List Index

Tobias Nipkow

March 17, 2025

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 = (λ*a*. *find-index* (λ*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*: *find-index P (xs @ ys) =*
(if ∃ x∈set xs. P x then find-index P xs else size xs + find-index P ys)
⟨*proof*⟩

lemma *find-index-le-size*: *find-index P xs <= size xs*
⟨*proof*⟩

lemma *index-le-size*: *index xs x <= size xs*
⟨*proof*⟩

lemma *last-index-le-size*: *last-index xs x <= size xs*
⟨*proof*⟩

lemma *index-Nil*[*simp*]: *index [] a = 0*
⟨*proof*⟩

lemma *index-Cons*[*simp*]: *index (x#xs) a = (if x=a then 0 else index xs a + 1)*
⟨*proof*⟩

lemma *index-append*: *index (xs @ ys) x =*
(if x : set xs then index xs x else size xs + index ys x)
⟨*proof*⟩

lemma *index-conv-size-if-notin*[*simp*]: *x ∉ set xs ⟹ index xs x = size xs*
⟨*proof*⟩

lemma *find-index-eq-size-conv*:
size xs = n ⟹ (find-index P xs = n) = (∀ x ∈ set xs. ~ P x)
⟨*proof*⟩

lemma *size-eq-find-index-conv*:
size xs = n ⟹ (n = find-index P xs) = (∀ x ∈ set xs. ~ P x)
⟨*proof*⟩

lemma *index-size-conv*: *size xs = n ⟹ (index xs x = n) = (x ∉ set xs)*
⟨*proof*⟩

lemma *size-index-conv*: *size xs = n ⟹ (n = index xs x) = (x ∉ set xs)*
⟨*proof*⟩

lemma *last-index-size-conv*:
assumes *size xs = n*
shows *(last-index xs x = n) = (x ∉ set xs)*
⟨*proof*⟩

lemma *size-last-index-conv*:

$size\ xs = n \Longrightarrow (n = last\text{-}index\ xs\ x) = (x \notin set\ xs)$
⟨*proof*⟩

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

lemma *index-less-size-conv*:
$(index\ xs\ x < size\ xs) = (x \in set\ xs)$
⟨*proof*⟩

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

lemma *index-less*[*simp*]:
$x : set\ xs \Longrightarrow size\ xs <= n \Longrightarrow index\ xs\ x < n$
⟨*proof*⟩

lemma *last-index-less*[*simp*]:
$x : set\ xs \Longrightarrow size\ xs <= n \Longrightarrow last\text{-}index\ xs\ x < n$
⟨*proof*⟩

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

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

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

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

lemma *nth-index*[*simp*]: $x \in set\ xs \Longrightarrow xs\ !\ index\ xs\ x = x$
⟨*proof*⟩

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

lemma *index-rev*: $[\![$ *distinct xs*; $x \in$ *set xs* $]\!] \Longrightarrow$
index (*rev xs*) $x =$ *length xs* $-$ *index xs x* $-$ *1*
⟨*proof*⟩

lemma *index-nth-id*:
$[\![$ *distinct xs*; $n <$ *length xs* $]\!] \Longrightarrow$ *index xs* (*xs* ! n) $= n$
⟨*proof*⟩

lemma *index-upt*[*simp*]: $m \leq i \Longrightarrow i < n \Longrightarrow$ *index* $[m..<n]$ $i = i-m$
⟨*proof*⟩

lemma *index-eq-index-conv*[*simp*]: $x \in$ *set xs* $\vee$ $y \in$ *set xs* $\Longrightarrow$
(*index xs x* $=$ *index xs y*) $= (x = y)$
⟨*proof*⟩

lemma *last-index-eq-index-conv*[*simp*]: $x \in$ *set xs* $\vee$ $y \in$ *set xs* $\Longrightarrow$
(*last-index xs x* $=$ *last-index xs y*) $= (x = y)$
⟨*proof*⟩

lemma *inj-on-index*: *inj-on* (*index xs*) (*set xs*)
⟨*proof*⟩

lemma *inj-on-index2*: $I \subseteq$ *set xs* $\Longrightarrow$ *inj-on* (*index xs*) I
⟨*proof*⟩

lemma *inj-on-last-index*: *inj-on* (*last-index xs*) (*set xs*)
⟨*proof*⟩

lemma *find-index-conv-takeWhile*:
find-index P xs $=$ *size*(*takeWhile* (*Not o P*) *xs*)
⟨*proof*⟩

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

lemma *find-index-first*: $i <$ *find-index P xs* $\Longrightarrow \neg P$ (*xs*!i)
⟨*proof*⟩

lemma *index-first*: $i <$*index xs x* $\Longrightarrow x \neq$*xs*!i
⟨*proof*⟩

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

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

lemma *index-eqI*:
$\quad$ **assumes** $i \le$ *length xs*
$\quad$ **assumes** $\forall j<i.$ *xs*!*j* $\neq x$
$\quad$ **assumes** $i<$*length xs* $\Longrightarrow$ *xs*!*i* $= x$
$\quad$ **shows** *index xs x* $= i$
$\quad$ ⟨*proof*⟩

lemma *index-eq-iff*:
$\quad$ *index xs x* $= i$
$\quad$ $\longleftrightarrow$ ($i \le$*length xs* $\wedge$ ($\forall j<i.$ *xs*!*j* $\neq x$) $\wedge$ ($i<$*length xs* $\longrightarrow$ *xs*!*i* $= x$))
$\quad$ ⟨*proof*⟩

lemma *index-take*: *index xs x* $>=$ *i* $\Longrightarrow$ $x \notin$ *set*(*take i xs*)
$\quad$ ⟨*proof*⟩

lemma *last-index-drop*:
$\quad$ *last-index xs x* $<$ *i* $\Longrightarrow$ $x \notin$ *set*(*drop i xs*)
$\quad$ ⟨*proof*⟩

lemma *set-take-if-index*: **assumes** *index xs x* $<$ *i* **and** $i \le$ *length xs*
$\quad$ **shows** $x \in$ *set* (*take i xs*)
⟨*proof*⟩

lemma *index-take-if-index*:
$\quad$ **assumes** *index xs x* $\le n$ **shows** *index* (*take n xs*) *x* $=$ *index xs x*
⟨*proof*⟩

lemma *index-take-if-set*:
$\quad$ *x* : *set*(*take n xs*) $\Longrightarrow$ *index* (*take n xs*) *x* $=$ *index xs x*
$\quad$ ⟨*proof*⟩

lemma *index-last*[*simp*]:
$\quad$ *xs* $\neq$ [] $\Longrightarrow$ *distinct xs* $\Longrightarrow$ *index xs* (*last xs*) $=$ *length xs* $-$ *1*
$\quad$ ⟨*proof*⟩

lemma *index-update-if-diff2*:
$\quad$ $n <$ *length xs* $\Longrightarrow$ $x \neq$ *xs*!*n* $\Longrightarrow$ $x \neq y$ $\Longrightarrow$ *index* (*xs*[*n* := *y*]) *x* $=$ *index xs x*
$\quad$ ⟨*proof*⟩

lemma *set-drop-if-index*: *distinct xs* $\Longrightarrow$ *index xs x* $<$ *i* $\Longrightarrow$ $x \notin$ *set*(*drop i xs*)
$\quad$ ⟨*proof*⟩

lemma *index-swap-if-distinct*: **assumes** *distinct xs* $i <$ *size xs* $j <$ *size xs*
$\quad$ **shows** *index* (*xs*[*i* := *xs*!*j*, *j* := *xs*!*i*]) *x* $=$

$(if\ x = xs!i\ then\ j\ else\ if\ x = xs!j\ then\ i\ else\ index\ xs\ x)$
⟨*proof*⟩

lemma *bij-betw-index*:
$distinct\ xs \Longrightarrow X = set\ xs \Longrightarrow l = size\ xs \Longrightarrow bij\text{-}betw\ (index\ xs)\ X\ \{0..<l\}$
⟨*proof*⟩

lemma *index-image*: $distinct\ xs \Longrightarrow set\ xs = X \Longrightarrow index\ xs\ `\ X = \{0..<size\ xs\}$
⟨*proof*⟩

lemma *index-map-inj-on*:
$[\![\ inj\text{-}on\ f\ S;\ y \in S;\ set\ xs \subseteq S\ ]\!] \Longrightarrow index\ (map\ f\ xs)\ (f\ y) = index\ xs\ y$
⟨*proof*⟩

lemma *index-map-inj*: $inj\ f \Longrightarrow index\ (map\ f\ xs)\ (f\ y) = index\ xs\ y$
⟨*proof*⟩

1.2 Map with index

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

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

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

abbreviation $map\text{-}index \equiv map\text{-}index'\ 0$

lemmas $map\text{-}index = map\text{-}index'\text{-}map\text{-}zip[of\ 0,\ simplified]$

lemma *take-map-index*: $take\ p\ (map\text{-}index\ f\ xs) = map\text{-}index\ f\ (take\ p\ xs)$
⟨*proof*⟩

lemma *drop-map-index*: $drop\ p\ (map\text{-}index\ f\ xs) = map\text{-}index'\ p\ f\ (drop\ p\ xs)$
⟨*proof*⟩

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

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

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

$i) = x)$
⟨*proof*⟩

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

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

lemma *map-index-cong*:
assumes $length\ xs = length\ ys\ \bigwedge i.\ i < length\ xs \Longrightarrow f\ i\ (xs\ !\ i) = g\ i\ (ys\ !\ i)$
shows $map\text{-}index\ f\ xs = map\text{-}index\ g\ ys$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

1.3 Insert at position

primrec *insert-nth* :: *nat* ⇒ *'a* ⇒ *'a list* ⇒ *'a list* **where**
insert-nth 0 x xs = *x* # *xs*
| *insert-nth* (*Suc n*) *x xs* = (*case xs of* [] ⇒ [*x*] | *y* # *ys* ⇒ *y* # *insert-nth n x ys*)

lemma *insert-nth-take-drop*[*simp*]: *insert-nth n x xs* = *take n xs* @ [*x*] @ *drop n xs*
⟨*proof*⟩

lemma *length-insert-nth*: *length* (*insert-nth n x xs*) = *Suc* (*length xs*)
⟨*proof*⟩

lemma *set-insert-nth*:
set (*insert-nth i x xs*) = *insert x* (*set xs*)
⟨*proof*⟩

lemma *distinct-insert-nth*:
assumes *distinct xs*
assumes $x \notin$ *set xs*
shows *distinct* (*insert-nth i x xs*)
⟨*proof*⟩

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

lemma *nth-insert-nth-index-eq*:
assumes $i \leq$ *length xs*
shows *insert-nth i x xs* ! *i* = *x*
⟨*proof*⟩

lemma *nth-insert-nth-back*:
assumes $j < i$ $i \leq$ *length xs*
shows *insert-nth j x xs* ! *i* = *xs* ! (*i* − *1*)
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *insert-nth-inverse*:
assumes $j \leq$ *length xs* $j' \leq$ *length xs'*

assumes $x \notin set\ xs\ x \notin set\ xs'$
assumes $insert\text{-}nth\ j\ x\ xs = insert\text{-}nth\ j'\ x\ xs'$
shows $j = j'$
⟨*proof*⟩

Insert several elements at given (ascending) positions

lemma *length-fold-insert-nth*:
$length\ (fold\ (\lambda(p,\ b).\ insert\text{-}nth\ p\ b)\ pxs\ xs) = length\ xs + length\ pxs$
⟨*proof*⟩

lemma *invar-fold-insert-nth*:
$[\![\forall x \in set\ pxs.\ p < fst\ x;\ p < length\ xs;\ xs\ !\ p = b]\!] \Longrightarrow$
$fold\ (\lambda(x,\ y).\ insert\text{-}nth\ x\ y)\ pxs\ xs\ !\ p = b$
⟨*proof*⟩

lemma *nth-fold-insert-nth*:
$[\![sorted\ (map\ fst\ pxs);\ distinct\ (map\ fst\ pxs);\ \forall (p,\ b) \in set\ pxs.\ p < length\ xs +$
$length\ pxs;$
$i < length\ pxs;\ pxs\ !\ i = (p,\ b)]\!] \Longrightarrow$
$fold\ (\lambda(p,\ b).\ insert\text{-}nth\ p\ b)\ pxs\ xs\ !\ p = b$
⟨*proof*⟩

1.4 Remove at position

fun $remove\text{-}nth :: nat \Rightarrow 'a\ list \Rightarrow 'a\ list$
where
$remove\text{-}nth\ i\ [] = []$
$|\ remove\text{-}nth\ 0\ (x\ \#\ xs) = xs$
$|\ remove\text{-}nth\ (Suc\ i)\ (x\ \#\ xs) = x\ \#\ remove\text{-}nth\ i\ xs$

lemma *remove-nth-take-drop*:
$remove\text{-}nth\ i\ xs = take\ i\ xs\ @\ drop\ (Suc\ i)\ xs$
⟨*proof*⟩

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

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

lemma *length-remove-nth*:
assumes $i < length\ xs$
shows $length\ (remove\text{-}nth\ i\ xs) = length\ xs - 1$
⟨*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 *set* (*remove-nth* *j* *xs*) = *set* *xs* − {*xs* ! *j*}
⟨*proof*⟩

lemma *distinct-remove-nth*:
assumes *distinct* *xs*
shows *distinct* (*remove-nth* *i* *xs*)
⟨*proof*⟩

1.5 Additional lemmas contributed by Manuel Eberl

lemma *map-index-idI*: ($\bigwedge$*i*. *f* *i* (*xs* ! *i*) = *xs* ! *i*) $\Longrightarrow$ *map-index* *f* *xs* = *xs*
⟨*proof*⟩

lemma *map-index-transfer* [*transfer-rule*]:
rel-fun (*rel-fun* (=) (*rel-fun* *R1* *R2*)) (*rel-fun* (*list-all2* *R1*) (*list-all2* *R2*))
map-index *map-index*
⟨*proof*⟩

lemma *map-index-Cons*: *map-index* *f* (*x* # *xs*) = *f* *0* *x* # *map-index* (λ*i* *x*. *f* (*Suc* *i*) *x*) *xs*
⟨*proof*⟩

lemma *map-index-rev*: *map-index* *f* (*rev* *xs*) = *rev* (*map-index* (λ*i*. *f* (*length* *xs* − *i* − *1*)) *xs*)
⟨*proof*⟩

lemma *map-conv-map-index*: *map* *f* *xs* = *map-index* (λ*i* *x*. *f* *x*) *xs*
⟨*proof*⟩

lemma *map-index-map-index*: *map-index* *f* (*map-index* *g* *xs*) = *map-index* (λ*i* *x*. *f* *i* (*g* *i* *x*)) *xs*
⟨*proof*⟩

lemma *map-index-replicate* [*simp*]: *map-index* *f* (*replicate* *n* *x*) = *map* (λ*i*. *f* *i* *x*) [*0*..<*n*]
⟨*proof*⟩

lemma *zip-map-index*:
zip (*map-index* *f* *xs*) (*map-index* *g* *ys*) = *map-index* (λ*i*. *map-prod* (*f* *i*) (*g* *i*)) (*zip* *xs* *ys*)
⟨*proof*⟩

lemma *map-index-conv-fold*:
map-index *f* *xs* = *rev* (*snd* (*fold* (λ*x* (*i*,*ys*). (*i*+*1*, *f* *i* *x* # *ys*)) *xs* (*0*, [])))

⟨*proof*⟩

lemma *map-index-code-conv-foldr*:

$map\text{-}index\ f\ xs = snd\ (foldr\ (\lambda x\ (i,ys).\ (i-1,\ f\ i\ x\ \#\ ys))\ xs\ (length\ xs\ -\ 1,\ []))$

⟨*proof*⟩

end