List Index

Tobias Nipkow

December 14, 2021

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* ⇒ *bool*) ⇒ '*a list* ⇒ *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* ⇒ '*a* ⇒ *nat* **where**
index xs = (λ*a*. *find-index* (λ*x*. *x*=*a*) *xs*)

definition *last-index* :: '*a list* ⇒ '*a* ⇒ *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*:
size xs = n ⟹ (last-index xs x = n) = (x ∉ set xs)
⟨*proof*⟩

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

lemma *find-index-less-size-conv*:
(*find-index P xs* < *size xs*) = (∃ *x* ∈ *set xs*. *P x*)
⟨*proof*⟩

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

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

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

lemma *last-index-less*[*simp*]:
x : *set xs* ⟹ *size xs* <= *n* ⟹ *last-index xs x* < *n*
⟨*proof*⟩

lemma *last-index-Cons*: *last-index* (*x*#*xs*) *y* =
(*if x*=*y then*
if x ∈ *set xs then last-index xs y* + *1 else 0*
else last-index xs y + *1*)
⟨*proof*⟩

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

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

lemma *nth-find-index*: *find-index P xs* < *size xs* ⟹ *P*(*xs* ! *find-index P xs*)
⟨*proof*⟩

lemma *nth-index*[*simp*]: *x* ∈ *set xs* ⟹ *xs* ! *index xs x* = *x*
⟨*proof*⟩

lemma *nth-last-index*[*simp*]: *x* ∈ *set xs* ⟹ *xs* ! *last-index xs x* = *x*
⟨*proof*⟩

lemma *index-rev*: ⟦ *distinct xs*; *x* ∈ *set xs* ⟧ ⟹
index (*rev xs*) *x* = *length xs* − *index xs x* − *1*

⟨*proof*⟩

lemma *index-nth-id*:
  ⟦ *distinct xs*; $n < length\ xs$ ⟧ ⟹ $index\ xs\ (xs\ !\ n) = n$
⟨*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*]: $x \in set\ xs \vee y \in set\ xs \Longrightarrow$
  $(last\text{-}index\ xs\ x = last\text{-}index\ xs\ y) = (x = y)$
⟨*proof*⟩

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

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

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

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

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

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

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

⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

lemma *index-swap-if-distinct*: **assumes** $distinct\ xs\ i < size\ xs\ j < size\ xs$
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-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-on f S*; *y* ∈ *S*; *set xs* ⊆ *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-index′* :: *nat* ⇒ (*nat* ⇒ *′a* ⇒ *′b*) ⇒ *′a list* ⇒ *′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* ≡ *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* (λ*n x*. *g* (*f n x*)) *xs*
⟨*proof*⟩

lemma *map-index-map*[*simp*]: *map-index f* (*map g xs*) = *map-index* (λ*n x*. *f n* (*g x*)) *xs*
⟨*proof*⟩

lemma *set-map-index*[*simp*]: *x* ∈ *set* (*map-index f xs*) = (∃ *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*:
$\forall\, p < length\ xs.\ f\ p\ (xs\ !\ p) = g\ p\ (xs\ !\ p) \Longrightarrow map\text{-}index\ f\ xs = map\text{-}index\ g\ xs$
⟨*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*]: $map\text{-}index\ f\ (a\ @\ b) = map\text{-}index\ f\ a\ @\ map\text{-}index'\ (length\ a)\ f\ b$
⟨*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* ∉ *set xs*
  shows *distinct* (*insert-nth i x xs*)
⟨*proof*⟩

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

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

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

lemma *nth-insert-nth*:
  assumes *i* ≤ *length xs j* ≤ *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* ≤ *length xs j'* ≤ *length xs'*
  assumes *x* ∉ *set xs x* ∉ *set xs'*
  assumes *insert-nth j x xs* = *insert-nth j' x xs'*
  shows *j* = *j'*
⟨*proof*⟩

Insert several elements at given (ascending) positions

lemma *length-fold-insert-nth*:
length (*fold* (λ(*p*, *b*). *insert-nth p b*) *pxs xs*) = *length xs* + *length pxs*
⟨*proof*⟩

lemma *invar-fold-insert-nth*:
⟦∀ *x*∈*set pxs*. *p* < *fst x*; *p* < *length xs*; *xs* ! *p* = *b*⟧ ⟹
fold (λ(*x*, *y*). *insert-nth x y*) *pxs xs* ! *p* = *b*
⟨*proof*⟩

lemma *nth-fold-insert-nth*:
⟦*sorted* (*map fst pxs*); *distinct* (*map fst pxs*); ∀ (*p*, *b*) ∈ *set pxs*. *p* < *length xs* + *length pxs*;
i < *length pxs*; *pxs* ! *i* = (*p*, *b*)⟧ ⟹
fold (λ(*p*, *b*). *insert-nth p b*) *pxs xs* ! *p* = *b*
⟨*proof*⟩

1.4 Remove at position

fun *remove-nth* :: *nat* ⇒ *'a list* ⇒ *'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* ≤ *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*) ⊆ *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*⟩

end