

Chapter 1

Library c03_lists

Capítulo 3 - Working with Structured Data (Lists)

From *LF* Require Export *c02_induction*.

Par Ordenado

Inductive *natprod* : Type :=
| *pair* (*x y* : nat) : *natprod*.

Check (*pair* 3 5).

Projeção x

Definition *fst* (*p* : *natprod*) : nat :=
match *p* with
| *pair* *x y* => *x*
end.

Projeção y

Definition *snd* (*p* : *natprod*) : nat :=
match *p* with
| *pair* *x y* => *y*
end.

Notation "(*x* , *y*)" := (*pair* *x y*).

Projeção x

Definition *fst'* (*p* : *natprod*) : nat :=
match *p* with
| (*x*, *y*) => *x*
end.

Projeção y

Definition *snd'* (*p* : *natprod*) : nat :=
match *p* with

| (x, y) \Rightarrow y
end.

Troca de Componentes

Definition *swap_pair* (p : natprod) : natprod :=
match p with
| (x, y) \Rightarrow (y, x)
end.

Sobrejetividade

Theorem *surjective_pairing'* : $\forall (x\ y : \text{nat}),$
(x, y) = (fst (x, y), snd (x, y)).

Theorem *surjective_pairing* : $\forall (p : \text{natprod}),$
p = (fst p, snd p).

Exercise: 1 star, standard (snd_fst_is_swap)

Theorem *snd_fst_is_swap* : $\forall (p : \text{natprod}),$
(snd p, fst p) = swap_pair p.

Exercise: 1 star, standard, optional (fst_swap_is_snd)

Theorem *fst_swap_is_snd* : $\forall (p : \text{natprod}),$
fst (swap_pair p) = snd p.

Lista de Naturais

Inductive *natlist* : Type :=
| nil : natlist
| cons (x : nat) (l : natlist) : natlist.

Notações

Notation "x :: l" := (cons x l) (at level 60, right associativity).

Notation "[]" := nil.

Notation "[x ; .. ; y]" := (cons x .. (cons y nil) ..).

Definições Equivalentes

Definition *mylist1* := cons 1 (cons 2 (cons 3 nil)).

Print *mylist1*.

Definition *mylist2* := 1 :: (2 :: (3 :: nil)).

Print *mylist2*.

Definition *mylist3* := 1 :: 2 :: 3 :: nil.

Print *mylist3*.

Definition *mylist4* := [1; 2; 3].

Print *mylist4*.

Função Repetir

```
Fixpoint repeat (x count : nat) : natlist :=  
  match count with  
  | 0 => nil  
  | S count' => x :: (repeat x count')  
end.
```

Função Comprimento

```
Fixpoint length (l : natlist) : nat :=  
  match l with  
  | nil => 0  
  | h :: t => S (length t)  
end.
```

Função Concatenar

```
Fixpoint app (l1 l2 : natlist) : natlist :=  
  match l1 with  
  | nil => l2  
  | h :: t => h :: (app t l2)  
end.
```

Notation "x ++ y" := (app x y) (right associativity, at level 60).

Example test_app1: [1; 2; 3] ++ [4; 5] = [1; 2; 3; 4; 5].

Example test_app2: nil ++ [4; 5] = [4; 5].

Example test_app3: [1; 2; 3] ++ nil = [1; 2; 3].

Função Cabeça

```
Definition hd (default : nat) (l : natlist) : nat :=  
  match l with  
  | nil => default  
  | h :: t => h  
end.
```

Example test_hd1: hd 0 [1; 2; 3] = 1.

Example test_hd2: hd 0 [] = 0.

Função Cauda

```
Definition tl (l : natlist) : natlist :=  
  match l with  
  | nil => nil  
  | h :: t => t  
end.
```

Example test_tl1: tl [1; 2; 3] = [2; 3].

Example *test_tl2*: $tl\ [1; 2; 3; 4] = [2; 3; 4]$.

Exercise: 2 stars, standard, recommended (list_funs)

```
Fixpoint nonzeros (l : natlist) : natlist :=
  match l with
  | nil ⇒ nil
  | O :: t ⇒ nonzeros t
  | h :: t ⇒ h :: (nonzeros t)
  end.
```

Example *test_nonzeros*: $nonzeros\ [0; 1; 0; 2; 3; 0; 0] = [1; 2; 3]$.

```
Fixpoint oddmembers (l : natlist) : natlist :=
  match l with
  | nil ⇒ nil
  | h :: t ⇒ match (oddb h) with
    | true ⇒ h :: (oddmembers t)
    | false ⇒ oddmembers t
  end
end.
```

Example *test_oddmembers*: $oddmembers\ [0; 1; 0; 2; 3; 0; 0] = [1; 3]$.

```
Fixpoint countoddmembers (l : natlist) : nat :=
  match l with
  | nil ⇒ 0
  | h :: t ⇒ if (oddb h) then 1 + (countoddmembers t) else countoddmembers t
  end.
```

Example *test_countoddmembers1*: $countoddmembers\ [1; 0; 3; 1; 4; 5] = 4$.

Example *test_countoddmembers2*: $countoddmembers\ [0; 2; 4] = 0$.

Example *test_countoddmembers3*: $countoddmembers\ nil = 0$.

Exercise: 3 stars, advanced (alternate)

```
Fixpoint alternate (l1 l2 : natlist) : natlist :=
  match l1, l2 with
  | nil, nil ⇒ nil
  | nil, yb ⇒ yb
  | xb, nil ⇒ xb
  | x :: xb, y :: yb ⇒ x :: y :: alternate xb yb
  end.
```

Example *test_alternate1*: $alternate\ [1; 2; 3]\ [4; 5; 6] = [1; 4; 2; 5; 3; 6]$.

Example *test_alternate2*: $alternate\ [1]\ [4; 5; 6] = [1; 4; 5; 6]$.

Example *test_alternate3*: $alternate\ [1; 2; 3]\ [4] = [1; 4; 2; 3]$.

Example *test_alternate4*: $alternate\ []\ [20; 30] = [20; 30]$.

Definition *bag* := *natlist*.

Exercise: 3 stars, standard, recommended (*bag_functions*)

```
Fixpoint count (x : nat) (b : bag) : nat :=
  match b with
  | nil => 0
  | h :: t => if eqb x h then S (count x t) else count x t
  end.
```

Example *test_count1*: *count* 1 [1; 2; 3; 1; 4; 1] = 3.

Example *test_count2*: *count* 6 [1; 2; 3; 1; 4; 1] = 0.

Definition *sum* : *bag* → *bag* → *bag* := *app*.

Example *test_sum1*: *count* 1 (*sum* [1; 2; 3] [1; 4; 1]) = 3.

Definition *add* (x : nat) (b : bag) : bag := x :: b.

Example *test_add1*: *count* 1 (*add* 1 [1; 4; 1]) = 3.

Example *test_add2*: *count* 5 (*add* 1 [1; 4; 1]) = 0.

```
Definition member (x : nat) (b : bag) : bool :=
  match (count x b) with
  | 0 => false
  | _ => true
  end.
```

Example *test_member1*: *member* 1 [1; 4; 1] = *true*.

Example *test_member2*: *member* 2 [1; 4; 1] = *false*.

Exercise: 3 stars, standard, optional (*bag_more_functions*)

```
Fixpoint remove_one (x : nat) (b : bag) : bag :=
  match b with
  | nil => nil
  | h :: t => if eqb x h then t else h :: remove_one x t
  end.
```

Example *test_remove_one1*: *count* 5 (*remove_one* 5 [2; 1; 5; 4; 1]) = 0.

Example *test_remove_one2*: *count* 5 (*remove_one* 5 [2; 1; 4; 1]) = 0.

Example *test_remove_one3*: *count* 4 (*remove_one* 5 [2; 1; 4; 5; 1; 4]) = 2.

Example *test_remove_one4*: *count* 5 (*remove_one* 5 [2; 1; 5; 4; 5; 1; 4]) = 1.

```
Fixpoint remove_all (x : nat) (b : bag) : bag :=
  match b with
  | nil => nil
  | h :: t => if eqb x h then remove_all x t else h :: remove_all x t
  end.
```

Example *test_remove_all1*: *count* 5 (*remove_all* 5 [2; 1; 5; 4; 1]) = 0.

Example *test_remove_all2*: $\text{count } 5 (\text{remove_all } 5 [2; 1; 4; 1]) = 0$.

Example *test_remove_all3*: $\text{count } 4 (\text{remove_all } 5 [2; 1; 4; 5; 1; 4]) = 2$.

Example *test_remove_all4*: $\text{count } 5 (\text{remove_all } 5 [2; 1; 5; 4; 5; 1; 4; 5; 1; 4]) = 0$.

```
Fixpoint subset (b1 : bag) (b2 : bag) : bool :=
  match b1 with
  | nil  $\Rightarrow$  true
  | h :: t  $\Rightarrow$  andb (member h b2) (subset t (remove_one h b2))
  end.
```

Example *test_subset1*: $\text{subset } [1; 2] [2; 1; 4; 1] = \text{true}$.

Example *test_subset2*: $\text{subset } [1; 2; 2] [2; 1; 4; 1] = \text{false}$.

Exercise: 2 stars, standard, recommended (bag_theorem)

Theorem *bag_theorem* : $\forall (b : \text{bag}), \forall (x : \text{nat}),$
 $S (\text{length } b) = \text{length } (\text{add } x \ b).$

Raciocínio sobre listas

Theorem *nil_app* : $\forall l : \text{natlist},$
 $[] ++ l = l.$

Theorem *tl_length_pred* : $\forall l : \text{natlist},$
 $\text{pred } (\text{length } l) = \text{length } (\text{tl } l).$

Theorem *app_assoc* : $\forall l1 \ l2 \ l3 : \text{natlist},$
 $(l1 ++ l2) ++ l3 = l1 ++ (l2 ++ l3).$

```
Fixpoint rev (l : natlist) : natlist :=
  match l with
  | nil  $\Rightarrow$  nil
  | h :: t  $\Rightarrow$  rev t ++ [h]
  end.
```

Example *test_rev1*: $\text{rev } [1; 2; 3] = [3; 2; 1].$

Example *test_rev2*: $\text{rev } \text{nil} = \text{nil}.$

Theorem *app_length* : $\forall l1 \ l2 : \text{natlist},$
 $\text{length } (l1 ++ l2) = (\text{length } l1) + (\text{length } l2).$

Theorem *rev_length* : $\forall l : \text{natlist},$
 $\text{length } (\text{rev } l) = \text{length } l.$

Exercise: 3 stars, standard (list_exercises)

Theorem *app_nil_r* : $\forall l : \text{natlist},$
 $l ++ [] = l.$

Theorem *rev_app_distr*: $\forall l1 \ l2 : \text{natlist},$
 $\text{rev } (l1 ++ l2) = \text{rev } l2 ++ \text{rev } l1.$

Theorem *rev_involutive* : $\forall l : \text{natlist},$

$rev (rev l) = l.$

Theorem *app_assoc4* : $\forall l1\ l2\ l3\ l4 : natlist,$
 $l1 ++ (l2 ++ (l3 ++ l4)) = ((l1 ++ l2) ++ l3) ++ l4.$

Lemma *nonzeros_app* : $\forall l1\ l2 : natlist,$
 $nonzeros (l1 ++ l2) = (nonzeros l1) ++ (nonzeros l2).$

Exercise: 2 stars, standard (eqblist)

Fixpoint *eqblist* (l1 l2 : natlist) : bool :=
 match l1, l2 with
 | [], [] \Rightarrow true
 | -, [] \Rightarrow false
 | [], - \Rightarrow false
 | h1 :: t1, h2 :: t2 \Rightarrow if eqb h1 h2 then eqblist t1 t2 else false
 end.

Example *test_eqblist1* : (eqblist nil nil = true).

Example *test_eqblist2* : eqblist [1; 2; 3] [1; 2; 3] = true.

Example *test_eqblist3* : eqblist [1; 2; 3] [1; 2; 4] = false.

Theorem *eqblist_refl* : $\forall l : natlist,$
 $true = eqblist\ l\ l.$

Exercise: 1 star, standard (count_member_nonzero)

Theorem *count_member_nonzero* : $\forall (b : bag),$
 $leb\ 1\ (count\ 1\ (1 :: b)) = true.$

Theorem *leb_x_Sx* : $\forall x,$
 $leb\ x\ (S\ x) = true.$

Exercise: 3 stars, advanced (remove_does_not_increase_count)

Theorem *remove_does_not_increase_count* : $\forall (b : bag),$
 $leb\ (count\ 0\ (remove_one\ 0\ b))\ (count\ 0\ b) = true.$

Exercise: 3 stars, standard, optional (bag_count_sum)

Falta fazer

Exercise: 4 stars, advanced (rev_injective)

Theorem *rev_injective* : $\forall (l1\ l2 : natlist),$
 $rev\ l1 = rev\ l2 \rightarrow l1 = l2.$

Opções

Fixpoint *xth_bad* (l : natlist) (x : nat) : nat :=
 match l with
 | nil \Rightarrow 42 arbitrário | a :: l' \Rightarrow match eqb x 0 with
 | true \Rightarrow a
 | false \Rightarrow xth_bad l' (pred x)

```

    end
  end.

Inductive natoption : Type :=
| Some (x : nat) : natoption
| None : natoption.

Fixpoint xth_error (l : natlist) (x : nat) : natoption :=
  match l with
  | nil ⇒ None
  | a :: l' ⇒ match eqb x 0 with
    | true ⇒ Some a
    | false ⇒ xth_error l' (pred x)
  end
end.

Example test_xth_error1 : xth_error [4; 5; 6; 7] 0 = Some 4.
Example test_xth_error2 : xth_error [4; 5; 6; 7] 3 = Some 7.
Example test_xth_error3 : xth_error [4; 5; 6; 7] 9 = None.

Fixpoint xth_error' (l : natlist) (x : nat) : natoption :=
  match l with
  | nil ⇒ None
  | a :: l' ⇒ if eqb x 0 then Some a else xth_error' l' (pred x)
  end.

Definition option_elim (x : nat) (o : natoption) : nat :=
  match o with
  | Some n' ⇒ n'
  | None ⇒ x
  end.

  Exercise: 2 stars (hd_error)

Definition hd_error (l : natlist) : natoption :=
  match l with
  | [] ⇒ None
  | h :: _ ⇒ Some h
  end.

Example test_hd_error1 : hd_error [] = None.
Example test_hd_error2 : hd_error [1] = Some 1.
Example test_hd_error3 : hd_error [5; 6] = Some 5.

  Exercise: 1 star, optional (option_elim_hd)

Theorem option_elim_hd : ∀ (l : natlist) (default : nat),
  hd default l = option_elim default (hd_error l).

```


Partial Maps

Inductive *id* : Type :=
| *Id* (*x* : nat) : *id*.

Definition *eqb_id* (*x1 x2* : *id*) :=
match *x1, x2* with
| *Id n1, Id n2* ⇒ *eqb n1 n2*
end.

Exercise: 1 star (*eqb_id_refl*)

Theorem *eqb_id_refl* : $\forall x, \text{true} = \text{eqb_id } x \ x$.

Inductive *partial_map* : Type :=
| *empty* : *partial_map*
| *record* (*i* : *id*) (*x* : nat) (*m* : *partial_map*) : *partial_map*.

Definition *update* (*m* : *partial_map*) (*i* : *id*) (*x* : nat) : *partial_map* :=
record i x m.

Fixpoint *find* (*i* : *id*) (*m* : *partial_map*) : *natoption* :=
match *m* with
| *empty* ⇒ *None*
| *record x y m'* ⇒ if *eqb_id i x* then *Some y* else *find i m'*
end.

Exercise: 1 star (*update_eq*)

Theorem *update_eq* : $\forall (m : \text{partial_map}) (i : \text{id}) (x : \text{nat}),$
find i (update m i x) = Some x.

Exercise: 1 star (update_neq) **Theorem** *update_neq* : $\forall (m : \text{partial_map}) (x \ y : \text{id})$
(*n* : nat),
eqb_id x y = false → *find x (update m y n) = find x m*.

Exercise: 2 stars (*baz_num_elts*)

Inductive *baz* : Type :=
| *Baz1* (*x* : *baz*) : *baz*
| *Baz2* (*y* : *baz*) (*b* : bool) : *baz*.

How *many* elements does the type *baz* have?

Falta fazer