

# Spivey's Generalized Recurrence for Bell Numbers

Lukas Bulwahn

May 26, 2024

## 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

<b>1</b>	<b>Bell Numbers and Spivey's Generalized Recurrence</b>	<b>1</b>
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**  $\bigwedge x. x \in A \implies f\ x = g\ x$

**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**  $\bigwedge x. x \in X \implies x \subseteq A$

**shows** *inj-on*  $((\cdot)\ f)\ X$

**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**  $Q\ B\ Q\ D$

$\bigwedge S\ T. P\ S \implies Q\ T \implies S \cap T = \{\}$

**shows**  $A = C \wedge B = D$

**using** *assms* **by** (*Int-Un-distrib Int-commute inf-sup-absorb* **by** *blast+*)

**lemma** *injectivity-image*:

**assumes**  $f\ 'A = g\ 'A$

**assumes**  $\forall x \in A. \text{invert}\ (f\ x) = x \wedge \text{invert}\ (g\ x) = x$

**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)*  $\langle X \in P \rangle$  have  $X: (X \cup F X) \cap A = X$  by *auto*  
 moreover from *assms(3,5)*  $\langle X' \in P' \rangle$  have  $X': (X' \cup G X') \cap A = X'$  by  
*auto*  
 ultimately have  $X = X'$  by *simp*  
 from *this*  $\langle X' \in P' \rangle$  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)*  $\langle X \in P \rangle$  have  $X: (X \cup F X) \cap A = X$  by *auto*  
 moreover from *assms(3,5)*  $\langle X' \in P' \rangle$  have  $X': (X' \cup G X') \cap A = X'$  by  
*auto*  
 ultimately have  $X = X'$  by *simp*  
 from *this*  $\langle X \in P \rangle$  show  $X' \in P$  by *auto*  
 qed  
 qed

## 1.2 Definition of Bell Numbers

**definition** *Bell* :: *nat*  $\Rightarrow$  *nat*

where

*Bell*  $n = \text{card } \{P. \text{partition-on } \{0..<n\} P\}$

**lemma** *Bell-altdef*:

assumes *finite*  $A$

shows *Bell* (*card*  $A$ ) = *card*  $\{P. \text{partition-on } A P\}$

**proof** –

from  $\langle \text{finite } A \rangle$  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*:  $A = f \text{ ` } \{0..<\text{card } A\}$

using *bij-betw-imp-surj-on* by *blast*

have  $\forall x \in \{P. \text{partition-on } \{0..<\text{card } A\} P\}. x \subseteq \text{Pow } \{0..<\text{card } A\}$

by (*auto elim: partition-onE*)

from *this inj* have *inj-on*  $((\cdot) ((\cdot) f)) \{P. \text{partition-on } \{0..<\text{card } A\} P\}$

by (*intro inj-on-impl-inj-on-image[of - Pow {0..<card A}]*)

*inj-on-impl-inj-on-image[of - {0..<card A}]*) *blast+*

moreover from *inj* have  $(\cdot) ((\cdot) f) \text{ ` } \{P. \text{partition-on } \{0..<\text{card } A\} P\} = \{P. \text{partition-on } A P\}$

by (*subst image-f-eq, auto elim!: set-of-partition-on-map*)

ultimately have *bij-betw*  $((\cdot) ((\cdot) f)) \{P. \text{partition-on } \{0..<\text{card } A\} P\} \{P. \text{partition-on } A P\}$

by (*auto intro: bij-betw-imageI*)

**from** *this*  $\langle$ *finite A* $\rangle$  **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*  $\Rightarrow$  '*a set*  $\Rightarrow$  '*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-on C P  $\wedge$  card P = j};
    B'  $\leftarrow$  {B'. B'  $\subseteq$  B  $\wedge$  card B' = k};
    Q  $\leftarrow$  {Q. partition-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-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-on C P*  $\wedge$  *card P = j**  
**and** *B': B'  $\subseteq$  B  $\wedge$  *card B' = k**  
**and** *Q: *partition-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-on (B - B'  $\cup$  C) P'*  
**unfolding** *P'* **using**  $\langle$ *B  $\cap$  C = {}* $\rangle$   
**by** (*intro partition-on-insert-elements*) *auto*  
**from** *this Q* **have** *partition-on ((B - B'  $\cup$  C)  $\cup$  B') Q'*  
**unfolding** *Q'* **using** *B'*  $\langle$ *B  $\cap$  C = {}* $\rangle$  **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  $Q: Q = ((\lambda X. \text{if } X \subseteq B \text{ then } X \text{ else } \{\}) ' Q') - \{\{\}\}$  by blast
  obtain  $P'$  where  $P': P' = ((\lambda X. \text{if } X \subseteq B \text{ then } \{\} \text{ else } X) ' Q') - \{\{\}\}$  by
blast
  from  $P' Q \{\} \notin Q'$  have  $Q'\text{-prop}: Q' = P' \cup Q$  by auto
  have  $P'\text{-nosubset}: \forall X \in P'. \neg X \subseteq B$ 
    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  $P'\text{-witness}: \forall X \in P'. \exists x. x \in X \cap C$ 
    using  $\langle B \cap C = \{\} \rangle$  by fastforce
  obtain  $B'$  where  $B': B' = \bigcup Q$  by blast
  have  $Q\text{-prop}: \text{partition-on } B' Q$ 
    using  $B' Q' Q'\text{-prop partition-on-split2 mem-Collect-eq}$  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  $X: x \in X \wedge X \in Q'$ 
    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
  from this have  $\text{partition-on-}P': \text{partition-on } (B - B' \cup C) P'$ 
    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  $P$   $\text{partition-on-}P' P'\text{-witness}$  have  $\text{partition-on } C P$ 
    using partition-on-intersect-on-elements by auto
  obtain  $f$  where  $f: f = (\lambda x. \text{if } x \in B - B' \text{ then } (\text{THE } X. x \in X \wedge X \in P') \cap C \text{ else undefined})$  by blast
  have  $P'\text{-prop}: P' = (\lambda X. X \cup \{x \in B - B'. f x = X\}) ' P$ 
  proof
    {

```

```

fix X
assume X ∈ P'
have X-subset: X ⊆ (B - B') ∪ C
  using partition-on-P' ⟨X ∈ P'⟩ by (auto elim: partition-onE)
have X = X ∩ C ∪ {x ∈ B - B'. f x = X ∩ C}
proof
  {
    fix x
    assume x ∈ X
    from this X-subset have x ∈ (B - B') ∪ C by auto
    from this have x ∈ X ∩ C ∪ {x ∈ B - B'. f x = X ∩ C}
    proof
      assume x ∈ C
      from this ⟨x ∈ X⟩ show ?thesis by simp
    next
      assume x ∈ B - B'
      from partition-on-P' ⟨x ∈ X⟩ ⟨X ∈ P'⟩ have (THE X. x ∈ X ∧ X ∈
P') = X
        by (simp add: partition-on-the-part-eq)
      from ⟨x ∈ B - B'⟩ this show ?thesis unfolding f by auto
    qed
  }
from this show X ⊆ X ∩ C ∪ {x ∈ B - B'. f x = X ∩ C} by auto
next
show X ∩ C ∪ {x ∈ B - B'. f x = X ∩ C} ⊆ X
proof
  fix x
  assume x ∈ X ∩ C ∪ {x ∈ B - B'. f x = X ∩ C}
  from this show x ∈ X
  proof
    assume x ∈ X ∩ C
    from this show ?thesis by simp
  next
    assume x-in: x ∈ {x ∈ B - B'. f x = X ∩ C}
    from this have ex1: ∃!X. x ∈ X ∧ X ∈ P'
      using partition-on-P' by (auto intro!: partition-on-partition-on-unique)
    from x-in X-subset have eq: (THE X. x ∈ X ∧ X ∈ P') ∩ C = X ∩ C
      unfolding f by auto
    from P'-nosubset ⟨X ∈ P'⟩ have ¬ X ⊆ B by simp
    from this have X ∩ C ≠ {}
      using X-subset assms(3) by blast
    from this obtain y where y: y ∈ X ∩ C by auto
    from this eq have y-in: y ∈ (THE X. x ∈ X ∧ X ∈ P') ∩ C by simp
    from y y-in have y ∈ X y ∈ (THE X. x ∈ X ∧ X ∈ P') by auto
    moreover from y have ∃!X. y ∈ X ∧ X ∈ P'
      using partition-on-P' by (simp add: partition-on-partition-on-unique)
    moreover have (THE X. x ∈ X ∧ X ∈ P') ∈ P'
      using ex1 by (rule the1I2) auto
    ultimately have (THE X. x ∈ X ∧ X ∈ P') = X using ⟨X ∈ P'⟩ by

```

```

auto
  from this ex1 show ?thesis by (auto intro: the1I2)
  qed
  qed
  qed
  from  $\langle X \in P' \rangle$  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  $f x = X \cap C \iff (THE X. x \in X \wedge X \in P') \cap C = X \cap C$ 
        using  $\langle x \in B - B' \rangle$  unfolding f by auto
        also have  $\dots \iff (THE X. x \in X \wedge X \in P') = X$ 
        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  $\langle (THE X. x \in X \wedge X \in P') \cap C = X \cap C \rangle \langle X \in P' \rangle$  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  $\langle X \in P' \rangle$  by metis
        qed
        also have  $\dots \iff x \in X$ 
        using ex1  $\langle X \in P' \rangle$  x-in by (auto; metis (no-types, lifting) the-equality)
        finally have  $f x = X \cap C \iff x \in X .$ 
      }
    }
  }
  from this show ?thesis by auto
  qed
  moreover have  $X \subseteq B - B' \cup C$ 
  using partition-on-P'  $\langle X \in P' \rangle$  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**  $(\lambda X. X \cup \{x \in B - B'. f x = X\}) \text{ ' } P \subseteq P' \text{ ..}$   
**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**  $k = \text{card } B'$  **by** *blast*  
**from**  $\langle \text{finite } B \rangle \langle B' \subseteq B \rangle k$  **have** *k-prop*:  $k \in \{0.. \text{card } B\}$  **by** (*simp add: card-mono*)  
**obtain j where**  $j = \text{card } P$  **by** *blast*  
**from j**  $\langle \text{partition-on } C P \rangle$  **have** *j-prop*:  $j \in \{0.. \text{card } C\}$   
**by** (*simp add: assms(2) partition-on-le-set-elements*)  
**from**  $\langle \text{partition-on } C P \rangle j$  **have** *P-prop*:  $\text{partition-on } C P \wedge \text{card } P = j$  **by** *auto*  
**from**  $k \langle B' \subseteq B \rangle$  **have** *B'-prop*:  $B' \subseteq B \wedge \text{card } B' = k$  **by** *auto*  
**show**  $Q' \in \text{construct-partition-on } B C$   
**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*  $\wedge \text{card } P = j$ )  $\wedge$  (*partition-on C P'*  $\wedge \text{card } P' = j'$ )  
**assumes**  $B'$ : ( $B' \subseteq B \wedge \text{card } B' = k$ )  $\wedge$  ( $B'' \subseteq B \wedge \text{card } B'' = k'$ )  
**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**  $P'$ :  $P'' = (\lambda X. X \cup \{x \in B - B'. f x = X\}) \text{ ' } P \wedge$   
 $P''' = (\lambda X. X \cup \{x \in B - B''. g x = X\}) \text{ ' } P'$   
**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**  $\langle B \cap C = \{\} \rangle Q B'$  **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*:  $(\lambda X. X \cup \{x \in B - B'. f x = X\}) \text{ ' } P = (\lambda X. X \cup \{x \in B - B''. g x = X\}) \text{ ' } P'$   
**using**  $P' \text{ ' } P'' = P'''$  **by** *auto*  
**from** *this*  $\langle B \cap C = \{\} \rangle$  **show**  $P = P'$   
**by** (*auto dest: injectivity-image-union[OF - subset-C]*)  
**have** *eq2*:  $(\lambda X. X \cup \{x \in B - B'. f x = X\}) \text{ ' } P = (\lambda X. X \cup \{x \in B - B'. g x = X\}) \text{ ' } P$   
**using**  $\langle P = P' \rangle \langle B' = B'' \rangle$  *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**  $\langle B \cap C = \{\} \rangle$  *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**  $\langle B \cap C = \{\} \rangle \langle X \subseteq C \rangle$  **by** *auto*  
**from** *eq3*  $\langle X \in P \rangle$  **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*  $f \text{ ' } \langle B' = B'' \rangle \langle P = P' \rangle$  **show** *eq6*:  $f = g$  **by** (*auto intro: extensional-funcset-ext*)  
**from**  $P \text{ ' } \langle P = P' \rangle$  **show**  $j = j'$  **by** *simp*  
**from**  $B' \text{ ' } \langle B' = B'' \rangle$  **show**  $k = k'$  **by** *simp*  
**qed**

## 1.5 The Generalized Bell Recurrence Relation

**theorem** *Bell-eq*:

*Bell*  $(n + m) = (\sum k \leq n. \sum j \leq m. j \wedge (n - k) * \text{Stirling } m j * (n \text{ choose } k) * \text{Bell } k)$

**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*  $(n + m) = \text{card } \{P. \text{partition-on } A P\}$

```

unfolding Bell-def A-def ..
from  $\langle A = B \cup C \rangle \langle B \cap C = \{\} \rangle \langle \text{finite } B \rangle \langle \text{finite } C \rangle$ 
have step2:  $\text{card } \{P. \text{partition-on } A \ P\} = \text{card } (\text{construct-partition-on } B \ C)$ 
  by (simp add: construct-partition-on)
note injectivity = injectivity[OF  $\langle B \cap C = \{\} \rangle$ ]
let ?expr = do {
  k  $\leftarrow \{0.. \text{card } B\}$ ;
  j  $\leftarrow \{0.. \text{card } C\}$ ;
  P  $\leftarrow \{P. \text{partition-on } C \ P \wedge \text{card } P = j\}$ ;
  B'  $\leftarrow \{B'. B' \subseteq B \wedge \text{card } B' = k\}$ ;
  Q  $\leftarrow \{Q. \text{partition-on } B' \ Q\}$ ;
  f  $\leftarrow (B - B') \rightarrow_E P$ ;
  P'  $\leftarrow \{(\lambda X. X \cup \{x \in B - B'. f \ x = X\}) \text{ ' } P\}$ ;
   $\{P' \cup Q\}$ 
}
let ?S  $\gg=$  ?comp = ?expr
{
  fix k
  assume k:  $k \in \{.. \text{card } B\}$ 
  let ?expr = ?comp k
  let ?S  $\gg=$  ?comp = ?expr
  {
    fix j
    assume j  $\in \{.. \text{card } C\}$ 
    let ?expr = ?comp j
    let ?S  $\gg=$  ?comp = ?expr
    from  $\langle \text{finite } C \rangle$  have finite ?S
    by (intro finite-Collect-conjI disjI1 finitely-many-partition-on)
    {
      fix P
      assume P:  $P \in \{P. \text{partition-on } C \ P \wedge \text{card } P = j\}$ 
      from this have partition-on C P by simp
      let ?expr = ?comp P
      let ?S  $\gg=$  ?comp = ?expr
      have finite P
      using P  $\langle \text{finite } C \rangle$  by (auto intro: finite-elements)
      from  $\langle \text{finite } B \rangle$  have finite ?S by (auto simp add: finite-subset)
      moreover
      {
        fix B'
        assume B':  $B' \in \{B'. B' \subseteq B \wedge \text{card } B' = k\}$ 
        from this have  $B' \subseteq B$  by simp
        let ?expr = ?comp B'
        let ?S  $\gg=$  ?comp = ?expr
        from  $\langle \text{finite } B \rangle$  have finite B'
        using B' by (auto simp add: finite-subset)
        from  $\langle \text{finite } B' \rangle$  have finite  $\{Q. \text{partition-on } B' \ Q\}$ 
        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*:

$Bell\ m = (\sum_{j \leq m}. Stirling\ m\ j)$   
**proof** –  
**have**  $Bell\ m = Bell\ (0 + m)$  **by** *simp*  
**also have**  $\dots = (\sum_{j \leq m}. Stirling\ m\ j)$   
**unfolding**  $Bell\text{-eq}[of\ 0]$  **by** (*simp add: Bell-0*)  
**finally show** *?thesis* .  
**qed**

**corollary** *Bell-recursive-eq*:  
 $Bell\ (n + 1) = (\sum_{k \leq n}. (n\ choose\ k) * Bell\ k)$   
**unfolding**  $Bell\text{-eq}[of\ -1]$  **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*  $\Rightarrow$  *nat list*  
**where**  
 $Bell\text{-list-aux}\ 0 = [1]$  |  
 $Bell\text{-list-aux}\ (Suc\ n) =$   
*let* *prev-list* = *Bell-list-aux* *n*;  
 $next\text{-val} = (\sum (k,z) \leftarrow List.enumerate\ 0\ prev\text{-list}. z * (n\ choose\ (n-k)))$   
*in*  $next\text{-val}\#\text{prev-list}$

**definition** *Bell-list* :: *nat*  $\Rightarrow$  *nat list*  
**where**  $Bell\text{-list}\ n = rev\ (Bell\text{-list-aux}\ n)$

**lemma** *bell-list-eq*:  $Bell\text{-list}\ n = map\ Bell\ [0..<n+1]$

**proof** –  
**have**  $Bell\text{-list-aux}\ n = rev\ (map\ Bell\ [0..<Suc\ n])$   
**proof** (*induction* *n*)  
**case** *0*  
**then show** *?case* **by** (*simp add: Bell-0*)  
**next**  
**case** (*Suc* *n*)  
**define** *x* **where**  $x = Bell\text{-list-aux}\ n$   
**define** *y* **where**  $y = (\sum (k,z) \leftarrow List.enumerate\ 0\ x. z * (n\ choose\ (n-k)))$   
**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**  $i \in set\ (List.enumerate\ 0\ x)$  **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 = (∑ i←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 ... = (∑ i←map fst (enumerate 0 x). Bell (n - i) * (n choose (n -
i)))
  by (subst map-map) (simp add:comp-def)
also have ... = (∑ i = 0..

```

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>.