Spivey's Generalized Recurrence for Bell Numbers

Lukas Bulwahn

March 17, 2025

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 Bell Numbers and Spivey's Generalized Recurrence 1
1.1 Preliminaries . . . 2
1.1.1 Additions to FuncSet . . . 2
1.1.2 Additions for Injectivity Proofs . . . 2
1.2 Definition of Bell Numbers . . . 3
1.3 Construction of the Partitions . . . 4
1.4 Injectivity of the Set Construction . . . 8
1.5 The Generalized Bell Recurrence Relation . . . 9
1.6 Corollaries of the Generalized Bell Recurrence . . . 12
1.7 Code equations for the computation of Bell numbers . . . 13

1 Bell Numbers and Spivey's Generalized Recurrence

theory *Bell-Numbers*
imports
HOL−Library.FuncSet

HOL−Library.Monad-Syntax
HOL−Library.Code-Target-Nat
HOL−Combinatorics.Stirling
Card-Partitions.Injectivity-Solver
Card-Partitions.Card-Partitions
begin

1.1 Preliminaries

1.1.1 Additions to FuncSet

lemma *extensional-funcset-ext*:
  assumes $f \in A \rightarrow_E B$ $g \in A \rightarrow_E B$
  assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x = g\ x$
  shows $f = g$
using *assms* **by** (*metis PiE-iff extensionalityI*)

1.1.2 Additions for Injectivity Proofs

lemma *inj-on-impl-inj-on-image*:
  assumes *inj-on f A*
  assumes $\bigwedge x.\ x \in X \Longrightarrow x \subseteq A$
  shows *inj-on* ((‘) f) X
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 $Q\ B\ Q\ D$
    $\bigwedge S\ T.\ P\ S \Longrightarrow Q\ T \Longrightarrow S \cap T = \{\}$
  shows $A = C \wedge B = D$
using *assms Int-Un-distrib Int-commute inf-sup-absorb* **by** *blast+*

lemma *injectivity-image*:
  assumes f ‘ $A = g$ ‘ A
  assumes $\forall x{\in}A.\ invert\ (f\ x) = x \wedge invert\ (g\ x) = x$
  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 ($metis\ imageE\ image\text{-}eqI$)
moreover from $assms(2,4)$ ‹$X \in P$› **have** X: $(X \cup F\ X) \cap A = X$ **by** $auto$
moreover from $assms(3,5)$ ‹$X' \in P'$› **have** X': $(X' \cup G\ X') \cap A = X'$ **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 ($metis\ imageE\ image\text{-}eqI$)
moreover from $assms(2,4)$ ‹$X \in P$› **have** X: $(X \cup F\ X) \cap A = X$ **by** $auto$
moreover from $assms(3,5)$ ‹$X' \in P'$› **have** X': $(X' \cup G\ X') \cap A = X'$ **by** $auto$
ultimately have $X = X'$ **by** $simp$
from $this$ ‹$X \in P$› **show** $X' \in P$ **by** $auto$
qed
qed

1.2 Definition of Bell Numbers

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

lemma $Bell\text{-}altdef$:
assumes $finite\ A$
shows $Bell\ (card\ A) = card\ \{P.\ partition\text{-}on\ A\ P\}$
proof −
from ‹$finite\ A$› **obtain** f **where** bij: $bij\text{-}betw\ f\ \{0..<card\ A\}\ A$
using $ex\text{-}bij\text{-}betw\text{-}nat\text{-}finite$ **by** $blast$
from $this$ **have** inj: $inj\text{-}on\ f\ \{0..<card\ A\}$
using $bij\text{-}betw\text{-}imp\text{-}inj\text{-}on$ **by** $blast$
from bij **have** $image\text{-}f\text{-}eq$: $A = f\ `\ \{0..<card\ A\}$
using $bij\text{-}betw\text{-}imp\text{-}surj\text{-}on$ **by** $blast$
have $\forall x \in \{P.\ partition\text{-}on\ \{0..<card\ A\}\ P\}.\ x \subseteq Pow\ \{0..<card\ A\}$
by ($auto\ elim$: $partition\text{-}onE$)
from $this$ inj **have** $inj\text{-}on\ ((`)\ ((`)\ f))\ \{P.\ partition\text{-}on\ \{0..<card\ A\}\ P\}$
by ($intro\ inj\text{-}on\text{-}impl\text{-}inj\text{-}on\text{-}image[of\ \text{-}\ Pow\ \{0..<card\ A\}]$
$inj\text{-}on\text{-}impl\text{-}inj\text{-}on\text{-}image[of\ \text{-}\ \{0..<card\ A\}]$) $blast+$
moreover from inj **have** $(`)\ ((`)\ f)\ `\ \{P.\ partition\text{-}on\ \{0..<card\ A\}\ P\} = \{P.\ partition\text{-}on\ A\ P\}$
by ($subst\ image\text{-}f\text{-}eq$, $auto\ elim!$: $set\text{-}of\text{-}partition\text{-}on\text{-}map$)
ultimately have $bij\text{-}betw\ ((`)\ ((`)\ f))\ \{P.\ partition\text{-}on\ \{0..<card\ A\}\ P\}\ \{P.\ partition\text{-}on\ A\ P\}$
by ($auto\ intro$: $bij\text{-}betw\text{-}imageI$)

from *this* ‹*finite A*› **show** *?thesis*
unfolding *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*)

1.3 Construction of the Partitions

definition *construct-partition-on* :: *'a set* ⇒ *'a set* ⇒ *'a set set set*
where
construct-partition-on B C =
do {
$k \leftarrow \{0..card\ B\}$;
$j \leftarrow \{0..card\ C\}$;
$P \leftarrow \{P.\ partition\text{-}on\ C\ P \wedge card\ P = j\}$;
$B' \leftarrow \{B'.\ B' \subseteq B \wedge card\ B' = k\}$;
$Q \leftarrow \{Q.\ partition\text{-}on\ B'\ Q\}$;
$f \leftarrow (B - B') \rightarrow_E P$;
$P' \leftarrow \{(\lambda X.\ X \cup \{x \in B - B'.\ f\ x = X\})\ `\ P\}$;
$\{P' \cup Q\}$
}

lemma *construct-partition-on*:
assumes *finite B finite C*
assumes $B \cap C = \{\}$
shows *construct-partition-on B C* = $\{P.\ partition\text{-}on\ (B \cup C)\ P\}$
proof (*rule set-eqI'*)
fix Q'
assume $Q' \in$ *construct-partition-on B C*
from *this* **obtain** $j\ k\ P\ P'\ Q\ B'\ f$
where $j \leq card\ C$
and $k \leq card\ B$
and P: $partition\text{-}on\ C\ P \wedge card\ P = j$
and B': $B' \subseteq B \wedge card\ B' = k$
and Q: $partition\text{-}on\ B'\ Q$
and f: $f \in B - B' \rightarrow_E P$
and P': $P' = (\lambda X.\ X \cup \{x \in B - B'.\ f\ x = X\})\ `\ P$
and Q': $Q' = P' \cup Q$
unfolding *construct-partition-on-def* **by** *auto*
from *P f* **have** $partition\text{-}on\ (B - B' \cup C)\ P'$
unfolding P' **using** ‹$B \cap C = \{\}$›
by (*intro partition-on-insert-elements*) *auto*
from *this Q* **have** $partition\text{-}on\ ((B - B' \cup C) \cup B')\ Q'$
unfolding Q' **using** B' ‹$B \cap C = \{\}$› **by** (*auto intro*: *partition-on-union*)
from *this* **have** $partition\text{-}on\ (B \cup C)\ Q'$
using B' **by** (*metis Diff-partition sup.assoc sup.commute*)

from *this* **show** $Q' \in \{P.\ \textit{partition-on}\ (B \cup C)\ P\}$ **by** *auto*
next
fix Q'
assume Q': $Q' \in \{Q'.\ \textit{partition-on}\ (B \cup C)\ Q'\}$
from Q' **have** $\{\} \notin Q'$ **by** (*auto elim*!: *partition-onE*)
obtain Q **where** Q: $Q = ((\lambda X.\ \textit{if}\ X \subseteq B\ \textit{then}\ X\ \textit{else}\ \{\})\ `\ Q') - \{\{\}\}$ **by** *blast*
obtain P' **where** P': $P' = ((\lambda X.\ \textit{if}\ X \subseteq B\ \textit{then}\ \{\}\ \textit{else}\ X)\ `\ Q') - \{\{\}\}$ **by** *blast*
from P' Q $\langle\{\} \notin Q'\rangle$ **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 $\langle B \cap C = \{\}\rangle$ **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 $\langle x \in B - B' \cup C\rangle$ **have** $\neg\ X \subseteq B$
using $\langle B \cap C = \{\}\rangle$ **by** *auto*
from *this* $\langle X \in Q'\rangle$ **have** $X \in P'$ **using** P' **by** *auto*
from *this* $\langle x \in X\rangle$ **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: $P = (\lambda X.\ X \cap C)\ `\ P'$ **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: $f = (\lambda x.\ \textit{if}\ x \in B - B'\ \textit{then}\ (\textit{THE}\ X.\ x \in X \wedge X \in P') \cap C\ \textit{else undefined})$ **by** *blast*
have P'-*prop*: $P' = (\lambda X.\ X \cup \{x \in B - B'.\ f\ x = X\})\ `\ P$
proof
{

fix X
assume $X \in P'$
have *X-subset*: $X \subseteq (B - B') \cup C$
  using *partition-on-P'* ‹$X \in P'$› **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** *?thesis* **by** *simp*
    next
      assume $x \in B - B'$
       from *partition-on-P'* ‹$x \in X$› ‹$X \in P'$› **have** $(THE\ X.\ x \in X \wedge X \in P') = X$
         by (*simp add*: *partition-on-the-part-eq*)
      from ‹$x \in B - B'$› *this* **show** *?thesis* **unfolding** f **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** *?thesis* **by** *simp*
    next
      assume *x-in*: $x \in \{x \in B - B'.\ f\, x = X \cap C\}$
      from *this* **have** *ex1*: $\exists!X.\ x \in X \wedge X \in P'$
        using *partition-on-P'* **by** (*auto intro*!: *partition-on-partition-on-unique*)
      from *x-in X-subset* **have** *eq*: $(THE\ X.\ x \in X \wedge X \in P') \cap C = X \cap C$
        unfolding f **by** *auto*
      from *P'-nosubset* ‹$X \in P'$› **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*: $y \in (THE\ X.\ x \in X \wedge X \in P') \cap C$ **by** *simp*
      from *y y-in* **have** $y \in X$ $y \in (THE\ X.\ x \in X \wedge X \in P')$ **by** *auto*
      moreover from y **have** $\exists!X.\ y \in X \wedge X \in P'$
        using *partition-on-P'* **by** (*simp add*: *partition-on-partition-on-unique*)
      moreover have $(THE\ X.\ x \in X \wedge X \in P') \in P'$
        using *ex1* **by** (*rule the1I2*) *auto*
      ultimately have $(THE\ X.\ x \in X \wedge X \in P') = X$ **using** ‹$X \in P'$› **by**

auto
          from *this ex1* **show** *?thesis* **by** (*auto intro*: *the1I2*)
        qed
      qed
    qed
    from ‹$X \in P'$› *this* **have** $X \in (\lambda X.\ X \cup \{x \in B - B'.\ f\ x = X\})$ ‘ P
      unfolding P **by** *simp*
  }
  from *this* **show** $P' \subseteq (\lambda X.\ X \cup \{x \in B - B'.\ f\ x = X\})$ ‘ P **..**
next
  {
    fix x
    assume *x-in-image*: $x \in (\lambda X.\ X \cup \{x \in B - B'.\ f\ x = X\})$ ‘ P
    {
      fix X
      assume $X \in P'$
      have $\{x \in B - B'.\ f\ x = X \cap C\} = \{x \in B - B'.\ x \in X\}$
      proof −
        {
          fix x
          assume $x \in B - B'$
          from *this* **have** *ex1*: $\exists!X.\ x \in X \wedge X \in P'$
           using *partition-on-P′* **by** (*auto intro*!: *partition-on-partition-on-unique*)
          from *this* **have** *in-p*: (*THE* $X.\ x \in X \wedge X \in P'$) $\in P'$
            and *x-in*: $x \in$ (*THE* $X.\ x \in X \wedge X \in P'$)
            by (*metis* (*mono-tags*, *lifting*) *theI*)+
          have $f\ x = X \cap C \longleftrightarrow$ (*THE* $X.\ x \in X \wedge X \in P'$) $\cap C = X \cap C$
            using ‹$x \in B - B'$› **unfolding** f **by** *auto*
          also have ... $\longleftrightarrow$ (*THE* $X.\ x \in X \wedge X \in P'$) $= X$
          proof
            assume (*THE* $X.\ x \in X \wedge X \in P'$) $= X$
            from *this* **show** (*THE* $X.\ x \in X \wedge X \in P'$) $\cap C = X \cap C$ **by** *auto*
          next
            assume (*THE* $X.\ x \in X \wedge X \in P'$) $\cap C = X \cap C$
            have (*THE* $X.\ x \in X \wedge X \in P'$) $\cap X \neq \{\}$
              using *P′-witness* ‹(*THE* $X.\ x \in X \wedge X \in P'$) $\cap C = X \cap C$› ‹$X \in$
P'› **by** *fastforce*
            from *this* **show** (*THE* $X.\ x \in X \wedge X \in P'$) $= X$
              using *partition-on-P′*[*unfolded partition-on-def disjoint-def*] *in-p* ‹X
$\in P'$› **by** *metis*
          qed
          also have ... $\longleftrightarrow x \in X$
           using *ex1* ‹$X \in P'$› *x-in* **by** (*auto*; *metis* (*no-types*, *lifting*) *the-equality*)
          finally have $f\ x = X \cap C \longleftrightarrow x \in X$ **.**
        }
        from *this* **show** *?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 $X \cap C \cup \{x \in B.\ x \notin B' \wedge f\ x = X \cap C\} = X$ **by** *auto*
}
from *this x-in-image* **have** $x \in P'$ **unfolding** P **by** *auto*
}
from *this* **show** $(\lambda X.\ X \cup \{x \in B - B'.\ f\ x = X\})$ ‘ $P \subseteq P'$ **..**
qed
from *partition-on-P'* **have** *f-prop*: $f \in (B - B') \rightarrow_E P$
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: $k = card\ B'$ **by** *blast*
from ‹*finite B*› ‹$B' \subseteq B$› k **have** *k-prop*: $k \in \{0..card\ B\}$ **by** (*simp add*: *card-mono*)
obtain j **where** j: $j = card\ P$ **by** *blast*
from j ‹*partition-on C P*› **have** *j-prop*: $j \in \{0..card\ C\}$
by (*simp add*: *assms*(*2*) *partition-on-le-set-elements*)
from ‹*partition-on C P*› j **have** *P-prop*: *partition-on C P* $\wedge$ *card P* $= j$ **by** *auto*
from k ‹$B' \subseteq B$› **have** *B'-prop*: $B' \subseteq B \wedge card\ B' = k$ **by** *auto*
show $Q' \in$ *construct-partition-on B C*
using *j-prop k-prop P-prop B'-prop Q-prop P'-prop f-prop Q'-prop*
unfolding *construct-partition-on-def*
by (*auto simp del*: *atLeastAtMost-iff*) *blast*
qed

1.4 Injectivity of the Set Construction

lemma *injectivity*:
assumes $B \cap C = \{\}$
assumes P: (*partition-on C P* $\wedge$ *card P* $= j$) $\wedge$ (*partition-on C P'* $\wedge$ *card P'* $= j'$)
assumes B': $(B' \subseteq B \wedge card\ B' = k) \wedge (B'' \subseteq B \wedge card\ B'' = k')$
assumes Q: *partition-on* $B'\ Q$ $\wedge$ *partition-on* $B''\ Q'$
assumes f: $f \in B - B' \rightarrow_E P \wedge g \in B - B'' \rightarrow_E P'$
assumes P': $P'' = (\lambda X.\ X \cup \{x \in B - B'.\ f\ x = X\})$ ‘ P $\wedge$
$P''' = (\lambda X.\ X \cup \{x \in B - B''.\ g\ x = X\})$ ‘ 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-nonempty-sets*: $\forall X \in P.\ \exists c \in C.\ c \in X$ $\forall X \in P'.\ \exists c \in C.\ c \in X$
using P **by** (*force elim*: *partition-onE*)+
have *1*: $\forall X \in P''.\ \exists c \in C.\ c \in X$ $\forall X \in P'''.\ \exists c \in C.\ c \in X$
using P' *P-nonempty-sets* **by** *fastforce*+
have *2*: $\forall X \in Q.\ \forall x \in X.\ x \notin C$ $\forall X \in Q'.\ \forall x \in X.\ x \notin C$
using ‹$B \cap C = \{\}$› $Q\ B'$ **by** (*auto elim*: *partition-onE*)
from *eq-result* **have** $P'' = P'''$ **and** $Q = Q'$
by (*auto dest*: *injectivity-union*[*OF - 1 2*])
from *this* Q **show** $Q = Q'$ **and** $B' = B''$
by (*auto intro*!: *partition-on-eq-implies-eq-carrier*)
have *subset-C*: $\forall X \in P.\ X \subseteq C$ $\forall X \in P'.\ X \subseteq C$

using P **by** (*auto elim*: *partition-onE*)
have *eq-image*: $(\lambda X.\ X \cup \{x \in B - B'.\ f\ x = X\})$ ‘ $P = (\lambda X.\ X \cup \{x \in B - B''.\ g\ x = X\})$ ‘ P'
using P' ‹$P'' = P'''$› **by** *auto*
from *this* ‹$B \cap C = \{\}$› **show** $P = P'$
by (*auto dest*: *injectivity-image-union*[*OF* - *subset-C*])
have *eq2*: $(\lambda X.\ X \cup \{x \in B - B'.\ f\ x = X\})$ ‘ $P = (\lambda X.\ X \cup \{x \in B - B'.\ g\ x = X\})$ ‘ P
using ‹$P = P'$› ‹$B' = B''$› *eq-image* **by** *simp*
from P **have** *P-props*: $\forall X \in P.\ X \subseteq C$ $\forall X \in P.\ X \neq \{\}$ **by** (*auto elim*: *partition-onE*)
have *invert*: $\forall X{\in}P.\ (X \cup \{x \in B - B'.\ f\ x = X\}) \cap C = X \wedge (X \cup \{x \in B - B'.\ g\ x = X\}) \cap C = X$
using ‹$B \cap C = \{\}$› *P-props* **by** *auto*
have *eq3*: $\forall X \in P.\ (X \cup \{x \in B - B'.\ f\ x = X\}) = (X \cup \{x \in B - B'.\ g\ x = X\})$
using *injectivity-image*[*OF eq2 invert*] **by** *blast*
have *eq4*: $\forall X \in P.\ \{x \in B - B'.\ f\ x = X\} = \{x \in B - B'.\ g\ x = X\}$
proof
fix X
assume $X \in P$
from *this* P **have** $X \subseteq C$ **by** (*auto elim*: *partition-onE*)
have *disjoint*: $X \cap \{x \in B - B'.\ f\ x = X\} = \{\}$ $X \cap \{x \in B - B'.\ g\ x = X\} = \{\}$
using ‹$B \cap C = \{\}$› ‹$X \subseteq C$› **by** *auto*
from *eq3* ‹$X \in P$› **have** $X \cup \{x \in B - B'.\ f\ x = X\} = X \cup \{x \in B - B'.\ g\ x = X\}$ **by** *auto*
from *this disjoint* **show** $\{x \in B - B'.\ f\ x = X\} = \{x \in B - B'.\ g\ x = X\}$
by (*auto intro*: *injectivity-union*)
qed
from *eq4* f **have** *eq5*: $\forall b{\in}B - B'.\ f\ b = g\ b$ **by** *blast*
from *eq5* f ‹$B' = B''$› ‹$P = P'$› **show** *eq6*: $f = g$ **by** (*auto intro*: *extensional-funcset-ext*)
from P ‹$P = P'$› **show** $j = j'$ **by** *simp*
from B' ‹$B' = B''$› **show** $k = k'$ **by** *simp*
qed

1.5 The Generalized Bell Recurrence Relation

theorem *Bell-eq*:
Bell $(n + m) = (\sum k{\le}n.\ \sum j{\le}m.\ j\ \hat{}\ (n - k) * Stirling\ m\ j * (n\ choose\ k) * Bell\ k)$
proof −
define A **where** $A = \{0..<n + m\}$
define B **where** $B = \{0..<n\}$
define C **where** $C = \{n..<n + m\}$
have $A = B \cup C$ $B \cap C = \{\}$ *finite* B *card* $B = n$ *finite* C *card* $C = m$
unfolding *A-def B-def C-def* **by** *auto*
have *step1*: *Bell* $(n + m) = card\ \{P.\ partition\text{-}on\ A\ P\}$

```
  unfolding Bell-def A-def ..
from ‹A = B ∪ C› ‹B ∩ C = {}› ‹finite B› ‹finite C›
have step2: card {P. partition-on A P} = card (construct-partition-on B C)
  by (simp add: construct-partition-on)
note injectivity = injectivity[OF ‹B ∩ C = {}›]
let ?expr = 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}
}
let ?S ≫= ?comp = ?expr
{
  fix k
  assume k: k ∈ {..card B}
  let ?expr = ?comp k
  let ?S ≫= ?comp = ?expr
  {
    fix j
    assume j ∈ {.. card C}
    let ?expr = ?comp j
    let ?S ≫= ?comp = ?expr
    from ‹finite C› have finite ?S
      by (intro finite-Collect-conjI disjI1 finitely-many-partition-on)
    {
      fix P
      assume P: P ∈ {P. partition-on C P ∧ card P = j}
      from this have partition-on C P by simp
      let ?expr = ?comp P
      let ?S ≫= ?comp = ?expr
      have finite P
       using P ‹finite C› by (auto intro: finite-elements)
      from ‹finite B› have finite ?S by (auto simp add: finite-subset)
      moreover
      {
        fix B'
        assume B': B' ∈ {B'. B' ⊆ B ∧ card B' = k}
        from this have B' ⊆ B by simp
        let ?expr = ?comp B'
        let ?S ≫= ?comp = ?expr
        from ‹finite B› have finite B'
          using B' by (auto simp add: finite-subset)
        from ‹finite B'› have finite {Q. partition-on B' Q}
          by (rule finitely-many-partition-on)
        moreover
```

```
{
  fix Q
  assume Q: Q ∈ {Q. partition-on B' Q}
  let ?expr = ?comp Q
  let ?S ≫= ?comp = ?expr
  {
    fix f
    assume f ∈ B − B' →E P
    let ?expr = ?comp f
    let ?S ≫= ?comp = ?expr
    have disjoint-family-on ?comp ?S
      by (auto intro: disjoint-family-onI)
    from this have card ?expr = 1
      by (simp add: card-bind-constant)
    moreover have finite ?expr
      by (simp add: finite-bind)
    ultimately have finite ?expr ∧ card ?expr = 1 by blast
  }
  moreover have finite ?S
    using ‹finite B› ‹finite P› by (auto intro: finite-PiE)
  moreover have disjoint-family-on ?comp ?S
    using P B' Q
    by (injectivity-solver rule: local.injectivity(1))
  moreover have card ?S = j ^ (n − k)
  proof −
    have card (B − B') = n − k
      using B' ‹finite B'› ‹card B = n›
      by (subst card-Diff-subset) auto
    from this show ?thesis
      using ‹finite B› P
      by (subst card-PiE) (simp add: prod-constant)+
  qed
  ultimately have card ?expr = j ^ (n − k)
    by (simp add: card-bind-constant)
  moreover have finite ?expr
    using ‹finite ?S› ‹finite {P. partition-on C P ∧ card P = j}›
    by (auto intro!: finite-bind)
  ultimately have finite ?expr ∧ card ?expr = j ^ (n − k) by blast
} note inner = this
moreover have card ?S = Bell k
  using B' ‹finite B'› by (auto simp add: Bell-altdef[symmetric])
moreover have disjoint-family-on ?comp ?S
  using P B'
  by (injectivity-solver rule: local.injectivity(2))
ultimately have card ?expr = j ^ (n − k) ∗ 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 ^ (n − k) ∗ Bell k by blast
```

} **note** *inner* = *this*
moreover have *card ?S* = *n choose k*
using ‹*card B* = *n*› ‹*finite B*› **by** (*simp add*: *n-subsets*)
moreover have *disjoint-family-on ?comp ?S*
using *P*
by (*injectivity-solver rule*: *local.injectivity*(*3*))
ultimately have *card ?expr* = *j* ^ (*n* − *k*) ∗ (*n choose k*) ∗ *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* ^ (*n* − *k*) ∗ (*n choose k*) ∗ *Bell k* **by** *blast*
} **note** *inner* = *this*
moreover note ‹*finite ?S*›
moreover have *card ?S* = *Stirling m j*
using ‹*finite C*› ‹*card C* = *m*› **by** (*simp add*: *card-partition-on*)
moreover have *disjoint-family-on ?comp ?S*
by (*injectivity-solver rule*: *local.injectivity*(*4*))
ultimately have *card ?expr* = *j* ^ (*n* − *k*) ∗ *Stirling m j* ∗ (*n choose k*) ∗ *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* ^ (*n* − *k*) ∗ *Stirling m j* ∗ (*n choose k*) ∗ *Bell k* **by** *blast*
} **note** *inner* = *this*
moreover have *finite ?S* **by** *simp*
moreover have *disjoint-family-on ?comp ?S*
by (*injectivity-solver rule*: *local.injectivity*(*5*))
ultimately have *card ?expr* = (∑ *j*≤*m*. *j* ^ (*n* − *k*) ∗ *Stirling m j* ∗ (*n choose k*) ∗ *Bell k*) (**is** - = *?formula*)
using ‹*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*
}
moreover have *finite ?S* **by** *simp*
moreover have *disjoint-family-on ?comp ?S*
by (*injectivity-solver rule*: *local.injectivity*(*6*))
ultimately have *step3*: *card* (*construct-partition-on B C*) = (∑ *k*≤*n*. ∑ *j*≤*m*. *j* ^ (*n* − *k*) ∗ *Stirling m j* ∗ (*n choose k*) ∗ *Bell k*)
unfolding *construct-partition-on-def*
using ‹*card B* = *n*› **by** (*subst card-bind*) (*auto intro*: *sum.cong*)
from *step1 step2 step3* **show** *?thesis* **by** *auto*
qed

1.6 Corollaries of the Generalized Bell Recurrence

corollary *Bell-Stirling-eq*:

```
  Bell m = (∑j≤m. Stirling m j)
proof –
  have Bell m = Bell (0 + m) by simp
  also have ... = (∑j≤m. Stirling m j)
    unfolding Bell-eq[of 0] by (simp add: Bell-0)
  finally show ?thesis .
qed

corollary Bell-recursive-eq:
  Bell (n + 1) = (∑k≤n. (n choose k) ∗ Bell k)
unfolding Bell-eq[of - 1] by simp
```

1.7 Code equations for the computation of Bell numbers

It is slow to compute Bell numbers without dynamic programming (DP). The following is a DP algorithm derived from the previous recursion formula *Bell-recursive-eq*.

```
fun Bell-list-aux :: nat ⇒ nat list
  where
  Bell-list-aux 0 = [1] |
  Bell-list-aux (Suc n) = (
    let prev-list = Bell-list-aux n;
        next-val = (∑(k,z) ← List.enumerate 0 prev-list. z ∗ (n choose (n−k)))
    in next-val#prev-list)

definition Bell-list :: nat ⇒ nat list
  where Bell-list n = rev (Bell-list-aux n)

lemma bell-list-eq: Bell-list n = map Bell [0..<n+1]
proof –
  have Bell-list-aux n = rev (map Bell [0..<Suc n])
  proof (induction n)
    case 0
    then show ?case by (simp add:Bell-0)
  next
    case (Suc n)
    define x where x = Bell-list-aux n
    define y where y = (∑(k,z) ← List.enumerate 0 x. z ∗ (n choose (n−k)))
    define sn where sn = n+1
    have b:x = rev (map Bell [0..<sn])
      using Suc x-def sn-def by simp
    have c: length x = sn
      unfolding b by simp

    have snd i = Bell (n − fst i) if i ∈ set (List.enumerate 0 x) for i
    proof –
      have fst i < length x snd i = x ! fst i
        using iffD1[OF in-set-enumerate-eq that] by auto
      hence snd i = Bell (sn − Suc (fst i))
```

unfolding *b* **by** (*simp add*:*rev-nth*)
thus *?thesis*
unfolding *sn-def* **by** *simp*
qed

hence $y = (\sum i{\leftarrow}enumerate\ 0\ x.\ Bell\ (n - fst\ i) * (n\ choose\ (n - fst\ i)))$
unfolding *y-def* **by** (*intro arg-cong*[**where** *f*=*sum-list*] *map-cong refl*)
(*simp add*:*case-prod-beta*)
also have ... $= (\sum i{\leftarrow}map\ fst\ (enumerate\ 0\ x).\ Bell\ (n - i) * (n\ choose\ (n - i)))$
by (*subst map-map*) (*simp add*:*comp-def*)
also have ... $= (\sum i = 0..{<}length\ x.\ Bell\ (n{-}i) * (n\ choose\ (n{-}i)))$
by (*simp add*:*interv-sum-list-conv-sum-set-nat*)
also have ... $= (\sum i{\le}n.\ Bell\ (n{-}i) * (n\ choose\ (n{-}i)))$
using *c sn-def* **by** (*intro sum.cong*) *auto*
also have ... $= (\sum i \in (\lambda k.\ n{-}\ k)\ `\ \{..n\}.\ Bell\ i * (n\ choose\ i))$
by (*subst sum.reindex*, *auto simp add*:*inj-on-def*)
also have ... $= (\sum i \le n.\ Bell\ i * (n\ choose\ i))$
by (*intro sum.cong refl iffD2*[*OF set-eq-iff*] *allI*)
(*simp add*:*image-iff atMost-def*, *presburger*)
also have ... = *Bell* (*Suc n*)
using *Bell-recursive-eq* **by** (*simp add*:*mult.commute*)
finally have *a*: *y* = *Bell* (*Suc n*) **by** *simp*

have *Bell-list-aux* (*Suc n*) = *y*#*x*
unfolding *x-def y-def* **by** (*simp add*:*Let-def*)
also have ... = *Bell* (*Suc n*)#(*rev* (*map Bell* [*0*..<*Suc n*]))
unfolding *a b sn-def* **by** *simp*
also have ... = *rev* (*map Bell* [*0*..<*Suc* (*Suc n*)])
by *simp*
finally show *?case* **by** *simp*
qed
thus *Bell-list n* = *map Bell* [*0*..<*n*+*1*]
by (*simp add*:*Bell-list-def*)
qed

lemma *Bell-eval*[*code*]: *Bell n* = *last* (*Bell-list n*)
unfolding *bell-list-eq* **by** *simp*

end

References

[1] N. J. A. Sloane. A000110: Bell or exponential numbers: number of ways to partition a set of n labeled elements. In *The On-Line Encyclopedia of Integer Sequences*. https://oeis.org/A000110.

[2] M. Z. Spivey. A generalized recurrence for Bell numbers. *Journal of*

Integer Sequences, 11, 2008. Electronic copy available at https://cs.uwaterloo.ca/journals/JIS/VOL11/Spivey/spivey25.pdf.