

List Index

Tobias Nipkow

September 13, 2023

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 $xs = (\lambda a. \text{find-index } (\lambda x. x=a) xs)$

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

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

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

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

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

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

lemma *index-Nil[simp]: index [] $a = 0$*
<proof>

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

lemma *index-append: index ($xs @ ys$) $x =$
 (if $x : \text{set } xs$ then index $xs x$ else 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:*
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:*
size $xs = n \implies (n = \text{find-index } P xs) = (\forall x \in \text{set } xs. \sim P x)$
<proof>

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

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

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

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

lemma *find-index-less-size-conv*:

$(\text{find-index } P \text{ } xs < \text{size } xs) = (\exists x \in \text{set } xs. P \ x)$
 $\langle \text{proof} \rangle$

lemma *index-less-size-conv*:

$(\text{index } xs \ x < \text{size } xs) = (x \in \text{set } xs)$
 $\langle \text{proof} \rangle$

lemma *last-index-less-size-conv*:

$(\text{last-index } xs \ x < \text{size } xs) = (x : \text{set } xs)$
 $\langle \text{proof} \rangle$

lemma *index-less[simp]*:

$x : \text{set } xs \implies \text{size } xs \leq n \implies \text{index } xs \ x < n$
 $\langle \text{proof} \rangle$

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

$x : \text{set } xs \implies \text{size } xs \leq n \implies \text{last-index } xs \ x < n$
 $\langle \text{proof} \rangle$

lemma *last-index-Cons*: $\text{last-index } (x \# xs) \ y =$

$(\text{if } x=y \text{ then}$
 $\quad \text{if } x \in \text{set } xs \text{ then last-index } xs \ y + 1 \text{ else } 0$
 $\quad \text{else last-index } xs \ y + 1)$
 $\langle \text{proof} \rangle$

lemma *last-index-append*: $\text{last-index } (xs @ ys) \ x =$

$(\text{if } x : \text{set } ys \text{ then size } xs + \text{last-index } ys \ x$
 $\quad \text{else if } x : \text{set } xs \text{ then last-index } xs \ x \text{ else size } xs + \text{size } ys)$
 $\langle \text{proof} \rangle$

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

$\text{last-index } (xs @ [x]) \ y =$
 $(\text{if } x=y \text{ then size } xs$
 $\quad \text{else if } y : \text{set } xs \text{ then last-index } xs \ y \text{ else size } xs + 1)$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\text{index } (\text{rev } xs) \ x = \text{length } xs - \text{index } xs \ x - 1$

$\langle proof \rangle$

lemma *index-nth-id*:

$\llbracket distinct\ xs;\ n < length\ xs \rrbracket \implies index\ xs\ (xs!\ n) = n$
 $\langle proof \rangle$

lemma *index-upt[simp]*: $m \leq i \implies i < n \implies index\ [m..<n]\ i = i - m$
 $\langle proof \rangle$

lemma *index-eq-index-conv[simp]*: $x \in set\ xs \vee y \in set\ xs \implies$
 $(index\ xs\ x = index\ xs\ y) = (x = y)$
 $\langle proof \rangle$

lemma *last-index-eq-index-conv[simp]*: $x \in set\ xs \vee y \in set\ xs \implies$
 $(last-index\ xs\ x = last-index\ xs\ y) = (x = y)$
 $\langle proof \rangle$

lemma *inj-on-index*: $inj-on\ (index\ xs)\ (set\ xs)$
 $\langle proof \rangle$

lemma *inj-on-index2*: $I \subseteq set\ xs \implies inj-on\ (index\ xs)\ I$
 $\langle proof \rangle$

lemma *inj-on-last-index*: $inj-on\ (last-index\ xs)\ (set\ xs)$
 $\langle proof \rangle$

lemma *find-index-conv-takeWhile*:
 $find-index\ P\ xs = size(takeWhile\ (Not\ o\ P)\ xs)$
 $\langle proof \rangle$

lemma *index-conv-takeWhile*: $index\ xs\ x = size(takeWhile\ (\lambda y. x \neq y)\ xs)$
 $\langle proof \rangle$

lemma *find-index-first*: $i < find-index\ P\ xs \implies \neg P\ (xs!i)$
 $\langle proof \rangle$

lemma *index-first*: $i < index\ xs\ x \implies x \neq xs!i$
 $\langle proof \rangle$

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

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

$\langle proof \rangle$

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

$\langle proof \rangle$

lemma *index-eq-iff*:

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

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

$\langle proof \rangle$

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

$\langle proof \rangle$

lemma *last-index-drop*:

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

$\langle proof \rangle$

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

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

$\langle proof \rangle$

lemma *index-take-if-index*:

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

$\langle proof \rangle$

lemma *index-take-if-set*:

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

$\langle proof \rangle$

lemma *index-last[simp]*:

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

$\langle proof \rangle$

lemma *index-update-if-diff2*:

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

$\langle proof \rangle$

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

$\langle proof \rangle$

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

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

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

$\langle proof \rangle$

lemma *bij-betw-index*:

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

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

lemma *index-map-inj-on*:

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

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

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]*: $length\ (map\ index'\ n\ f\ xs) = length\ xs$
 $\langle proof \rangle$

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

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

lemmas $map\ index = map\ index'\ map\ zip[of\ 0, simplified]$

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

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

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

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

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

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

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

lemma *map-index-cong*:
 $\forall p < \text{length } xs. f p (xs [p]) = g p (xs [p]) \implies \text{map-index } f xs = \text{map-index } g xs$
 $\langle \text{proof} \rangle$

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

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

lemma *map-index-congL*:
 $\forall p < \text{length } xs. f p (xs [p]) = xs [p] \implies \text{map-index } f xs = xs$
 $\langle \text{proof} \rangle$

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

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

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

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

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

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

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

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

1.3 Insert at position

primrec *insert-nth* :: *nat* \Rightarrow '*a* \Rightarrow '*a list* \Rightarrow '*a list* **where**

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

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

lemma *insert-nth-take-drop*[simp]: *insert-nth n x xs* = *take n xs @ [x] @ drop n xs*
 \langle proof \rangle

lemma *length-insert-nth*: *length (insert-nth n x xs)* = *Suc (length xs)*
 \langle proof \rangle

lemma *set-insert-nth*:
set (insert-nth i x xs) = *insert x (set xs)*
 \langle proof \rangle

lemma *distinct-insert-nth*:
assumes *distinct xs*
assumes *x* \notin *set xs*
shows *distinct (insert-nth i x xs)*
 \langle proof \rangle

lemma *nth-insert-nth-front*:
assumes *i* < *j* *j* \leq *length xs*
shows *insert-nth j x xs ! i* = *xs ! i*
 \langle proof \rangle

lemma *nth-insert-nth-index-eq*:
assumes *i* \leq *length xs*
shows *insert-nth i x xs ! i* = *x*
 \langle proof \rangle

lemma *nth-insert-nth-back*:
assumes *j* < *i* *i* \leq *length xs*
shows *insert-nth j x xs ! i* = *xs ! (i - 1)*
 \langle proof \rangle

lemma *nth-insert-nth*:
assumes *i* \leq *length xs* *j* \leq *length xs*
shows *insert-nth j x xs ! i* = (if *i* = *j* then *x* else if *i* < *j* then *xs ! i* else *xs ! (i - 1)*)
 \langle proof \rangle

lemma *insert-nth-inverse*:
assumes *j* \leq *length xs* *j'* \leq *length xs'*
assumes *x* \notin *set xs* *x* \notin *set xs'*
assumes *insert-nth j x xs* = *insert-nth j' x xs'*
shows *j* = *j'*
 \langle proof \rangle

Insert several elements at given (ascending) positions

lemma *length-fold-insert-nth*:

$\text{length } (\text{fold } (\lambda(p, b). \text{insert-nth } p \ b) \ \text{pxs } xs) = \text{length } xs + \text{length } \text{pxs}$
 $\langle \text{proof} \rangle$

lemma *invar-fold-insert-nth*:

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

lemma *nth-fold-insert-nth*:

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

1.4 Remove at position

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

where

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

lemma *remove-nth-take-drop*:

$\text{remove-nth } i \ xs = \text{take } i \ xs @ \text{drop } (\text{Suc } i) \ xs$
 $\langle \text{proof} \rangle$

lemma *remove-nth-insert-nth*:

assumes $i \leq \text{length } xs$
shows $\text{remove-nth } i \ (\text{insert-nth } i \ x \ xs) = xs$
 $\langle \text{proof} \rangle$

lemma *insert-nth-remove-nth*:

assumes $i < \text{length } xs$
shows $\text{insert-nth } i \ (xs ! i) \ (\text{remove-nth } i \ xs) = xs$
 $\langle \text{proof} \rangle$

lemma *length-remove-nth*:

assumes $i < \text{length } xs$
shows $\text{length } (\text{remove-nth } i \ xs) = \text{length } xs - 1$
 $\langle \text{proof} \rangle$

lemma *set-remove-nth-subset*:

$\text{set } (\text{remove-nth } j \ xs) \subseteq \text{set } xs$
 $\langle \text{proof} \rangle$

lemma *set-remove-nth*:

```

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

lemma distinct-remove-nth:
  assumes distinct xs
  shows distinct (remove-nth i xs)
 $\langle proof \rangle$ 

end

```