Abstract

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

theory List-Index imports Main begin

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

0.1 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 = (λa. find-index (λx. 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-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) = (ALL x : set xs. ~ P x)
⟨proof⟩

lemma size-eq-find-index-conv:
  size xs = n =⇒ (n = find-index P xs) = (ALL 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) = (EX x : set xs. P x)
⟨proof⟩

2
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-nth-id:
[ distinct xs; n < length xs ] ⇒ index xs (xs ! n) = n
⟨proof⟩

lemma index-upt[simp]: m ≤ i ⇒ i < n ⇒ index [m..<n] i = i - m
⟨proof⟩
lemma index-eq-index-conv[simp]: \( x \in set \, xs \lor y \in set \, xs \implies (index \, xs \, x = index \, xs \, y) = (x = y) \)

⟨proof⟩

lemma last-index-eq-index-conv[simp]: \( x \in set \, xs \lor y \in set \, xs \implies (last-index \, xs \, x = last-index \, xs \, y) = (x = y) \)

⟨proof⟩

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

lemma inj-on-index2: \( I \subseteq set \, xs \implies inj-on \, (index \, xs) \, I \)
⟨proof⟩

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

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

lemma index-take: \( index \, xs \, x \geq i \implies x \notin set (take \, i \, xs) \)
⟨proof⟩

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

lemma set-take-if-index: assumes \( index \, xs \, x < i \, \text{and} \, i \leq length \, xs \)
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: \( x : set (take \, n \, xs) \implies index \, (take \, n \, xs) \, x = index \, xs \, x \)
⟨proof⟩

lemma index-last[simp]: \( xs \neq [] \implies \text{distinct} \, xs \implies index \, (last \, xs) = length \, xs - 1 \)
⟨proof⟩

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

lemma set-drop-if-index: distinct \( xs \implies index \, xs \, x < i \implies x \notin set (drop \, i \, xs) \)
⟨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 ⇒ X = set xs ⇒ l = size xs ⇒ bij-betw (index xs) X {0..<l}
⟨proof⟩

lemma index-image: distinct xs ⇒ set xs = X ⇒ index xs ' X = {0..<size xs}
⟨proof⟩

0.2 Map with index

primrec map-index' :: nat ⇒ (nat ⇒ 'a ⇒ 'b) ⇒ 'a list ⇒ 'b list where
map-index' n [] = []
| map-index' n (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 (split 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 (λn x. g (f n
x)) xs
⟨proof⟩

lemma map-index-map[simp]: map-index f (map g xs) = map-index (λn 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∈set (map-index' n f xs)

5
\[ \exists i < \text{length } xs \cdot f (n + i) (xs!i) = x \]

\textit{lemma nth-map-index\[simp\]: } \(p < \text{length } xs \implies \text{map-index } f \ xs \mid p = f \ p \ (xs \mid p) \)

\textit{lemma map-index-cong: }
\(\forall p < \text{length } xs. \ f \ p \ (xs \mid p) = g \ p \ (xs \mid p) \implies \text{map-index } f \ xs = \text{map-index } g \ xs \)

\textit{lemma map-index-id: } \text{map-index} (\text{curry } \text{snd}) \ xs = xs

\textit{lemma map-index-no-index\[simp\]: } \text{map-index} (\lambda n \ x. f \ x) \ xs = \text{map } f \ xs

\textit{lemma map-index-congL: }
\(\forall p < \text{length } xs. \ f \ p \ (xs \mid p) = xs \mid p \implies \text{map-index } f \ xs = xs \)

\textit{lemma map-index\'\-is-NilD: } \text{map-index\'} \ n \ f \ xs = [] \implies xs = []

\textit{declare map-index\'\-is-NilD\[of \ 0, \ dest!\]}\]

\textit{lemma map-index\'\-is-ConsD: }
\text{map-index\'} \ n \ f \ zs = y \ # \ ys \implies \exists z \ zs. \ zs = z \ # \ zs \land f \ n \ z = y \land \text{map-index\'} (n + 1) \ f \ zs = ys

\textit{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

\textit{lemmas map-index\'\-eq-imp-length-eq = map-index\'\-eq-imp-length-eq\[of \ 0\]}\]

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

\textit{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

\textit{lemma map-index\-append\[simp\]: } \text{map-index} \ f \ (a @ b) = \text{map-index} \ f \ a @ \text{map-index} \ (\text{length } a) \ f \ b

\[6\]
0.3 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 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⟩

end