-insts.ML
ML-file match-method.ML

method solves methods m = (m; fail)

end

3 Bell Numbers and Spivey’s Generalized Recurrence

theory Bell-Numbers
imports
  Binomial
  ~/src/HOL/Library(FuncSet
  ~/src/HOL/Library(Monad-Syntax
  ~/src/HOL/Library(Stirling
  ../Card-Number-Partitions/Additions-to-Main
  Set-Partition
  ~/src/HOL/Eisbach/Eisbach
begin

3.1 Preliminaries

3.1.1 Additions to FuncSet

lemma extensional-funcset-ext:
  assumes f ∈ A →E B
  g ∈ A →E B
  assumes \( \forall x. x ∈ A ⇒ f x = g x \)
  shows f = g
  using assms by (metis PiE-iff extensionalityI)

3.1.2 Additions for Injectivity Proofs

lemma inj-on-impl-inj-on-image:
  assumes inj-on f A
  assumes \( \forall x. x ∈ X ⇒ x ⊆ A \)
  shows inj-on (op "f") X

20
using assms by (meson inj-onI inj-on-image-eq-iff)

lemma injectivity-union:
  assumes \( A \cup B = C \cup D \)
  assumes \( P A P C \)
  assumes \( Q B Q D \)
  \( \forall S T. \; P S \rightarrow Q T \rightarrow S \cap T = {} \)
  shows \( A = C \land B = D \)
using assms Int-Un-distrib Int-commute inf-sup-absorb by blast+

lemma injectivity-image:
  assumes \( f : A = g : A \)
  assumes \( \forall x \in A. \; invert (f x) = x \land invert (g x) = x \)
  shows \( \forall x \in A. \; f x = g x \)
using assms by (metis (no-types, lifting) image-iff)

lemma injectivity-image-union:
  assumes \( (\lambda X. \; X \cup F X) : P = (\lambda X. \; X \cup G X) : P' \)
  assumes \( \forall X \in P. \; X \subseteq A \; \forall X \in P'. \; X \subseteq A \)
  assumes \( \forall X \in P. \; \forall y \in F X. \; y \notin A \; \forall X \in P'. \; \forall y \in G X. \; y \notin A \)
  shows \( P = P' \)
proof
  show \( P \subseteq P' \)
  proof
    fix \( X \)
    assume \( X \in P \)
    from assms(1) this obtain \( X' \) where \( X' \in P' \) and \( X \cup F X = X' \cup G X' \)
    by (metis imageE image-eqI)
    moreover from assms(2,4) \( \forall X \in P \) have \( X : (X \cup F X) \cap A = X \) by auto
    moreover from assms(3,5) \( \forall X \in P'. \; (X' \cup G X') \cap A = X' \) by auto
    ultimately have \( X = X' \) by simp
    from this \( X' \in P' \) show \( X \in P' \) by auto
  qed
next
  show \( P' \subseteq P \)
  proof
    fix \( X' \)
    assume \( X' \in P' \)
    from assms(1) this obtain \( X \) where \( X \in P \) and \( X \cup F X = X' \cup G X' \)
    by (metis imageE image-eqI)
    moreover from assms(2,4) \( \forall X \in P \) have \( X : (X \cup F X) \cap A = X \) by auto
    moreover from assms(3,5) \( \forall X \in P'. \; (X' \cup G X') \cap A = X' \) by auto
    ultimately have \( X = X' \) by simp
    from this \( X \in P \) show \( X' \in P \) by auto
  qed
qed
3.1.3 Disjointness under Function Application

**definition** disjoint-under :: ('a ⇒ 'b set) ⇒ 'a set ⇒ bool

where
disjoint-under f S = (∀ s∈S. ∀ t∈S. s ≠ t → (f s) ∩ (f t) = {}) 

**lemma** disjoint-underI:
assumes (∀ s t. s ∈ S ∧ t ∈ S ⇒ s ≠ t ⇒ (f s) ∩ (f t) = {})
shows disjoint-under f S 
using assms unfolding disjoint-under-def by auto 

**lemma** disjoint-singleton: (∀ s t X Y. s ≠ t ⇒ (X = Y ⇒ s = t) ⇒ {X} ∩ {Y} = {})
by auto 

**lemma** disjoint-bind: (∀ S T f g. (∀ s t. S s ∧ T t ⇒ f s ∩ g t = {}) ⇒ ([s. S s] ⇒ f) ∩ ([t. T t] ⇒ g) = {})
by fastforce 

**lemma** disjoint-bind': (∀ S T f g. (∀ s t. s ∈ S ∧ t ∈ T ⇒ f s ∩ g t = {}) ⇒ (S ⇒ f) ∩ (T ⇒ g) = {})
by fastforce 

**lemma** injectivity-solver-CollectE:
assumes a ∈ {x. P x} ∧ a' ∈ {x. P' x}
assumes (P a ∧ P' a') ⇒ W
shows W
using assms by auto 

**lemma** injectivity-solver-prep-assms:
assumes x ∈ {x. P x}
shows P x ∧ P x
using assms by simp

**method** injectivity-solver uses rule =
insert method-facts, use nothing in :
((drule injectivity-solver-prep-assms)+)?; rule disjoint-underI;
(rule disjoint-bind | rule disjoint-bind')?; erule disjoint-singleton;
(elim injectivity-solver-CollectE)?; rule rule;
assumption+

3.1.4 Cardinality Theorems for Set.bind

**lemma** finite-bind:
assumes finite S
assumes (∀ x ∈ S. finite (f x)
shows finite \((S \gg f)\)

using assms by (simp add: bind-UNION)

lemma card-bind:
  assumes finite \(S\)
  assumes \(\forall X \in S.\ finite (f X)\)
  assumes disjoint-under \(f\) \(S\)
  shows \(card (S \gg f) = (\sum x \in S.\ card (f x))\)

proof
  have \(card (S \gg f) = card (\bigcup (f ' S))\)
    by (simp add: bind-UNION)
  also have \(card (\bigcup (f ' S)) = (\sum x \in S.\ card (f x))\)
    using assms unfolding disjoint-under-def
    by (subst card-Union-image simp+)
  finally show ?thesis.
qed

lemma card-bind-constant:
  assumes finite \(S\)
  assumes \(\forall X \in S.\ finite (f X)\)
  assumes disjoint-under \(f\) \(S\)
  assumes \(\forall x.\ x \in S \Rightarrow card (f x) = k\)
  shows \(card (S \gg f) = card S * k\)
  using assms by (simp add: card-bind)

3.2 Definition of Bell Numbers

definition Bell :: \(nat \Rightarrow nat\)
where
  \(Bell n = card \{ P.\ partition-on \{0..<n\} P\}\)

Show that definition holds for any set \(A\) with cardinality \(n\)

lemma Bell-altdef:
  assumes finite \(A\)
  shows \(Bell (card A) = card \{ P.\ partition-on A P\}\)

proof
  from \((finite A)\) obtain \(f\) where bij: bij-betw \(f\) \(\{0..<\text{card } A\} A\)
    using ex-bij-betw-nat-finite by blast
  from this have inj: inj-on \(f\) \(\{0..<\text{card } A\}\)
    using bij-betw-imp-inj-on by blast
  from bij have image-f-eq: \(A = f ' \{0..<\text{card } A\}\)
    using bij-betw-imp-surj-on by blast
  have \(\forall x \in \{ P.\ partition-on \{0..<\text{card } A\} P\}.\ x \subseteq \text{Pow } \{0..<\text{card } A\}\)
    by (auto elim: partition-onE)
  from this inj have inj-on \((op ' (op ' f))\) \(\{ P.\ partition-on \{0..<\text{card } A\} P\}\)
    by (intro inj-on-impl-inj-on-image[of - \text{Pow } \{0..<\text{card } A\}])
    blast+
  moreover from inj have \((op ' (op ' f)) ' \{ P.\ partition-on \{0..<\text{card } A\} P\}\)
    = \{ P.\ partition-on A P\}
ultimately have bij-betw (op (op ' f)) {P. partition-on {0..< card A} P} {P. partition-on A P}
  by (auto intro: bij-betw-imageI)
from this {finite A} show thesis
 unfold Bell-def
  by (subst bij-betw-iff-card [symmetric]) (auto intro: finitely-many-partition-on)
qed

lemma Bell-0:
  Bell 0 = 1
by (auto simp add: Bell-def partition-on-empty)

3.3 Construction of the Partitions

definition construct-partition-on :: 'a set ⇒ 'a set ⇒ 'a set set set
where
  construct-partition-on B C =
    do
      k ← {0..card B};
      j ← {0..card C};
      P ← {P. partition-on C P ∧ card P = j};
      B' ← {B'. B' ⊆ B ∧ card B' = k};
      Q ← {Q. partition-on B' Q};
      f ← (B − B') → E P;
      P' ← {{λX. X ∪ {x ∈ B − B'. f x = X}} ' P};
{P'∪ Q}
    
lemma construct-partition-on:
  assumes finite B finite C
  assumes B ∩ C = {}
  shows construct-partition-on B C = {P. partition-on (B ∪ C) P}
proof (rule set-eqI)
  fix Q'
  assume Q' ∈ construct-partition-on B C
from this obtain j k P P' Q B' f
  where j ≤ card C
  and k ≤ card B
  and P: partition-on C P ∧ card P = j
  and B': B' ⊆ B ∧ card B' = k
  and Q: partition-on B' Q
  and f: f ∈ B − B' → E P
  and P': P' = (λX. X ∪ {x ∈ B − B'. f x = X}) ' P
  and Q': Q' = P' ∪ Q
  unfolding construct-partition-on-def by auto
from P f have partition-on (B − B' ∪ C) P'
  unfolding P' using {B ∩ C = {}};
  by (intro partition-on-insert-elements) auto
from this Q have partition-on \(((B - B' \cup C) \cup B')\) \(Q'\)
unfolding \(Q'\) using \(B' \cap C = \{\}\) by (auto intro: partition-on-union)
from this have partition-on \((B \cup C)\) \(Q'\)
using \(B'\) by (metis Diff-partition sup.assoc sup.commute)
from this show \(Q' \in \{P. \text{partition-on } (B \cup C) P\}\) by auto

next
fix \(Q'\)
assume \(Q'\): \(Q' \in \{Q'. \text{partition-on } (B \cup C) Q'\}\)
from \(Q'\) have \(\{\} \notin Q'\) by (auto elim!: partition-onE)

obtain \(Q\) where \(Q\): \(Q = ((\lambda x. \text{if } x \subseteq B \text{ then } x \text{ else } \{\} \cdot Q') - \{\} )\) by blast

obtain \(P'\) where \(P'\): \(P' = ((\lambda x. \text{if } x \subseteq B \text{ then } \{\} \text{ else } x) \cdot Q') - \{\} \) by blast
from \(P'\) \(Q\) \(\{\}\) \(\notin Q'\) have \(Q'\)-prop: \(Q' = P'\cup Q\) by auto
have \(P'\)-nosubset: \(\forall X \in P'. \neg X \subseteq B\)
unfolding \(P'\) by auto
moreover have \(\forall X \in P'. X \subseteq B \cup C\)
using \(Q'\) \(P'\) by (auto elim: partition-onE)
ultimately have \(P'\)-witness: \(\forall X \in P'. \exists x. x \in X \cap C\)
using \(B \cap C = \{\}\) by fastforce
obtain \(B'\) where \(B'\): \(B' = \bigcup Q\) by blast
have \(Q\)-prop: partition-on \(B'\) \(Q\)
using \(B'\) \(Q'\) \(Q'\)-prop partition-on-split2 mem-Collect-eq by blast
have \(\bigcup P' = B - B' \cup C\)

proof
have \(\bigcup Q' = B \cup C \forall X \in Q'. \forall X' \in Q'. X \neq X' \longrightarrow X \cap X' = \{\}\)
using \(Q'\) unfolding partition-on-def disjoint-def by auto
from this show \(\bigcup P' \subseteq B - B' \cup C\)
unfolding \(P'\) \(B'\) \(Q\) by auto blast

next
show \(B - B' \cup C \subseteq \bigcup P'\)
proof
fix \(x\)
assume \(x \in B - B' \cup C\)
from this obtain \(X\) where \(X\): \(x \in X X \in Q'\)
using \(Q'\) by (metis Diff-iff Un-iff mem-Collect-eq partition-on-partition-on-unique)
have \(\forall X \in Q'. X \subseteq B \longrightarrow X \subseteq B'\)
unfolding \(B'\) \(Q\) by auto
from this \(X\) \(x \in B - B' \cup C\) have \(\neg X \subseteq B\)
using \(B \cap C = \{\}\) by auto
from this \(X \in Q'\) have \(X \in P'\) using \(P'\) by auto
from this \(x \in X\) show \(x \in \bigcup P'\) by auto
qed

qed
from this have partition-on-\(P'\): partition-on \((B - B' \cup C)\) \(P'\)
using partition-on-split1 \(Q'\)-prop \(Q'\) mem-Collect-eq by fastforce
obtain \(P\) where \(P\): \(P = (\lambda x. X \cap C) \cdot P'\) by blast
from \(P\) partition-on-\(P'\) \(P'\)-witness have partition-on \(C\) \(P\)
using partition-on-intersect-on-elements by auto
obtain \(f\) where \(f\): \(f = (\lambda x. \text{if } x \in B - B' \text{ then } (\text{THE } X. x \in X \wedge X \in P') \cap\)
have $P'$-prop: $P' = (\lambda X. X \cup \{x \in B - B', f x = X\}) \cdot P$

proof

{  
  fix $X$
  assume $X \in P'$
  have $X$-subset: $X \subseteq (B - B') \cup C$
  
  using partition-on-$P'$ $(\forall x \in P')$ by (auto elim: partition-onE)
  have $X = X \cap C \cup \{x \in B - B', f x = X \cap C\}$

proof

{  
  fix $x$
  assume $x \in X$
  
  from this $X$-subset have $x \in (B - B') \cup C$ by auto
  
  from this have $x \in X \cap C \cup \{xa \in B - B', f xa = X \cap C\}$

proof

  assume $x \in C$
  from this ($x \in X$) show $\neg\text{thesis by simp}$

next

  assume $x \in B - B'$
  
  from partition-on-$P'$ $(\forall x \in X) (\forall X \in P')$ have $(\forall x \in X \cap X \in P') = X$

  by (simp add: partition-on-the-part-eq)
  
  from $(\forall x \in B - B')$ this show $\neg\text{thesis unfolding } f \text{ by auto}$

qed

}  

from this show $X \subseteq X \cap C \cup \{x \in B - B', f x = X \cap C\}$ by auto

next

show $X \cap C \cup \{xa \in B - B', f xa = X \cap C\} \subseteq X$

proof

  fix $x$
  
  assume $x \in X \cap C \cup \{x \in B - B', f x = X \cap C\}$

from this show $x \in X$

proof

  assume $x \in X \cap C$

from this show $\neg\text{thesis by simp}$

next

assume $x$-in: $x \in \{x \in B - B', f x = X \cap C\}$

from this have $\exists ! X. x \in X \cap X \in P'$

  using partition-on-$P'$ by (auto intro!: partition-on-partition-on-unique)

  from $x$-in $X$-subset have eq: $(\forall X. x \in X \cap X \in P') \cap C = X \cap C$

  unfolding $f$ by auto

from $P'$-no-subset $\forall X \in P'$ have $\neg X \subseteq B$ by simp

from this have $X \cap C \neq {}$

  using $X$-subset assms(3) by blast

from this obtain $y$ where $y: y \in X \cap C$ by auto

from this eq have $y$-in: $y \in (\forall X. x \in X \cap X \in P') \cap C$ by simp

from $y$ $y$-in have $y \in X \cap X \in (\forall X. x \in X \cap X \in P')$ by auto

moreover from $y$ have $\exists ! X. y \in X \cap X \in P'$

26
using partition-on-P' by (simp add: partition-on-partition-on-unique)
moreover have (THE. x ∈ X ∧ X ∈ P') ∈ P'
  using ex1 by (rule thel2) auto
ultimately have (THE. x ∈ X ∧ X ∈ P') = X using X ∈ P' by auto
from this ex1 show ?thesis by (auto intro: thel2)
qed
qed
from ⟨X ∈ P'⟩ this have X ∈ (λX. X ∪ {x ∈ B − B', f x = X}) ' P
unfolding P by simp
from this show P' ⊆ (λX. X ∪ {x ∈ B − B', f x = X}) ' P ..
next
{ fix x
assume x-in-image: x ∈ (λX. X ∪ {x ∈ B − B', f x = X}) ' P
{ fix X
assume X ∈ P'
have {x ∈ B − B', f x = X ∩ C} = {x ∈ B − B', x ∈ X}
proof −
{ fix x
assume x ∈ B − B'
from this have ex1: ∃!X. x ∈ X ∧ X ∈ P'
  using partition-on-P' by (auto intro: partition-on-partition-on-unique)
from this have in-p: (THE. x ∈ X ∧ X ∈ P') ∈ P'
  and x-in: x ∈ (THE. x ∈ X ∧ X ∈ P')
  by (metis (mono-tags, lifting) thel1)+
have f x = X ∩ C ⟷ (THE. x ∈ X ∧ X ∈ P') ∩ C = X ∩ C
  using (x ∈ B − B') unfolding f by auto
also have ... ⟷ (THE. x ∈ X ∧ X ∈ P') = X
proof
  assume (THE. x ∈ X ∧ X ∈ P') = X
  from this show (THE. x ∈ X ∧ X ∈ P') ∩ C = X ∩ C by auto
next
assume (THE. x ∈ X ∧ X ∈ P') ∩ C = X ∩ C
have (THE. x ∈ X ∧ X ∈ P') ∩ X ≠ {} by (metis P'-witness ((THE. x ∈ X ∧ X ∈ P') ∩ C = X ∩ C) ⟨X ∈ P'⟩ by fastforce
from this show (THE. x ∈ X ∧ X ∈ P') = X
using partition-on-P'[unfolded partition-on-def disjoint-def] in-p ⟨X ∈ P'⟩ by metis
qed
also have ... ⟷ x ∈ X
using ex1 'X ∈ P' x-in by (auto; metis (no-types, lifting) the-equality)
finally have f x = X ∩ C ⟷ x ∈ X .
} }
from this show \( \text{thesis by auto} \)

qed

moreover have \( X \subseteq B - B' \cup C \)

using partition-on-\(P'\) \(X \in P'\) by (blast elim: partition-onE)

ultimately have \( X \cap C \cup \{ x \in B. \, x \notin B' \land f x = X \cap C \} = X \) by auto

} from this \( x \text{-in-image} \) have \( x \in P' \) unfolding \( P \) by auto

} from this show \( (\lambda X. \, X \cup \{ x \in B - B', \, f x = X \}) \quad \forall \quad P \subseteq P' \) ..

qed

from partition-on-\(P'\) have \( f \text{-prop}: f \in (B - B') - E P \)

unfolding \( f \) \( P \) by (auto simp add: partition-on-the-part-mem)

from \( Q \) \( B' \) have \( B' \subseteq B \) by auto

obtain \( k \) where \( k: \, k = \text{card} \ B' \) by blast

from \( \text{finite} \ B \) \( \{B' \subseteq B\} \) have \( k \text{-prop}: \, k \in \{0...\text{card} \ B\} \) by (simp add: card-mono)

obtain \( j \) where \( j: \, j = \text{card} \ P \) by blast

from \( j \) (partition-on \( C \) \( P \)) have \( j \text{-prop}: \, j \in \{0...\text{card} \ C\} \)

by (simp add: assms(2) partition-on-le-set-elements)

from (partition-on \( C \) \( P \), \( j \) have \( P \text{-prop}: \, \text{partition-on} \ C \ P \land \text{card} \ P = j \) by auto

from \( k \) \( \{B' \subseteq B\} \) have \( B' \text{-prop}: \, B' \subseteq B \land \text{card} \ B' = k \) by auto

show \( Q' \in \text{construct-partition-on} \ B \ C \)

using \( j \text{-prop} \ k \text{-prop} \ P' \text{-prop} \ Q \text{-prop} \ P' \text{-prop} \ Q' \text{-prop} \)

unfolding construct-partition-on-def

by (auto simp del: atLeastAtMost-iff) blast

qed

3.4 Injectivity of the Set Construction

lemma injectivity:

assumes \( B \cap C = \{\} \)

assumes \( P: \, (\text{partition-on} \ C \ P \land \text{card} \ P = j) \land (\text{partition-on} \ C \ P' \land \text{card} \ P' = j') \)

assumes \( B'': \, (B' \subseteq B \land \text{card} \ B' = k) \land (B'' \subseteq B \land \text{card} \ B'' = k') \)

assumes \( Q: \, \text{partition-on} \ B' \ Q \land \text{partition-on} \ B'' \ Q' \)

assumes \( f: \, f \in B - B' - E P \land g \in B - B'' - E P' \)

assumes \( P': \, P'' \in \{\lambda X. \, X \cup \{x \in B - B', \, f x = X\} \cdot P\} \land \)

\( P''' \in \{\lambda X. \, X \cup \{x \in B - B'', \, g x = X\} \cdot P'\} \)

assumes eq-result: \( P'' \cup Q = P''' \cup Q' \)

shows \( f = g \) and \( Q = Q' \) and \( B' = B'' \)

and \( P = P' \) and \( j = j' \) and \( k = k' \)

proof –

have \( P \text{-nonempty-sets}: \forall X \in P. \, \exists c \in C. \, c \in X \quad \forall x \in P'. \, \exists c \in C. \, c \in X \)

using \( P \) by (force elim: partition-onE)+

have \( 1: \, \forall X \in P''. \, \exists c \in C. \, c \in X \quad \forall x \in P'''. \, \exists c \in C. \, c \in X \)

using \( P'' \ P \text{-nonempty-sets} \) by fastforce+

have \( 2: \, \forall X \in Q. \, \forall x \in X. \, x \notin C \quad \forall x \in Q'. \, \forall x \in X. \, x \notin C \)

using \( \{B \cap C = \{\}\} \) \( Q \) \( B' \) by (auto elim: partition-onE)

from eq-result have \( P'' = P''' \and Q = Q' \)

28
3.5 The Generalized Bell Recurrence Relation

**Theorem Bell-eq:**

\[ \text{Bell} \ (n + m) = (\sum \text{for } k \leq n \sum \text{for } j \leq m \text{ } j \text{ } \ast \text{ Stirling } m j \ast (n \text{ choose } k) \ast \text{ Bell } k) \]

**Proof**

- **def A:** \{0..<n + m\}
- **def B:** \{0..<n\}
- **def C:** \{n..<n + m\}
have \( A = B \cup C \cap C = \{ \} \) finite \( B \) card \( B = n \) finite \( C \) card \( C = m \)

unfolding \( A \)-def \( B \)-def \( C \)-def by auto

have step1: \( \text{Bell} (n + m) = \text{card} \{ P. \text{partition-on} A P \} \)

unfolding \( \text{Bell} \)-def \( A \)-def ..

from \((A = B \cup C) \land (B \cap C = \{ \}) \land \text{finite} B \land \text{finite} C\)

have step2: \( \text{card} \{ P. \text{partition-on} A P \} = \text{card} \left( \text{construct-partition-on} B C \right) \)

by (simp add: \text{construct-partition-on})

note injectivity = injectivity[OF \( \text{B} \cap C = \{ \} \)]

let \( \text{expr} = \) do {
  \( k \leftarrow \{ 0 .. \text{card} B \} \);
  \( j \leftarrow \{ 0 .. \text{card} C \} \);
  \( P \leftarrow \{ P. \text{partition-on} C P \land \text{card} P = j \} \);
  \( B' \leftarrow \{ B'. \text{B'} \subseteq B \land \text{card} B' = k \} \);
  \( Q \leftarrow \{ Q. \text{partition-on} B' Q \} \);
  \( f \leftarrow (B - B') \rightarrow E P \);
  \( P' \leftarrow \{ (\lambda X. X \cup \{ x \in B - B'. f x = X \}) \cdot P \} \);
  \( \{ P' \cup Q \} \)
}

let \( S \gg \) \( \text{comp} = \text{expr} \)

{ 
  fix \( k \)
  assume \( k: k \in \{ .. \text{card} B \} \)
  let \( \text{expr} = \text{comp} k \)
  let \( S \gg \) \( \text{comp} = \text{expr} \)

  { 
    fix \( j \)
    assume \( j \in \{ .. \text{card} C \} \)
    let \( \text{expr} = \text{comp} j \)
    let \( S \gg \) \( \text{comp} = \text{expr} \)

    from \( \text{finite} C \) have \( \text{finite} S \)
    by (intro \text{finite-Collect-conj} \text{disj1} \text{finitely-many-partition-on})

    { 
      fix \( P \)
      assume \( P: P \in \{ P. \text{partition-on} C P \land \text{card} P = j \} \)
      from this have \( \text{partition-on} C P \) by simp
      let \( \text{expr} = \text{comp} P \)
      let \( S \gg \) \( \text{comp} = \text{expr} \)
      have \( \text{finite} P \)
      using \( P \) \( \text{finite} C \) by (auto intro: \text{finite-elements})
      from \( \text{finite} B \) have \( \text{finite} S \) by (auto simp add: \text{finite-subset})
      moreover
      { 
        fix \( B' \)
        assume \( B': B' \in \{ B'. B' \subseteq B \land \text{card} B' = k \} \)
        from this have \( B' \subseteq B \) by simp
        let \( \text{expr} = \text{comp} B' \)
        let \( S \gg \) \( \text{comp} = \text{expr} \)
        from \( \text{finite} B \) have \( \text{finite} B' \)
        using \( B' \) by (auto simp add: \text{finite-subset})
    }
  }
}

30
from \((\text{finite } B') \text{ have finite } \{Q. \text{ partition-on } B' \ Q\}\) by \((\text{rule finitely-many-partition-on})\)
moreover 
\{
  \begin{align*}
    &\text{fix } Q \\
    &\text{assume } Q: Q \in \{Q. \text{ partition-on } B' \ Q\}\ \\
    &\text{let } ?expr = ?\text{comp } Q \\
    &\text{let } ?S \gg \gg ?\text{comp} = ?\text{expr} \\
    &\{ \\
      &\text{fix } f \\
      &\text{assume } f \in B - B' \to E \ P \\
      &\text{let } ?expr = ?\text{comp } f \\
      &\text{let } ?S \gg \gg \gg ?\text{comp} = ?\text{expr} \\
      &\text{have disjoint-under } ?\text{comp } ?S \\
      &\quad \text{unfolding disjoint-under-def by auto} \\
      &\quad \text{from this have card } ?\text{expr} = 1 \\
      &\quad \quad \text{by } (\text{simp add: card-bind-constant}) \\
      &\quad \text{moreover have finite } ?\text{expr} \\
      &\quad \quad \text{by } (\text{simp add: finite-bind}) \\
      &\quad \text{ultimately have finite } ?\text{expr} \land \text{ card } ?\text{expr} = 1 \text{ by blast} \\
    \}
  \\
  &\text{moreover have finite } ?S \\
  &\quad \text{using } (\text{finite } B') (\text{finite } P) \text{ by } (\text{auto intro: finite-PiE}) \\
  &\text{moreover have disjoint-under } ?\text{comp } ?S \\
  &\quad \text{using } P B' Q \\
  &\quad \text{by } (\text{injectivity-solver rule: local.injectivity(1)}) \\
  &\text{moreover have card } ?S = j \ ^{\ast} (n - k) \\
  \text{proof} - \\
  &\text{have card } (B - B') = n - k \\
  &\quad \text{using } B' (\text{finite } B') (\text{card } B = n) \\
  &\quad \text{by } (\text{subst card-Diff-subset}) \text{ auto} \\
  &\text{from this show } \text{?thesis} \\
  &\quad \text{using } (\text{finite } B) P \\
  &\quad \text{by } (\text{subst card-PiE}) (\text{simp add: prod-constant})+ \\
  \text{qed} \\
  \text{ultimately have card } ?\text{expr} = j ^{\ast} (n - k) \\
  &\text{by } (\text{simp add: card-bind-constant}) \\
  \text{moreover have finite } ?\text{expr} \\
  &\quad \text{using } (?\text{finite } ?S) (\text{finite } \{P. \text{ partition-on } C P \land \text{ card } P = j\}) \\
  &\quad \text{by } (\text{auto intro!: finite-bind}) \\
  \text{ultimately have finite } ?\text{expr} \land \text{ card } ?\text{expr} = j ^{\ast} (n - k) \text{ by blast} \\
\} \text{ note inner = this} \\
\text{moreover have card } ?S = \text{Bell } k \\
&\quad \text{using } B' (\text{finite } B') \text{ by } (\text{auto simp add: Bell-altdef[symmetric]}) \\
\text{moreover have disjoint-under } ?\text{comp } ?S \\
&\quad \text{using } P B' \\
&\quad \text{by } (\text{injectivity-solver rule: local.injectivity(2)}) \\
\text{ultimately have card } ?\text{expr} = j ^{\ast} (n - k) * \text{Bell } k \\
&\quad \text{by } (\text{subst card-bind-constant}) \text{ auto} 
\}
moreover have finite ?expr
using inner (finite ?S) by (auto intro: finite-bind)
ultimately have finite ?expr ∧ card ?expr = \( j \cdot (n - k) \cdot \text{Bell } k \) by blast

} note inner = this
moreover have card ?S = \( n \cdot \text{choose } k \)
using \( \text{card } B = n \cdot \text{finite } B \) by (simp add: n-subsets)
moreover have disjoint-under ?comp ?S
using \( P \)
by (injectivity-solver rule: local.injectivity(3))
ultimately have card ?expr = \( j \cdot (n - k) \cdot (n \text{ choose } k) \cdot \text{Bell } k \)
by (subst card-bind-constant) auto
moreover have finite ?expr
using inner (finite ?S) by (auto intro: finite-bind)
ultimately have finite ?expr ∧ card ?expr = \( j \cdot (n - k) \cdot (n \text{ choose } k) \cdot \text{Bell } k \) by blast

} note inner = this
moreover have finite ?S by simp
moreover have disjoint-under ?comp ?S
by (injectivity-solver rule: local.injectivity(5))
ultimately have card ?expr = \( \sum j \leq m. j \cdot (n - k) \cdot \text{Stirling } m j \cdot (n \text{ choose } k) \cdot \text{Bell } k \) (is - \( = \) ?formula)
using \( \text{card } C = m \) by (subst card-bind) (auto intro: sum.cong)
moreover have finite ?expr
using inner (finite ?S) by (auto intro: finite-bind)
ultimately have finite ?expr ∧ card ?expr = ?formula by blast

} note inner = this
moreover have finite ?S by simp
moreover have disjoint-under ?comp ?S
by (injectivity-solver rule: local.injectivity(6))
ultimately have step3: card (construct-partition-on B C) = \( \sum k \leq n. \sum j \leq m. j \cdot (n - k) \cdot \text{Stirling } m j \cdot (n \text{ choose } k) \cdot \text{Bell } k \)

unfolding construct-partition-on-def
using \( \text{card } B = n \) by (subst card-bind) (auto intro: sum.cong)
from step1 step2 step3 show \?thesis by auto
qed
3.6 Corollaries of the Generalized Bell Recurrence

**corollary** Bell-Stirling-eq:

\[ Bell \ m = (\sum_{j \leq m} Stirling \ m \ j) \]

**proof** –

have \[ Bell \ m = Bell \ (0 + m) \] by simp

also have \[ ... = (\sum_{j \leq m} Stirling \ m \ j) \]

unfolding Bell-eq[of 0] by (simp add: Bell-0)

finally show \[ \text{thesis} \].

qed

**corollary** Bell-recursive-eq:

\[ Bell \ (n + 1) = (\sum_{k \leq n} (n \ choose \ k) \ast Bell \ k) \]

unfolding Bell-eq[of - 1] by simp

end

References
