

# List Index

Tobias Nipkow

June 17, 2024

## Abstract

This theory provides functions for finding the index of an element in a list, by predicate and by value.

## 1 Index-based manipulation of lists

**theory** *List-Index* **imports** *Main* **begin**

This theory collects functions for index-based manipulation of lists.

### 1.1 Finding an index

This subsection defines three functions for finding the index of items in a list:

*find-index*  $P$   $xs$  finds the index of the first element in  $xs$  that satisfies  $P$ .

*index*  $xs$   $x$  finds the index of the first occurrence of  $x$  in  $xs$ .

*last-index*  $xs$   $x$  finds the index of the last occurrence of  $x$  in  $xs$ .

All functions return *length*  $xs$  if  $xs$  does not contain a suitable element.

The argument order of *find-index* follows the function of the same name in the Haskell standard library. For *index* (and *last-index*) the order is intentionally reversed: *index* maps lists to a mapping from elements to their indices, almost the inverse of function *nth*.

**primrec** *find-index* :: ('a  $\Rightarrow$  bool)  $\Rightarrow$  'a list  $\Rightarrow$  nat **where**  
*find-index* - [] = 0 |  
*find-index*  $P$  (x#xs) = (if  $P$  x then 0 else *find-index*  $P$  xs + 1)

**definition** *index* :: 'a list  $\Rightarrow$  'a  $\Rightarrow$  nat **where**  
*index* xs = ( $\lambda a$ . *find-index* ( $\lambda x$ . x=a) xs)

**definition** *last-index* :: 'a list  $\Rightarrow$  'a  $\Rightarrow$  nat **where**  
*last-index* xs x =  
(let i = *index* (rev xs) x; n = *size* xs

in if  $i = n$  then  $i$  else  $n - (i+1)$ )

**lemma** *find-index-append*:  $\text{find-index } P (xs @ ys) =$   
(if  $\exists x \in \text{set } xs. P x$  then  $\text{find-index } P xs$  else  $\text{size } xs + \text{find-index } P ys$ )  
(proof)

**lemma** *find-index-le-size*:  $\text{find-index } P xs \leq \text{size } xs$   
(proof)

**lemma** *index-le-size*:  $\text{index } xs x \leq \text{size } xs$   
(proof)

**lemma** *last-index-le-size*:  $\text{last-index } xs x \leq \text{size } xs$   
(proof)

**lemma** *index-Nil[simp]*:  $\text{index } [] a = 0$   
(proof)

**lemma** *index-Cons[simp]*:  $\text{index } (x \# xs) a = (\text{if } x=a \text{ then } 0 \text{ else } \text{index } xs a + 1)$   
(proof)

**lemma** *index-append*:  $\text{index } (xs @ ys) x =$   
(if  $x : \text{set } xs$  then  $\text{index } xs x$  else  $\text{size } xs + \text{index } ys x$ )  
(proof)

**lemma** *index-conv-size-if-notin[simp]*:  $x \notin \text{set } xs \implies \text{index } xs x = \text{size } xs$   
(proof)

**lemma** *find-index-eq-size-conv*:  
 $\text{size } xs = n \implies (\text{find-index } P xs = n) = (\forall x \in \text{set } xs. \sim P x)$   
(proof)

**lemma** *size-eq-find-index-conv*:  
 $\text{size } xs = n \implies (n = \text{find-index } P xs) = (\forall x \in \text{set } xs. \sim P x)$   
(proof)

**lemma** *index-size-conv*:  $\text{size } xs = n \implies (\text{index } xs x = n) = (x \notin \text{set } xs)$   
(proof)

**lemma** *size-index-conv*:  $\text{size } xs = n \implies (n = \text{index } xs x) = (x \notin \text{set } xs)$   
(proof)

**lemma** *last-index-size-conv*:  
 $\text{size } xs = n \implies (\text{last-index } xs x = n) = (x \notin \text{set } xs)$   
(proof)

**lemma** *size-last-index-conv*:  
 $\text{size } xs = n \implies (n = \text{last-index } xs x) = (x \notin \text{set } xs)$   
(proof)

**lemma** *find-index-less-size-conv*:  
 $(\text{find-index } P \text{ } xs < \text{size } xs) = (\exists x \in \text{set } xs. P \ x)$   
 <proof>

**lemma** *index-less-size-conv*:  
 $(\text{index } xs \ x < \text{size } xs) = (x \in \text{set } xs)$   
 <proof>

**lemma** *last-index-less-size-conv*:  
 $(\text{last-index } xs \ x < \text{size } xs) = (x : \text{set } xs)$   
 <proof>

**lemma** *index-less[simp]*:  
 $x : \text{set } xs \implies \text{size } xs \leq n \implies \text{index } xs \ x < n$   
 <proof>

**lemma** *last-index-less[simp]*:  
 $x : \text{set } xs \implies \text{size } xs \leq n \implies \text{last-index } xs \ x < n$   
 <proof>

**lemma** *last-index-Cons*:  $\text{last-index } (x\#\text{xs}) \ y =$   
 (if  $x=y$  then  
   if  $x \in \text{set } xs$  then  $\text{last-index } xs \ y + 1$  else 0  
   else  $\text{last-index } xs \ y + 1$ )  
 <proof>

**lemma** *last-index-append*:  $\text{last-index } (xs \ @ \ ys) \ x =$   
 (if  $x : \text{set } ys$  then  $\text{size } xs + \text{last-index } ys \ x$   
   else if  $x : \text{set } xs$  then  $\text{last-index } xs \ x$  else  $\text{size } xs + \text{size } ys$ )  
 <proof>

**lemma** *last-index-Snoc[simp]*:  
 $\text{last-index } (xs \ @ \ [x]) \ y =$   
 (if  $x=y$  then  $\text{size } xs$   
   else if  $y : \text{set } xs$  then  $\text{last-index } xs \ y$  else  $\text{size } xs + 1$ )  
 <proof>

**lemma** *nth-find-index*:  $\text{find-index } P \ xs < \text{size } xs \implies P(xs \ ! \ \text{find-index } P \ xs)$   
 <proof>

**lemma** *nth-index[simp]*:  $x \in \text{set } xs \implies xs \ ! \ \text{index } xs \ x = x$   
 <proof>

**lemma** *nth-last-index[simp]*:  $x \in \text{set } xs \implies xs \ ! \ \text{last-index } xs \ x = x$   
 <proof>

**lemma** *index-rev*:  $\llbracket \text{distinct } xs; x \in \text{set } xs \rrbracket \implies$   
 $\text{index } (\text{rev } xs) \ x = \text{length } xs - \text{index } xs \ x - 1$

*<proof>*

**lemma** *index-nth-id*:

$\llbracket \text{distinct } xs; n < \text{length } xs \rrbracket \implies \text{index } xs (xs ! n) = n$   
*<proof>*

**lemma** *index-upt[simp]*:  $m \leq i \implies i < n \implies \text{index } [m..<n] i = i - m$   
*<proof>*

**lemma** *index-eq-index-conv[simp]*:  $x \in \text{set } xs \vee y \in \text{set } xs \implies$   
 $(\text{index } xs x = \text{index } xs y) = (x = y)$   
*<proof>*

**lemma** *last-index-eq-index-conv[simp]*:  $x \in \text{set } xs \vee y \in \text{set } xs \implies$   
 $(\text{last-index } xs x = \text{last-index } xs y) = (x = y)$   
*<proof>*

**lemma** *inj-on-index*:  $\text{inj-on } (\text{index } xs) (\text{set } xs)$   
*<proof>*

**lemma** *inj-on-index2*:  $I \subseteq \text{set } xs \implies \text{inj-on } (\text{index } xs) I$   
*<proof>*

**lemma** *inj-on-last-index*:  $\text{inj-on } (\text{last-index } xs) (\text{set } xs)$   
*<proof>*

**lemma** *find-index-conv-takeWhile*:  
 $\text{find-index } P xs = \text{size}(\text{takeWhile } (\text{Not } o P) xs)$   
*<proof>*

**lemma** *index-conv-takeWhile*:  $\text{index } xs x = \text{size}(\text{takeWhile } (\lambda y. x \neq y) xs)$   
*<proof>*

**lemma** *find-index-first*:  $i < \text{find-index } P xs \implies \neg P (xs ! i)$   
*<proof>*

**lemma** *index-first*:  $i < \text{index } xs x \implies x \neq xs ! i$   
*<proof>*

**lemma** *find-index-eqI*:  
**assumes**  $i \leq \text{length } xs$   
**assumes**  $\forall j < i. \neg P (xs ! j)$   
**assumes**  $i < \text{length } xs \implies P (xs ! i)$   
**shows**  $\text{find-index } P xs = i$   
*<proof>*

**lemma** *find-index-eq-iff*:  
 $\text{find-index } P xs = i$   
 $\longleftrightarrow (i \leq \text{length } xs \wedge (\forall j < i. \neg P (xs ! j)) \wedge (i < \text{length } xs \longrightarrow P (xs ! i)))$

*<proof>*

**lemma** *index-eqI*:

**assumes**  $i \leq \text{length } xs$

**assumes**  $\forall j < i. xs!j \neq x$

**assumes**  $i < \text{length } xs \implies xs!i = x$

**shows**  $\text{index } xs \ x = i$

*<proof>*

**lemma** *index-eq-iff*:

$\text{index } xs \ x = i$

$\longleftrightarrow (i \leq \text{length } xs \wedge (\forall j < i. xs!j \neq x) \wedge (i < \text{length } xs \longrightarrow xs!i = x))$

*<proof>*

**lemma** *index-take*:  $\text{index } xs \ x \geq i \implies x \notin \text{set}(\text{take } i \ xs)$

*<proof>*

**lemma** *last-index-drop*:

$\text{last-index } xs \ x < i \implies x \notin \text{set}(\text{drop } i \ xs)$

*<proof>*

**lemma** *set-take-if-index*: **assumes**  $\text{index } xs \ x < i$  **and**  $i \leq \text{length } xs$

**shows**  $x \in \text{set}(\text{take } i \ xs)$

*<proof>*

**lemma** *index-take-if-index*:

**assumes**  $\text{index } xs \ x \leq n$  **shows**  $\text{index}(\text{take } n \ xs) \ x = \text{index } xs \ x$

*<proof>*

**lemma** *index-take-if-set*:

$x \in \text{set}(\text{take } n \ xs) \implies \text{index}(\text{take } n \ xs) \ x = \text{index } xs \ x$

*<proof>*

**lemma** *index-last[simp]*:

$xs \neq [] \implies \text{distinct } xs \implies \text{index } xs \ (\text{last } xs) = \text{length } xs - 1$

*<proof>*

**lemma** *index-update-if-diff2*:

$n < \text{length } xs \implies x \neq xs!n \implies x \neq y \implies \text{index}(xs[n := y]) \ x = \text{index } xs \ x$

*<proof>*

**lemma** *set-drop-if-index*:  $\text{distinct } xs \implies \text{index } xs \ x < i \implies x \notin \text{set}(\text{drop } i \ xs)$

*<proof>*

**lemma** *index-swap-if-distinct*: **assumes**  $\text{distinct } xs$   $i < \text{size } xs$   $j < \text{size } xs$

**shows**  $\text{index}(xs[i := xs!j, j := xs!i]) \ x =$

$(\text{if } x = xs!i \text{ then } j \text{ else if } x = xs!j \text{ then } i \text{ else } \text{index } xs \ x)$

*<proof>*

**lemma** *bij-betw-index*:

$distinct\ xs \implies X = set\ xs \implies l = size\ xs \implies bij\ betw\ (index\ xs)\ X\ \{0..<l\}$   
*<proof>*

**lemma** *index-image*:  $distinct\ xs \implies set\ xs = X \implies index\ xs\ 'X = \{0..<size\ xs\}$   
*<proof>*

**lemma** *index-map-inj-on*:

$\llbracket inj\ on\ f\ S; y \in S; set\ xs \subseteq S \rrbracket \implies index\ (map\ f\ xs)\ (f\ y) = index\ xs\ y$   
*<proof>*

**lemma** *index-map-inj*:  $inj\ f \implies index\ (map\ f\ xs)\ (f\ y) = index\ xs\ y$   
*<proof>*

## 1.2 Map with index

**primrec** *map-index'* ::  $nat \Rightarrow (nat \Rightarrow 'a \Rightarrow 'b) \Rightarrow 'a\ list \Rightarrow 'b\ list$  **where**  
 $map\ index'\ n\ f\ [] = []$   
 $| map\ index'\ n\ f\ (x\#\!xs) = f\ n\ x\ \# map\ index'\ (Suc\ n)\ f\ xs$

**lemma** *length-map-index'[simp]*:  $length\ (map\ index'\ n\ f\ xs) = length\ xs$   
*<proof>*

**lemma** *map-index'-map-zip*:  $map\ index'\ n\ f\ xs = map\ (case\ prod\ f)\ (zip\ [n..<n + length\ xs]\ xs)$   
*<proof>*

**abbreviation** *map-index*  $\equiv map\ index'\ 0$

**lemmas** *map-index* = *map-index'-map-zip*[of 0, simplified]

**lemma** *take-map-index*:  $take\ p\ (map\ index\ f\ xs) = map\ index\ f\ (take\ p\ xs)$   
*<proof>*

**lemma** *drop-map-index*:  $drop\ p\ (map\ index\ f\ xs) = map\ index'\ p\ f\ (drop\ p\ xs)$   
*<proof>*

**lemma** *map-map-index[simp]*:  $map\ g\ (map\ index\ f\ xs) = map\ index\ (\lambda n\ x.\ g\ (f\ n\ x))\ xs$   
*<proof>*

**lemma** *map-index-map[simp]*:  $map\ index\ f\ (map\ g\ xs) = map\ index\ (\lambda n\ x.\ f\ n\ (g\ x))\ xs$   
*<proof>*

**lemma** *set-map-index[simp]*:  $x \in set\ (map\ index\ f\ xs) = (\exists i < length\ xs.\ f\ i\ (xs\ !\ i) = x)$   
*<proof>*

**lemma** *set-map-index'[simp]*:  $x \in \text{set } (\text{map-index}' n f xs)$   
 $\longleftrightarrow (\exists i < \text{length } xs. f (n+i) (xs!i) = x)$   
 ⟨proof⟩

**lemma** *nth-map-index[simp]*:  $p < \text{length } xs \implies \text{map-index } f xs ! p = f p (xs ! p)$   
 ⟨proof⟩

**lemma** *map-index-cong*:  
 $\forall p < \text{length } xs. f p (xs ! p) = g p (xs ! p) \implies \text{map-index } f xs = \text{map-index } g xs$   
 ⟨proof⟩

**lemma** *map-index-id*:  $\text{map-index } (\text{curry snd}) xs = xs$   
 ⟨proof⟩

**lemma** *map-index-no-index[simp]*:  $\text{map-index } (\lambda n x. f x) xs = \text{map } f xs$   
 ⟨proof⟩

**lemma** *map-index-congL*:  
 $\forall p < \text{length } xs. f p (xs ! p) = xs ! p \implies \text{map-index } f xs = xs$   
 ⟨proof⟩

**lemma** *map-index'-is-NilD*:  $\text{map-index}' n f xs = [] \implies xs = []$   
 ⟨proof⟩

**declare** *map-index'-is-NilD*[of 0, dest!]

**lemma** *map-index'-is-ConsD*:  
 $\text{map-index}' n f xs = y \# ys \implies \exists z zs. xs = z \# zs \wedge f n z = y \wedge \text{map-index}' (n + 1) f zs = ys$   
 ⟨proof⟩

**lemma** *map-index'-eq-imp-length-eq*:  $\text{map-index}' n f xs = \text{map-index}' n g ys \implies \text{length } xs = \text{length } ys$   
 ⟨proof⟩

**lemmas** *map-index-eq-imp-length-eq* = *map-index'-eq-imp-length-eq*[of 0]

**lemma** *map-index'-comp[simp]*:  $\text{map-index}' n f (\text{map-index}' n g xs) = \text{map-index}' n (\lambda n. f n o g n) xs$   
 ⟨proof⟩

**lemma** *map-index'-append[simp]*:  $\text{map-index}' n f (a @ b)$   
 $= \text{map-index}' n f a @ \text{map-index}' (n + \text{length } a) f b$   
 ⟨proof⟩

**lemma** *map-index-append[simp]*:  $\text{map-index } f (a @ b)$   
 $= \text{map-index } f a @ \text{map-index}' (\text{length } a) f b$   
 ⟨proof⟩

### 1.3 Insert at position

**primrec** *insert-nth* ::  $\text{nat} \Rightarrow 'a \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list}$  **where**

*insert-nth* 0  $x$   $xs = x \# xs$

| *insert-nth* (Suc  $n$ )  $x$   $xs = (\text{case } xs \text{ of } [] \Rightarrow [x] \mid y \# ys \Rightarrow y \# \text{insert-nth } n \ x \ ys)$

**lemma** *insert-nth-take-drop*[simp]:  $\text{insert-nth } n \ x \ xs = \text{take } n \ xs \ @ \ [x] \ @ \ \text{drop } n \ xs$   
<proof>

**lemma** *length-insert-nth*:  $\text{length } (\text{insert-nth } n \ x \ xs) = \text{Suc } (\text{length } xs)$   
<proof>

**lemma** *set-insert-nth*:  
 $\text{set } (\text{insert-nth } i \ x \ xs) = \text{insert } x \ (\text{set } xs)$   
<proof>

**lemma** *distinct-insert-nth*:  
**assumes** *distinct*  $xs$   
**assumes**  $x \notin \text{set } xs$   
**shows** *distinct* ( $\text{insert-nth } i \ x \ xs$ )  
<proof>

**lemma** *nth-insert-nth-front*:  
**assumes**  $i < j \ j \leq \text{length } xs$   
**shows**  $\text{insert-nth } j \ x \ xs ! i = xs ! i$   
<proof>

**lemma** *nth-insert-nth-index-eq*:  
**assumes**  $i \leq \text{length } xs$   
**shows**  $\text{insert-nth } i \ x \ xs ! i = x$   
<proof>

**lemma** *nth-insert-nth-back*:  
**assumes**  $j < i \ i \leq \text{length } xs$   
**shows**  $\text{insert-nth } j \ x \ xs ! i = xs ! (i - 1)$   
<proof>

**lemma** *nth-insert-nth*:  
**assumes**  $i \leq \text{length } xs \ j \leq \text{length } xs$   
**shows**  $\text{insert-nth } j \ x \ xs ! i = (\text{if } i = j \ \text{then } x \ \text{else if } i < j \ \text{then } xs ! i \ \text{else } xs ! (i - 1))$   
<proof>

**lemma** *insert-nth-inverse*:  
**assumes**  $j \leq \text{length } xs \ j' \leq \text{length } xs'$   
**assumes**  $x \notin \text{set } xs \ x \notin \text{set } xs'$   
**assumes**  $\text{insert-nth } j \ x \ xs = \text{insert-nth } j' \ x \ xs'$   
**shows**  $j = j'$   
<proof>



Insert several elements at given (ascending) positions

**lemma** *length-fold-insert-nth*:

$length (fold (\lambda(p, b). insert-nth p b) pxs xs) = length xs + length pxs$   
 ⟨proof⟩

**lemma** *invar-fold-insert-nth*:

$\llbracket \forall x \in set\ pxs. p < fst\ x; p < length\ xs; xs\ !\ p = b \rrbracket \implies$   
 $fold (\lambda(x, y). insert-nth x y) pxs xs\ !\ p = b$   
 ⟨proof⟩

**lemma** *nth-fold-insert-nth*:

$\llbracket sorted\ (map\ fst\ pxs); distinct\ (map\ fst\ pxs); \forall (p, b) \in set\ pxs. p < length\ xs +$   
 $length\ pxs;$   
 $i < length\ pxs; pxs\ !\ i = (p, b) \rrbracket \implies$   
 $fold (\lambda(p, b). insert-nth p b) pxs xs\ !\ p = b$   
 ⟨proof⟩

## 1.4 Remove at position

**fun** *remove-nth* ::  $nat \Rightarrow 'a\ list \Rightarrow 'a\ list$

**where**

$remove-nth\ i\ [] = []$   
 $| remove-nth\ 0\ (x \# xs) = xs$   
 $| remove-nth\ (Suc\ i)\ (x \# xs) = x \# remove-nth\ i\ xs$

**lemma** *remove-nth-take-drop*:

$remove-nth\ i\ xs = take\ i\ xs @ drop\ (Suc\ i)\ xs$   
 ⟨proof⟩

**lemma** *remove-nth-insert-nth*:

**assumes**  $i \leq length\ xs$   
**shows**  $remove-nth\ i\ (insert-nth\ i\ x\ xs) = xs$   
 ⟨proof⟩

**lemma** *insert-nth-remove-nth*:

**assumes**  $i < length\ xs$   
**shows**  $insert-nth\ i\ (xs\ !\ i)\ (remove-nth\ i\ xs) = xs$   
 ⟨proof⟩

**lemma** *length-remove-nth*:

**assumes**  $i < length\ xs$   
**shows**  $length\ (remove-nth\ i\ xs) = length\ xs - 1$   
 ⟨proof⟩

**lemma** *set-remove-nth-subset*:

$set\ (remove-nth\ j\ xs) \subseteq set\ xs$   
 ⟨proof⟩

**lemma** *set-remove-nth*:

**assumes** *distinct xs j < length xs*  
**shows**  $\text{set } (\text{remove-nth } j \text{ } xs) = \text{set } xs - \{xs ! j\}$   
<proof>

**lemma** *distinct-remove-nth:*  
**assumes** *distinct xs*  
**shows** *distinct (remove-nth i xs)*  
<proof>

**end**