

# More

By filip

February 24, 2018

## Contents

<b>1</b>	<b>Auxiliary lemmas about lists</b>	<b>1</b>
<b>2</b>	<b>Auxiliary lemmas about (finite) sets</b>	<b>9</b>
<b>3</b>	<b>Auxiliary lemmas about sum of set operator</b>	<b>13</b>
<b>4</b>	<b>Auxiliary lemmas about mappings</b>	<b>14</b>
<b>5</b>	<b>Auxiliary lemmas about natural numbers</b>	<b>15</b>
<b>6</b>	<b>Auxiliary lemmas about bijective functions</b>	<b>15</b>
6.0.1	Extending bijective functions . . . . .	15
6.0.2	Bijections between the domain of a set family and natural numbers. . . . .	18
6.0.3	Bijections between distinct lists' elements and natural numbers . . . . .	19

## 1 Auxiliary lemmas about lists

```
theory MoreList
imports Main
begin

lemma ex-list-of-set:
  assumes finite A
  shows  $\exists l. \text{distinct } l \wedge \text{set } l = A$ 
using assms
proof (induct A)
  case empty
  thus ?case
    by auto
next
  case (insert x A')
  then obtain l where distinct l set l = A'
```

```

    by auto
  hence  $\text{distinct } (x \# l) \text{ set } (x \# l) = \text{insert } x A'$ 
    using insert(2)
    by auto
  thus ?case
    by blast
qed

lemma map-of-sort-key:
  assumes  $\text{distinct } P$ 
  shows  $\text{map-of } (\text{sort-key snd } (\text{zip } P (\text{map } f P))) = \text{map-of } (\text{zip } P (\text{map } f P))$ 
proof (rule ext)
  fix x
  show  $\text{map-of } (\text{sort-key snd } (\text{zip } P (\text{map } f P))) x = \text{map-of } (\text{zip } P (\text{map } f P)) x$ 
  proof (cases  $x \in \text{set } P$ )
    case True
    hence  $(x, f x) \in \text{set } (\text{zip } P (\text{map } f P))$ 
       $(x, f x) \in \text{set } (\text{sort-key snd } (\text{zip } P (\text{map } f P)))$ 
    by (auto simp add: set-zip in-set-conv-nth)
    moreover
    have  $\text{distinct } (\text{map fst } (\text{sort-key snd } (\text{zip } P (\text{map } f P))))$ 
      using  $\langle \text{distinct } P \rangle$ 
    by (auto simp add: distinct-map inj-on-def set-zip intro: distinct-zipI1)
    ultimately
    show ?thesis
      using  $\langle \text{distinct } P \rangle$ 
      using Some-eq-map-of-iff[of  $\text{zip } P (\text{map } f P) f x x$ ]
      using Some-eq-map-of-iff[of  $\text{sort-key snd } (\text{zip } P (\text{map } f P)) f x x$ ]
      by simp
    next
    case False
    hence *:  $\text{map-of } (\text{zip } P (\text{map } f P)) x = \text{None}$ 
       $\text{map-of } (\text{sort-key snd } (\text{zip } P (\text{map } f P))) x = \text{None}$ 
    using map-of-eq-None-iff[of  $\text{zip } P (\text{map } f P) x$ ]
    using map-of-eq-None-iff[of  $\text{sort-key snd } (\text{zip } P (\text{map } f P)) x$ ]
    by auto
    show ?thesis
      by (subst *, subst *) simp
  qed
qed

```

```

lemma sorted-first-min:
   $\llbracket \text{sorted } (x \# xs); x' \in \text{set } xs \rrbracket \implies x \leq x'$ 
by (induct xs arbitrary: x)(auto, force)

```

```

lemma sorted-tl:
   $\text{sorted } (x \# xs) \implies \text{sorted } xs$ 

```

```

by (induct xs) auto

context linorder
begin
inductive sorted-desc :: 'a list  $\Rightarrow$  bool where
  Nil [iff]: sorted-desc []
| Cons:  $\forall y \in \text{set } xs. x \geq y \implies \text{sorted-desc } xs \implies \text{sorted-desc } (x \# xs)$ 
end

lemma sorted-desc-single [iff]:
  sorted-desc [x]
by (rule sorted-desc.Cons) auto

lemma sorted-desc-many:
   $x \geq y \implies \text{sorted-desc } (y \# zs) \implies \text{sorted-desc } (x \# y \# zs)$ 
by (rule sorted-desc.Cons) (cases y # zs rule: sorted-desc.cases, auto)

lemma sorted-desc-many-eq [simp, code]:
   $\text{sorted-desc } (x \# y \# zs) \longleftrightarrow x \geq y \wedge \text{sorted-desc } (y \# zs)$ 
by (auto intro: sorted-desc-many elim: sorted-desc.cases)

lemma sorted-desc-first-max:  $\llbracket \text{sorted-desc } (x \# xs); x' \in \text{set } xs \rrbracket \implies x \geq x'$ 
by (induct xs arbitrary: x)(auto, force)

lemma sorted-desc-tl:
   $\text{sorted-desc } (x \# xs) \implies \text{sorted-desc } xs$ 
by (induct xs) auto

lemma sorted-sorted-desc-rev:  $\text{sorted } xs = \text{sorted-desc } (\text{rev } xs)$ 
by (induct xs rule: rev-induct) (auto simp add: sorted-append elim: sorted-desc.cases)

lemma sorted-negate:  $\text{sorted } xs = \text{sorted-desc } (\text{map } (\lambda x::'a::\text{linordered-ab-group-add. } - x) xs)$ 
proof (induct xs)
  case Nil
  thus ?case
  by simp
next
  case (Cons x xs)
  thus ?case
  using sorted-tl[of x xs] sorted-desc-tl[of -x map uminus xs]
  using sorted-first-min[of x xs] sorted-desc-first-max[of -x map uminus xs]
  by (auto intro!: sorted-desc.Cons sorted.Cons) force
qed

lemma sorted-rev-uminus:
  fixes l::('a  $\times$  'b::linordered-ab-group-add) list
  shows sorted (rev (map ( $\lambda(a, b). - b$ ) (sort-key snd l)))
  apply (subst sorted-sorted-desc-rev)

```

```

using sorted-negate[of map snd (sort-key snd l)]
by (simp add: comp-def split-def)

lemma sorted-takeWhile-snd-neg:
  fixes f :: 'a  $\Rightarrow$  'b::linordered-ab-group-add
  assumes U = sort-key snd (zip P (map f P)) and A  $\in$  set P and f A < 0
  shows (A, f A)  $\in$  set (takeWhile ( $\lambda$  (a, b). b < 0) U)
proof -
  have sorted (rev (map ( $\lambda$ (a, b). - b) U))
    using sorted-rev-uminus  $\langle$ U = sort-key snd (zip P (map f P)) $\rangle$ 
    by simp
  thus ?thesis
    using filter-equals-takeWhile-sorted-rev[of  $\lambda$  (a, b). -b U 0, THEN sym]
    using  $\langle$ A  $\in$  set P $\rangle$   $\langle$ f A < 0 $\rangle$   $\langle$ U = sort-key snd (zip P (map f P)) $\rangle$ 
    unfolding split-def
    by (auto simp add: zip-map2 zip-same-conv-map)
qed

lemma sorted-map:
  assumes sorted l and  $\forall$  x y. x  $\leq$  y  $\longrightarrow$  f x  $\leq$  f y
  shows sorted (map f l)
using assms
by (induct l) (auto simp add: sorted-Cons)

lemma subset-card-length:
  assumes A  $\subseteq$  set l length l = card A
  shows A = set l
using assms
by (metis List.finite-set card-length card-seteq)

lemma length-sorted-list-of-set':
  assumes finite X and sorted l and distinct l and set l = X
  shows length l = card X
using assms
proof (induct X arbitrary: l)
  case empty
  thus ?case
    by simp
next
  case (insert a l')
  have length l  $\neq$  0
    using insert(6)
    by auto
  show ?case
    using insert(1) insert(2) insert(3)[of remove1 a l] insert(4) insert(5) insert(6)
    apply (auto simp add: sorted-remove1 length-remove1)
    using  $\langle$ length l  $\neq$  0 $\rangle$ 

```

by *arith*  
qed

**lemma** *length-sorted-list-of-set*:  
 assumes *finite X*  
 shows  $\text{length } (\text{sorted-list-of-set } X) = \text{card } X$   
 using *assms*  
 using *sorted-list-of-set*[of *X*] *length-sorted-list-of-set*'[of *X sorted-list-of-set X*]  
 by *simp*

**lemma** *sorted-list-of-set-inj*:  
 assumes *finite x* and *finite y*  
 assumes  $\text{sorted-list-of-set } x = \text{sorted-list-of-set } y$   
 shows  $x = y$   
**proof** –  
 have  $x = \text{set } (\text{sorted-list-of-set } x)$   $y = \text{set } (\text{sorted-list-of-set } y)$   
 using *sorted-list-of-set*[of *x*] *sorted-list-of-set*[of *y*]  
 using  $\langle \text{finite } x \rangle$   $\langle \text{finite } y \rangle$   
 by *simp-all*  
 thus ?thesis  
 using  $\langle \text{sorted-list-of-set } x = \text{sorted-list-of-set } y \rangle$   
 by *simp*  
 qed

**lemma** *length-filter-less* [*simp*]:  $\text{length } [l \leftarrow \text{list} . P \ l] < \text{Suc } (\text{length } \text{list})$   
 using *length-filter-le*[of *P list*]  
 by *arith*

**lemma** *exists-zip*:  $\forall (x, y) \in \text{set } L'. y = f \ x \implies L' = \text{zip } (\text{map } \text{fst } L') (\text{map } f (\text{map } \text{fst } L'))$   
**proof** (*rule nth-equalityI*)  
 assume  $\forall (x, y) \in \text{set } L'. y = f \ x$   
 show  $\forall i < \text{length } L'. L' ! i = (\text{zip } (\text{map } \text{fst } L') (\text{map } f (\text{map } \text{fst } L')) ! i$   
**proof** (*safe*)  
 fix *i*  
 assume  $i < \text{length } L'$   
 hence  $\text{snd } (L' ! i) = f (\text{fst } (L' ! i))$   
 using  $\langle \forall (x, y) \in \text{set } L'. y = f \ x \rangle$   
 using *in-set-conv-nth*[of  $L' ! i L'$ ]  
 by *auto*  
 thus  $L' ! i = \text{zip } (\text{map } \text{fst } L') (\text{map } f (\text{map } \text{fst } L')) ! i$   
 using  $\langle i < \text{length } L' \rangle$   
 using *surjective-pairing*[of  $L' ! i$ ]  
 by *simp*  
 qed  
 qed *simp*

```

lemma inj-on-fst:
  assumes  $\forall (x, y) \in \text{set } l. y = f x \text{ distinct } l$ 
  shows inj-on fst (set l)
using assms
unfolding inj-on-def split-def
by auto (subgoal-tac b = f aa, auto)

```

```

lemma distinct-list-distinct-sorted-elems:
  assumes  $i \neq j$  and  $i < \text{length } F$  and  $j < \text{length } F$ 
  distinct F and  $\forall l \in \text{List.set } F. \text{distinct } l \wedge \text{sorted } l$ 
  shows  $\text{List.set } (F ! i) \neq \text{List.set } (F ! j)$ 
using assms
using nth-eq-iff-index-eq[of F i j] (distinct F)
using sorted-distinct-set-unique[of F ! i F ! j]
by auto

```

```

definition drop-nth ::  $\text{nat} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list}$  where
  drop-nth n l = (take n l) @ (drop (n + 1) l)

```

```

lemma distinct-drop-nth:
  assumes distinct S
  shows distinct (drop-nth i S)
  using assms
  unfolding drop-nth-def
by (auto simp add: set-take-disj-set-drop-if-distinct)

```

```

lemma length-drop-nth:
  assumes  $i < \text{length } l$ 
  shows  $\text{length } (\text{drop-nth } i \ l) = \text{length } l - 1$ 
using assms
unfolding drop-nth-def
by auto

```

```

lemma set-drop-nth:
  assumes  $i < \text{length } l$ 
  shows  $\text{set } (\text{drop-nth } i \ l) \cup \{l ! i\} = \text{set } l$ 
proof
  show  $\text{set } (\text{drop-nth } i \ l) \cup \{l ! i\} \subseteq \text{set } l$ 
    using assms
    unfolding drop-nth-def
    by (auto dest: in-set-takeD in-set-dropD)
next
  show  $\text{set } l \subseteq \text{set } (\text{drop-nth } i \ l) \cup \{l ! i\}$ 

```

```

    using assms
    unfolding drop-nth-def
    by (subst id-take-nth-drop[of i l]) auto
qed

```

```

lemma set-drop-nth-distinct:
  assumes i < length l distinct l
  shows set (drop-nth i l) = set l - {l ! i}
proof-
  have distinct (take i l @ l ! i # drop (Suc i) l)
    using  $\langle \text{distinct } l \rangle \langle i < \text{length } l \rangle$ 
    by (subst (asm) id-take-nth-drop[of i l]) auto
  hence  $l ! i \notin \text{set } (take\ i\ l) \wedge l ! i \notin \text{set } (drop\ (Suc\ i)\ l)$ 
    by auto
  thus ?thesis
    using set-drop-nth[of i l]  $\langle i < \text{length } l \rangle$ 
    unfolding drop-nth-def
    by (auto dest: in-set-takeD in-set-dropD)
qed

```

```

definition spread where
  spread w l = concat (map ( $\lambda (w, a).$  replicate w a) (zip w l))

```

```

lemma length-spread:
  assumes length w = length l
  shows length (spread w l) = sum-list w
proof-
  have map (length  $\circ (\lambda(x, y).$  replicate x y)) (zip w l) = w
    using assms
  proof (induct w arbitrary: l)
    case (Cons w0 w')
    thus ?case
      by (cases l) auto
  qed simp
  thus ?thesis
    using assms
    unfolding spread-def
    by (auto simp add: length-concat)
qed

```

```

lemma set-spread:
  set (spread w l)  $\subseteq$  set l
unfolding spread-def
by (auto simp add: set-zip)

```

```

lemma [simp]: spread (w # w') (l # l') = (replicate w l) @ spread w' l'
unfolding spread-def

```

**by** *auto*

**lemma** [*simp*]:

*sum-list* (*replicate* *wi* *li*) = *int* *wi* \* *li*

**by** (*induct* *wi*) (*auto simp add: int-distrib*(1))

**lemma** *listsum-spread*:

**assumes** *length* *w* = *length* *l*

**shows** *sum-list* (*spread* *w* *l*) = *sum-list* (*map* ( $\lambda$  (*x*, *y*). *int* *x* \* *y*) (*zip* *w* *l*))

**using** *assms*

**proof** (*induct* *w* *arbitrary*: *l*)

**case** *Nil*

**thus** ?*case*

**by** (*simp add: spread-def*)

**next**

**case** (*Cons* *wi* *w'*)

**from**  $\langle \text{length } (wi \# w') = \text{length } l \rangle$

**obtain** *li* *l'* **where**  $l = li \# l'$

**by** (*cases* *l*) *auto*

**show** ?*case*

**using** *Cons*(1)[*of* *l'*] *Cons*(2)

**using**  $\langle l = li \# l' \rangle$

**by** *auto*

**qed**

**lemma** *map-spread*:

*map* *f* (*spread* *w* *l*) = *spread* *w* (*map* *f* *l*)

**unfolding** *spread-def*

**by** (*auto simp add: map-concat comp-def split-def zip-map2*)

**abbreviation** *inc-nth* **where**

*inc-nth* *L* *n*  $\equiv L[n := \text{Suc } (L ! n)]$

**lemma** *take-inc-nth* [*simp*]:

**assumes** *k* < *length* *L*

**shows** *take* *k* (*inc-nth* *L* *k*) = *take* *k* *L*

**using** *assms*

**by** *simp*

**lemma** *take-prefix*:

**assumes** *k* ≤ *k'* *take* *k'* *L* = *take* *k'* *L'*

**shows** *take* *k* *L* = *take* *k* *L'*

**using** *assms*

**by** (*metis* *min.absorb1* *take-take*)

**end**



## 2 Auxiliary lemmas about (finite) sets

```
theory MoreSet
imports Main MoreList
begin
```

```
lemma finiteUn-iff:
  shows finite ( $\bigcup F$ )  $\longleftrightarrow$  finite  $F \wedge (\forall S \in F. \text{finite } S)$ 
proof (auto dest: finite-UnionD)
  fix S
  assume finite ( $\bigcup F$ )  $S \in F$ 
  hence  $S \subseteq \bigcup F$ 
    by auto
  thus finite S
    using  $\langle \text{finite } (\bigcup F) \rangle$ 
    by (auto simp add: finite-subset)
qed
```

```
lemma card-1-iff-singleton:
  shows card A = 1  $\longleftrightarrow$  ( $\exists a. A = \{a\}$ )
proof
  assume card A = 1
  hence A  $\neq \{\}$ 
    by auto
  then obtain a where a  $\in A$ 
    by auto
  hence A = {a}
    using card-Diff-singleton[of A a]  $\langle \text{card } A = 1 \rangle$ 
    using card-0-eq[of A - {a}]
    by (auto simp add: card-ge-0-finite)
  thus  $\exists a. A = \{a\}$ 
    by simp
qed auto
```

```
lemma card-2-iff-dublet:
  shows card A = 2  $\longleftrightarrow$  ( $\exists a \ b. A = \{a, b\} \wedge a \neq b$ )
proof
  assume card A = 2
  hence A  $\neq \{\}$ 
    by auto
  then obtain a where a  $\in A$ 
    by auto
  hence card (A - {a}) = 1
    using  $\langle \text{card } A = 2 \rangle$ 
    using card-Diff-singleton[of A a]
    by (auto simp add: card-ge-0-finite)
  then obtain b where A - {a} = {b}
    using card-1-iff-singleton[of A - {a}]
    by auto
```

hence  $a \neq b$   
 by *auto*  
 moreover  
 hence  $A = \{a, b\}$   
 using  $\langle a \in A \rangle \langle A - \{a\} = \{b\} \rangle$   
 by *auto*  
 ultimately  
 show  $(\exists a\ b. A = \{a, b\} \wedge a \neq b)$   
 by *auto*  
 qed *auto*

lemma *subset-eq-card*:  
 assumes *finite F and card F  $\geq n$*   
 shows  $\exists F'. F' \subseteq F \wedge \text{card } F' = n$   
 using *assms*  
 proof (*induct F rule: finite-induct*)  
 case *empty*  
 thus ?case  
 by *simp*  
 next  
 case (*insert a F*)  
 thus ?case  
 proof (*cases n  $\leq \text{card } F$* )  
 case *True*  
 thus ?thesis  
 using *insert(3)*  
 by *auto*  
 next  
 case *False*  
 hence  $n = \text{card } (\text{insert } a\ F)$   
 using *insert*  
 by *auto*  
 thus ?thesis  
 by *auto*  
 qed  
 qed

lemma *card-le*:  
 assumes  $m \leq n$  *card A = n*  
 shows  $\exists B. B \subseteq A \wedge \text{card } B = m$   
 using *card-eq-0-iff[of A]*  
 using *subset-eq-card[of A m]*  
 using *assms*  
 by (*cases n = 0*) *auto*

lemma *card-Suc*:  
 assumes *card A = n + 1*  
 shows  $\exists A' a. A = A' \cup \{a\} \wedge a \notin A' \wedge \text{card } A' = n$   
 using *assms*

**by** *auto* (*metis card-eq-SucD*)

**lemma** *card-Suc'*:

**assumes**  $\text{card } A = k + 1$   $a \in A$

**shows**  $\text{card } (A - \{a\}) = k$

**using** *assms*

**using** *card.insert[of A - {a} a] card-eq-0-iff[of A]*

**by** *auto*

**lemma** *subset-card-eq*:

**assumes**  $A \subseteq B$  **and**  $\text{card } A = \text{card } B$  **and** *finite B*

**shows**  $A = B$

**proof** –

**obtain** *h* **where** *bij-betw h B A*

**using** *finite-same-card-bij[of B A] assms*

**by** (*auto simp add: finite-subset*)

**thus** *?thesis*

**unfolding** *bij-betw-def*

**using** *endo-inj-surj[of B h]*

**using**  $\langle A \subseteq B \rangle \langle \text{finite } B \rangle$

**by** *auto*

**qed**

**lemma** *card-n-n-aux*:

**assumes**  $A \neq B$  **and**  $\text{card } A = \text{card } B$  **and** *finite A* **and** *finite B*

**shows**  $\text{card } (A \cup B) \neq \text{card } A$

**apply** (*rule ccontr*)

**using** *assms*

**using** *subset-card-eq[of A A  $\cup$  B]*

**using** *subset-card-eq[of B A]*

**by** *auto*

**lemma** *card-n-n*:

**assumes**  $\text{card } A = n$  **and**  $\text{card } B = n$  **and**  $n > 0$  **and**  $A \neq B$

**shows**  $n < \text{card } (A \cup B) \wedge \text{card } (A \cup B) \leq 2 * n$

**using** *assms*

**using** *card-mono[of A A  $\cap$  B]*

**using** *card-Un-Int[of A B]*

**using** *card-gt-0-iff[of A] card-gt-0-iff[of B]*

**using** *card-n-n-aux[of A B]*

**by** *auto*

This should hold for every infinite type — for our applications, it suffices to consider only *nat*.

**lemma** *finite-disjoint-set*:

**fixes**  $X :: \text{nat set}$

**assumes** *finite X*

**shows**  $\exists Y. \text{finite } Y \wedge \text{card } Y = d \wedge X \cap Y = \{\}$

**using** *assms*

```

proof (induct d)
  case 0
  thus ?case
  by force
next
  case (Suc d)
  then obtain Y where card Y = d X ∩ Y = {} finite Y
  by auto
  moreover
  obtain a where a ∉ X ∪ Y
  using finite-nat-set-iff-bounded[of X ∪ Y]
  using ⟨finite X⟩ ⟨finite Y⟩
  by auto
  ultimately
  show ?case
  by (rule-tac x=Y ∪ {a} in exI) auto
qed

```

```

lemma map-fam-cong:
  assumes ∀ x ∈ (⋃ F). g' x = g x
  shows op 'g' F = op 'g' F
using assms
by auto (metis (lifting) image-cong image-iff)+

```

Remove n-th element of an ordered set

```

definition set-drop-nth :: nat ⇒ 'a::linorder set ⇒ 'a set where
  set-drop-nth n s = set (drop-nth n (sorted-list-of-set s))

```

```

lemma set-drop-nth-subset:
  assumes finite X
  shows set-drop-nth s X ⊆ X
unfolding set-drop-nth-def drop-nth-def
using assms
using sorted-list-of-set[of X]
by (auto dest: in-set-takeD in-set-dropD)

```

```

lemma card-set-drop-nth:
  assumes i < card S finite S
  shows card (set-drop-nth i S) = card S - 1
using assms
using distinct-card[of drop-nth i (sorted-list-of-set S)]
using distinct-card[of sorted-list-of-set S]
using distinct-drop-nth[of sorted-list-of-set S i]
using sorted-list-of-set[of S] length-drop-nth[of i sorted-list-of-set S]
unfolding set-drop-nth-def
by auto

```

```

lemma set-drop-nth-drops:
  assumes i < card S finite S

```

```

  shows  $\exists x \in S. x \notin \text{set-drop-nth } i \ S$ 
using assms
unfolding set-drop-nth-def
using sorted-list-of-set[of S] length-sorted-list-of-set[of S]  $\langle \text{finite } S \rangle$ 
using set-drop-nth-distinct[of i sorted-list-of-set S]
using in-set-conv-nth[of sorted-list-of-set S ! i sorted-list-of-set S]
by (rule-tac x=sorted-list-of-set S ! i in beXI, simp-all) force

lemma set-drop-nth-inj:
  assumes finite A i < card A j < card A set-drop-nth i A = set-drop-nth j A
  shows i = j
  using assms
  unfolding set-drop-nth-def
  using set-drop-nth-distinct[of - sorted-list-of-set A]
  using distinct-sorted-list-of-set[of A]
  using nth-eq-iff-index-eq[of sorted-list-of-set A]
  using length-sorted-list-of-set[of A]
  by (metis insertE insert-Diff nth-mem set-drop-nth-def set-drop-nth-drops sorted-list-of-set)

end

```

### 3 Auxiliary lemmas about sum of set operator

```

theory MoreBigOperators
imports Main Rat
begin

```

```

lemma sum-mono-single-lt:
  assumes finite K and
     $\bigwedge i. i \in K \implies f(i::'a) \leq ((g\ i)::rat)$  and
     $a \in K$  and  $f\ a < g\ a$ 
  shows  $(\sum x \in K. f\ x) < (\sum x \in K. g\ x)$ 
proof-
  have  $(\sum x \in K. f\ x) = (\sum x \in (K - \{a\}). f\ x) + f\ a$ 
    using  $\langle a \in K \rangle \langle \text{finite } K \rangle$ 
    by (auto simp add: sum-diff1-ring)
  also have  $\dots \leq (\sum x \in (K - \{a\}). g\ x) + f\ a$ 
    using assms(2)
    by (simp, rule sum-mono, simp)
  also have  $\dots < (\sum x \in (K - \{a\}). g\ x) + g\ a$ 
    using  $\langle a \in K \rangle \langle f\ a < g\ a \rangle$ 
    by simp
  finally show ?thesis
    using assms
    by (auto simp add: sum-diff1-ring)
qed

```

```

lemma sum-mono-single-lt-nat:
  assumes finite K and

```

```

       $\wedge i. i \in K \implies f (i :: 'a) \leq ((g i) :: nat)$  and
       $a \in K$  and  $f a < g a$ 
shows  $(\sum x \in K. f x) < (\sum x \in K. g x)$ 
proof –
  have  $(\sum x \in K. f x) = (\sum x \in (K - \{a\}). f x) + f a$ 
    using  $\langle a \in K \rangle \langle \text{finite } K \rangle$ 
    by  $(\text{auto simp add: sum.remove})$ 
  also have  $\dots \leq (\sum x \in (K - \{a\}). g x) + f a$ 
    using  $\text{assms}(2)$ 
    by  $(\text{simp, rule sum-mono, simp})$ 
  also have  $\dots < (\sum x \in (K - \{a\}). g x) + g a$ 
    using  $\langle a \in K \rangle \langle f a < g a \rangle$ 
    by  $\text{simp}$ 
  finally show  $?thesis$ 
    using  $\text{assms}$ 
    by  $(\text{auto simp add: sum.remove})$ 
qed

```

```

lemma sum-mod:
  assumes finite X and  $\forall x \in X. f x \bmod k = 0$ 
  shows  $(\sum x \in X. f x) \bmod k = (0 :: nat)$ 
using  $\text{assms}$ 
by  $(\text{induct rule: finite-induct}) \text{ auto}$ 

end

```

## 4 Auxiliary lemmas about mappings

```

theory MoreMap
imports HOL-Library.Mapping
begin

```

```

definition tabulate ::  $('a \times 'b) \text{ list} \Rightarrow ('a, 'b) \text{ mapping}$  where
  tabulate  $l = \text{foldl } (\lambda m a. \text{Mapping.update } (\text{fst } a) (\text{snd } a) m) \text{ Mapping.empty } l$ 

```

```

lemma tabulate-map-of:
   $\text{distinct } (\text{map fst } l) \implies \text{tabulate } l = \text{Mapping.Mapping } (\text{map-of } l)$ 
unfolding tabulate-def
apply  $(\text{induct } l \text{ rule: rev-induct})$ 
apply  $(\text{auto simp add: Mapping.empty-def Mapping.update-def})$ 
by  $(\text{metis dom-map-of-conv-image-fst empty-map-add lookup.abs-eq lookup.rep-eq map-add-upd-left})$ 

```

```

find-theorems Mapping.lookup Mapping.tabulate

```

```

lemma lookup-tabulate [simp]:
   $\text{Mapping.lookup } (\text{Mapping.tabulate } ks f) = (\text{Some } o f) \mid \text{' set } ks$ 
  by  $(\text{induct } ks) (\text{auto simp add: Mapping.tabulate.abs-eq Mapping.lookup.abs-eq})$ 

```

*restrict-map-def fun-eq-iff*)

end

## 5 Auxiliary lemmas about natural numbers

**theory** *MoreNat*

**imports** *Main*

**begin**

**lemma** *power-mod*:  $\llbracket k > (1::nat); x > 0 \rrbracket \implies (k \wedge x) \bmod k = 0$

**by** (*induct x*) *auto*

end

## 6 Auxiliary lemmas about bijective functions

**theory** *MoreFun*

**imports** *Main MoreSet*

**begin**

### 6.0.1 Extending bijective functions

**lemma** *bij-betw-extend*:

**fixes**  $X\ X' :: 'a\ set$  **and**  $Y\ Y' :: 'b\ set$

**assumes** *bij-betw*  $f\ X\ Y$  **and**  $card\ X' = card\ Y'$  **and** *finite*  $X'\ finite\ Y'$  **and**

$X \cap X' = \{\}$   $Y \cap Y' = \{\}$

**shows**  $\exists f'.\ bij\_betw\ f'\ (X \cup X')\ (Y \cup Y') \wedge (\forall x \in X.\ f\ x = f'\ x)$

**proof**–

**obtain**  $fa$  **where** *bij-betw*  $fa\ X'\ Y'$

**using** *finite-same-card-bij*[*of*  $X'\ Y'$ ]  $\langle finite\ X' \rangle \langle finite\ Y' \rangle \langle card\ X' = card\ Y' \rangle$

**by** *auto*

**let**  $?f = \lambda x.\ if\ x \in X\ then\ f\ x\ else\ fa\ x$

**have** *bij-betw*  $?f\ X\ Y$

**using**  $\langle bij\_betw\ f\ X\ Y \rangle$

**unfolding** *bij-betw-def inj-on-def*

**by** *auto*

**moreover**

**have** *bij-betw*  $?f\ X'\ Y'$

**using**  $\langle bij\_betw\ fa\ X'\ Y' \rangle \langle X \cap X' = \{\} \rangle$

**unfolding** *bij-betw-def inj-on-def*

**by** *auto*

**ultimately**

**show** *?thesis*

**apply** (*rule-tac*  $x=?f$  **in** *exI*)

**using** *bij-betw-combine*[*of*  $?f\ X\ Y\ X'\ Y'$ ]  $\langle X \cap X' = \{\} \rangle \langle Y \cap Y' = \{\} \rangle$

**by** *auto*

qed

This should hold for every infinite type — for our applications, it suffices to consider only *nat*.

**lemma** *nat-set-extend*:

```

assumes finite X
shows  $\exists X'::\text{nat set. } \text{finite } X' \wedge \text{card } X' = n \wedge X \cap X' = \{\}$ 
proof–
  let  $?X' = \{(Max\ X) + 1 ..< (Max\ X) + 1 + n\}$ 
  have  $X \cap ?X' = \{\}$ 
  proof (auto)
    fix x
    assume  $x \in X \ x \geq Suc\ (Max\ X)$ 
    have  $x \leq Max\ X$ 
    using  $\langle \text{finite } X \rangle \langle x \in X \rangle \text{Max-ge}[of\ X\ x]$ 
    by simp
    show False
    using  $\langle x \geq Suc\ (Max\ X) \rangle \langle x \leq Max\ X \rangle$ 
    by simp
  qed
  thus ?thesis
  by (rule-tac x=?X' in exI) simp-all
qed

```

**lemma** *bij-betw-inj-extend*:

```

fixes  $X::'a\ \text{set}$  and  $Y::\text{nat set}$ 
assumes bij-betw f X Y and finite Y and finite X'
shows  $\exists f'. \text{inj-on } f' (X \cup X') \wedge (\forall x \in X. f' x = f x)$ 
proof–
  let  $?X' = X' - X$ 
  obtain Y' where  $\text{card } ?X' = \text{card } Y' \ \text{finite } Y' \ Y \cap Y' = \{\}$ 
  using nat-set-extend[of Y card ?X']
  using  $\langle \text{finite } X' \rangle \langle \text{finite } Y \rangle$ 
  by force
  thus ?thesis
  using assms
  using bij-betw-extend[of f X Y ?X' Y']
  unfolding bij-betw-def
  by auto
qed

```

**lemma** *bij-betw-complement*:

```

assumes  $B \subseteq X \ B' \subseteq X \ A \subseteq X$  finite X
  bij-betw f B B'  $\text{card } A \leq \text{card } X$ 
shows  $\exists A'. \text{card } (A - B) = \text{card } (A' - B') \wedge A' \subseteq X$ 
proof–
  have finite B finite B' finite A
  using  $\langle B' \subseteq X \rangle \langle B \subseteq X \rangle \langle A \subseteq X \rangle \langle \text{finite } X \rangle$ 
  by (auto simp add: finite-subset)
  have  $\text{card } B = \text{card } B'$ 
  using  $\langle \text{bij-betw } f\ B\ B' \rangle$ 

```



```

    unfolding bij-betw-def
    by (metis card-image)
  hence  $\text{card } (X - B') = \text{card } X - \text{card } B$ 
    using  $\langle \text{finite } B' \rangle \langle B' \subseteq X \rangle$ 
    using card-Diff-subset[of  $B' X$ ]
    by auto
  have  $\text{card } (A - B) = \text{card } A - \text{card } (A \cap B)$ 
    using card-Diff-subset-Int[of  $A B$ ]
    using finite-subset[of  $A \cap B A$ ]  $\langle \text{finite } A \rangle$ 
    by simp
  have  $\text{card } B - \text{card } (A \cap B) = \text{card } (B - A)$ 
    using card-Diff-subset-Int[of  $B A$ ] finite-subset[of  $A \cap B A$ ]  $\langle \text{finite } A \rangle$ 
    by (auto simp add: Int-commute)
  have  $\text{card } B \leq \text{card } X$ 
    using card-mono[of  $X B$ ]  $\langle B \subseteq X \rangle \langle \text{finite } X \rangle$ 
    by auto
  have  $\text{card } (A \cap B) \leq \text{card } A$ 
    using card-mono[of  $A A \cap B$ ]  $\langle \text{finite } A \rangle$ 
    by simp
  have  $\text{card } (A \cup B) \leq \text{card } X$ 
    using  $\langle A \subseteq X \rangle \langle B \subseteq X \rangle \langle \text{finite } X \rangle$ 
    using card-mono[of  $X A \cup B$ ]
    by auto
  have  $\text{card } B - \text{card } (A \cap B) \leq \text{card } X - \text{card } A$ 
    using card-Un-Int[of  $A B$ ]  $\langle \text{card } (A \cup B) \leq \text{card } X \rangle \langle \text{finite } A \rangle \langle \text{finite } B \rangle$ 
    by auto
  have  $\exists F' \subseteq X - B'. \text{card } F' = \text{card } (A - B)$ 
  proof (rule subset-eq-card)
    show finite  $(X - B')$ 
      using finite-subset[of  $X - B' X$ ]  $\langle \text{finite } X \rangle$ 
      by auto
  next
    show  $\text{card } (A - B) \leq \text{card } (X - B')$ 
      using  $\langle \text{card } B - \text{card } (A \cap B) = \text{card } (B - A) \rangle$ 
      using  $\langle \text{card } (X - B') = \text{card } X - \text{card } B \rangle \langle \text{card } (A - B) = \text{card } A - \text{card } (A \cap B) \rangle$ 
      using  $\langle \text{card } B \leq \text{card } X \rangle \langle \text{card } (A \cap B) \leq \text{card } A \rangle$ 
      using  $\langle \text{card } B - \text{card } (A \cap B) \leq \text{card } X - \text{card } A \rangle \langle \text{card } A \leq \text{card } X \rangle$ 
      by auto
  qed
  then obtain  $F'$  where  $F' \subseteq (X - B') \text{card } F' = \text{card } (A - B)$ 
    by auto
  thus ?thesis
    apply (rule-tac  $x=B' \cup F'$  in exI)
    apply (subgoal-tac  $B' \cup F' - B' = F'$ )
    using  $\langle B' \subseteq X \rangle \langle \text{card } F' = \text{card } (A - B) \rangle$ 
    by auto
qed

```

## 6.0.2 Bijections between the domain of a set family and natural numbers.

**lemma** *bij-card*:  
**assumes** *bij-betw*  $h$   $(\bigcup F)$   $\{0..<\text{card } (\bigcup F)\}$   
**assumes**  $A \in F$   
**shows**  $\text{card } A = \text{card } (h \text{ ` } A)$   
**using** *assms*  
**using** *card-image*[*of*  $h$   $A$ ]  
**unfolding** *bij-betw-def inj-on-def*  
**by** *auto*

**lemma** *bij-card-Int*:  
**assumes** *bij-betw*  $h$   $(\bigcup F)$   $\{0..<\text{card } (\bigcup F)\}$   
**assumes**  $A \in F$  **and**  $B \in F$   
**shows**  $\text{card } (A \cap B) = \text{card } (h \text{ ` } A \cap h \text{ ` } B)$   
**proof** –  
**have**  $A \subseteq \bigcup F$   $B \subseteq \bigcup F$   
**using**  $\langle A \in F \rangle \langle B \in F \rangle$   
**by** *auto*  
**thus** *?thesis*  
**using** *assms*  
**using** *inj-on-image-Int*[*of*  $h$   $\bigcup F$   $A$   $B$ , *THEN sym*]  
**using** *card-image*[*of*  $h$   $A \cap B$ ]  
**unfolding** *bij-betw-def inj-on-def*  
**by** *simp*  
**qed**

**lemma** *bij-card-Un*:  
**assumes** *bij-betw*  $h$   $(\bigcup F)$   $\{0..<\text{card } (\bigcup F)\}$   
**assumes**  $A \subseteq F$   
**shows**  $\text{card } (\bigcup A) = \text{card } (\bigcup (op \text{ ` } h \text{ ` } A))$   
**proof** –  
**have**  $h \text{ ` } (\bigcup A) = \bigcup (op \text{ ` } h \text{ ` } A)$   
**by** *auto*  
**moreover**  
**have** *inj-on*  $h$   $(\bigcup A)$   
**using**  $\langle \text{bij-betw } h (\bigcup F) \{0..<\text{card } (\bigcup F)\} \rangle \langle A \subseteq F \rangle$   
**unfolding** *bij-betw-def*  
**using** *subset-inj-on*[*of*  $h$   $\bigcup F$   $\bigcup A$ ]  
**by** *auto*  
**ultimately**  
**show** *?thesis*  
**using** *card-image*[*of*  $h$   $\bigcup A$ ]  
**by** *simp*  
**qed**

**lemma** *bij-card-image-inj*:  
**assumes** *bij-betw*  $h$   $(\bigcup F)$   $\{0..<\text{card } (\bigcup F)\}$   
**shows** *inj-on*  $(op \text{ ` } h) F$

```

unfolding inj-on-def
proof (rule ballI, rule ballI)
  fix x y
  assume  $x \in F$   $y \in F$ 
  thus  $h \text{ ` } x = h \text{ ` } y \longrightarrow x = y$ 
    using inj-on-Un-image-eq-iff[of h x y] assms
    unfolding bij-betw-def inj-on-def
    by auto
qed

```

### 6.0.3 Bijections between distinct lists' elements and natural numbers

**definition** *swap* **where**  
 $swap\ f\ a\ b = f\ (a := f\ b, b := f\ a)$

**lemma** *bij-betw-swap*:  
**assumes** *bij-betw*  $f\ A\ B$  **and**  $a \in A$  **and**  $b \in A$   
**shows** *bij-betw*  $(swap\ f\ a\ b)\ A\ B$   
**proof** (*cases*  $a = b$ )  
**case** *True*  
**thus** ?thesis  
**unfolding** *swap-def*  
**using** *assms*  
**by** *simp*  
**next**  
**case** *False*  
**show** ?thesis  
**unfolding** *bij-betw-def*  
**proof**  
**let** ?f = *swap*  $f\ a\ b$   
**show** *inj-on* ?f  $A$   
**unfolding** *inj-on-def*  
**proof** (*safe*)  
**fix**  $x\ y$   
**assume**  $x \in A$   $y \in A$  ?f  $x = ?f\ y$   
**thus**  $x = y$   
**using** *assms*  
**unfolding** *swap-def* *bij-betw-def* *inj-on-def*  
**by** (*auto* *split: if-split-asm*)  
**qed**

**show** *swap*  $f\ a\ b \text{ ` } A = B$   
**unfolding** *swap-def*  
**using** *assms*  
**unfolding** *bij-betw-def*  
**proof** (*auto*)  
**fix**  $x$   
**assume**  $x \in A$

```

show  $f\ x \in f(a := f\ b, b := f\ a) \text{ ‘ } A$ 
proof (cases  $x = a$ )
  case True
    show ?thesis
      apply (rule rev-image-eqI[of b])
      using  $\langle x = a \rangle \langle b \in A \rangle \langle a \neq b \rangle$ 
      by simp-all
  next
  case False
    show ?thesis
    proof (cases  $x = b$ )
      case True
        show ?thesis
          apply (rule rev-image-eqI[of a])
          using  $\langle x = b \rangle \langle a \in A \rangle \langle a \neq b \rangle$ 
          by simp-all
      next
      case False
        show ?thesis
          apply (rule rev-image-eqI[of x])
          using  $\langle x \in A \rangle \langle x \neq a \rangle \langle x \neq b \rangle$ 
          by simp-all
    qed
  qed
qed
qed
qed
qed

lemma ex-bij-betw-nat-finite-list:
  assumes distinct l and set  $l \subseteq X$  and finite X
  shows  $\exists\ b. \text{bij-betw } b\ \{0..<\text{card } X\}\ X \wedge (\forall\ i. i < \text{length } l \longrightarrow b\ i = l\ !\ i)$ 
using assms
proof (induct  $l$  rule: rev-induct)
  case Nil
    from  $\langle \text{finite } X \rangle$  obtain  $b$  where  $\text{bij-betw } b\ \{0..<\text{card } X\}\ X$ 
    using ex-bij-betw-nat-finite
    by auto
  thus ?case
    by auto
  next
  case (snoc  $a\ l'$ )
    then obtain  $b$  where  $\text{bij-betw } b\ \{0..<\text{card } X\}\ X \wedge \forall i < \text{length } l'. b\ i = l'\ !\ i$ 
    by auto

  have  $a \in b \text{ ‘ } \{0..<\text{card } X\}$ 
    using  $\langle \text{bij-betw } b\ \{0..<\text{card } X\}\ X \rangle \text{snoc}(3)$ 
    unfolding bij-betw-def
    by auto
  then obtain  $k$  where  $k \in \{0..<\text{card } X\}\ b\ k = a$ 

```

```

by auto

let ?n = length l'

have ?n ∈ {0.. $\text{card } X$ }
  using snoc(2) snoc(3)
  using distinct-card[of l' @ [a]]
  using card-mono[of X set (l' @ [a])] ⟨finite X⟩
  by auto

show ?case
proof(rule exI, rule conjI)
  let ?b = swap b k ?n
  show bij-betw ?b {0.. $\text{card } X$ } X
    apply (rule bij-betw-swap)
    using ⟨bij-betw b {0.. $\text{card } X$ } X⟩ ⟨?n ∈ {0.. $\text{card } X$ }⟩ ⟨k ∈ {0.. $\text{card } X$ }⟩
    by simp-all

  have length l' ≤ k
    apply (rule ccontr)
    using ⟨∀ i < length l'. b i = l' ! i⟩ ⟨b k = a⟩ snoc(2)
    by auto

  show ∀ i < length (l' @ [a]). ?b i = (l' @ [a]) ! i
    using ⟨∀ i < length l'. b i = l' ! i⟩ ⟨b k = a⟩ ⟨k ≥ length l'⟩
    unfolding swap-def
    by (auto simp add: nth-append)
qed
qed

lemma ex-bij-betw-nat-finite-list-inv:
  assumes distinct l and set l ⊆ X and finite X
  shows ∃ b. bij-betw b X {0.. $\text{card } X$ } ∧ (∀ i. i < length l ⟶ b (l ! i) = i)
proof-
  from assms
  obtain b where bij-betw b {0.. $\text{card } X$ } X ∀ i < length l. b i = l ! i
    using ex-bij-betw-nat-finite-list[of l X]
    by auto
  let ?b = inv-into {0.. $\text{card } X$ } b
  show ?thesis
  proof (rule-tac x=?b in exI, safe)
    show bij-betw ?b X {0.. $\text{card } X$ }
      using ⟨bij-betw b {0.. $\text{card } X$ } X⟩
      by (rule bij-betw-inv-into)
  next
    fix i
    assume i < length l
    hence l ! i = b i
      using ⟨∀ i < length l. b i = l ! i⟩

```

```

    by simp
  moreover
  have  $i \in \{0..<\text{card } X\}$ 
  proof-
    have  $\text{length } l \leq \text{card } X$ 
    using assms
    using distinct-card[of  $l$ ] card-mono[of  $X$  set  $l$ ]
    by simp
    thus ?thesis
    using  $\langle i < \text{length } l \rangle$ 
    by auto
  qed
  ultimately
  show ?b  $(l ! i) = i$ 
    using  $\langle \text{bij-betw } b \{0..<\text{card } X\} X \rangle$ 
    unfolding bij-betw-def
    by auto
  qed
qed

end
theory MoreBinomial
imports Binomial
begin
lemma choose-mono-Suc:
  shows  $\text{Suc } n \text{ choose } k \geq n \text{ choose } k$ 
  using binomial-Suc-Suc[of  $n \ k-1$ ]
  by (cases  $k=0$ ) auto

lemma choose-mono:
  assumes  $m \geq n$ 
  shows  $m \text{ choose } k \geq n \text{ choose } k$ 
using assms
proof (induct  $m-n$  arbitrary:  $m \ n$ )
  case 0
  thus ?case
  by auto
next
  case (Suc  $r$ )
  show ?case
    using Suc(1)[of  $m \ \text{Suc } n$ ] Suc(2) Suc(3)
    using choose-mono-Suc[of  $n \ k$ ]
    by force
qed

lemma n-subsets-ub:
  assumes  $\bigcup F \subseteq \{0..<n\}$ 
  shows  $\text{card } \{A \in F. \text{card } A = k\} \leq n \text{ choose } k$ 
proof-

```

```

have  $\{A \in F. \text{card } A = k\} \subseteq \{B. B \subseteq \bigcup F \wedge \text{card } B = k\}$ 
  by auto
hence  $\text{card } \{A \in F. \text{card } A = k\} \leq \text{card } \{B. B \subseteq \bigcup F \wedge \text{card } B = k\}$ 
  using card-mono[of  $\{B. B \subseteq \bigcup F \wedge \text{card } B = k\} \{A \in F. \text{card } A = k\}$ ] assms
  by (auto simp add: finite-subset)
moreover
have  $\text{card } (\bigcup F) \text{ choose } k \leq n \text{ choose } k$ 
  using assms choose-mono card-mono[of  $\{0..<n\} \bigcup F$ ]
  by simp
ultimately
show ?thesis
  using assms
  using n-subsets[of  $\bigcup F k$ ]
  by (auto simp add: finite-subset)
qed

end

```