Título do Relatório

April 21, 2023

Coloque aqui o seu nome

1 Algoritmo que retorna o maior elemento de uma lista de naturais.

```
Require Import Arith List Lia.

Fixpoint elt_max_aux l max :=
  match l with
  | nil ⇒ max
  | h::tl ⇒ if max <? h then elt_max_aux tl h else elt_max_aux tl max
  end.

Eval compute in (elt_max_aux (1::2::3::nil) 0).
Eval compute in (elt_max_aux (1::2::3::nil) 7).

Definition ge_all x l := ∀ y, In y l → y ≤ x.
Infix ”¿=*” := ge_all (at level 70, no associativity).

Definition le_all x l := ∀ y, In y l → x ≤ y.
Infix ”¡=*” := le_all (at level 70, no associativity).

Lemma elt_max_aux_large: ∀ l a, a ≤ elt_max_aux l a.
Proof.
  induction l.
  - intro a. simpl. lia.
  - intro a'. simpl. destruct (a' <? a) eqn:H.
    + apply Nat.le_trans with a.
      × apply Nat.ltb_lt in H. apply Nat.lt_le_incl; assumption.
      × apply IHl.
    + apply IHl.
Qed.

Lemma elt_max_aux_le: ∀ l a a', a ≤ a' → elt_max_aux l a ≤ elt_max_aux
l a'.
Proof.
  induction l.
  - intros a a' H. simpl. assumption.
```

```
  - intros a' a'' H. simpl. destruct (a' <? a) eqn:H1.
    + destruct (a'' <? a) eqn:H2.
      * lia.
      * apply IHl. apply Nat.ltb_ge. assumption.
    + destruct (a'' <? a) eqn:H2.
      * apply Nat.ltb_lt in H2. apply Nat.ltb_ge in H1. lia.
      * apply IHl. assumption.
Qed.

Lemma elt_max_aux_correct_1: ∀ l a, elt_max_aux l a >= * a::l.
Proof.
  induction l.
  - intro a. simpl. unfold ge_all. intros y H. inversion H.
    + subst. lia.
    + inversion H0.
  - intros a'. simpl. destruct (a' <? a) eqn:H.
    + unfold ge_all in *. intros y H'. inversion H'; subst.
      * apply Nat.le_trans with a.
        ** apply Nat.ltb_lt in H. apply Nat.lt_le_incl; assumption.
        ** apply elt_max_aux_large.
      * apply IHl; assumption.
    + unfold ge_all in *. intros y H'. inversion H'; subst.
      * apply elt_max_aux_large.
      * apply Nat.le_trans with (elt_max_aux l a).
        ** apply IHl; assumption.
        ** apply elt_max_aux_le. apply Nat.ltb_ge. assumption.
Qed.

Lemma elt_max_aux_head: ∀ l d, In (elt_max_aux (d::l) d) (d::l).
Proof.
  induction l.
  - intro d. simpl. destruct (d <? d).
    + left. reflexivity.
    + left. reflexivity.
  - intro d. simpl in *. assert (H := IHl). specialize (H d). destruct
(d <? d) eqn:Hd.
    + rewrite Nat.ltb_irrefl in Hd. inversion Hd.
    + destruct H.
      * destruct (d <? a) eqn:Ha.
```

** specialize (*IHl a*). rewrite *Nat.ltb_irrefl* in *IHl*. destruct *IHl*.

*** right. left. assumption.

*** right. right. assumption.

** left. assumption.

× destruct (d <? a) *eqn*:*Ha*.

** specialize (*IHl a*). rewrite *Nat.ltb_irrefl* in *IHl*. destruct *IHl*.

*** right. left. assumption.

*** right. right. assumption.

** right. right. assumption.

Qed.

Lemma *elt_max_aux_correct_2*: ∀ l d, *elt_max_aux* (d::l) d >= ∗ d::l.

Proof.

intros l d. simpl. rewrite *Nat.ltb_irrefl*. apply *elt_max_aux_correct_1*.

Qed.

Lemma *in_swap*: ∀ (l: *list nat*) x y z, *In* z (x::y::l) → *In* z (y::x::l).

Proof.

intros l x y z H. simpl in *. rewrite ← *or_assoc* in *. destruct H.

- left. apply *or_comm*. assumption.
- right. assumption.

Qed.

Lemma *elt_max_aux_in*: ∀ l x, *In* (*elt_max_aux* l x) (x::l).

Proof.

induction l.

- intro x. simpl. left; reflexivity.
- intro x. simpl *elt_max_aux*. destruct (x <? a) *eqn*: *Hlt*.

× specialize (*IHl a*). apply *in_cons*; assumption.

× specialize (*IHl x*). apply *in_swap*. apply *in_cons*. assumption.

Qed.

Lemma *ge_ge_all*: ∀ l x y, $x \geq y$ → y >= ∗ l → x >= ∗ l.

Proof.

induction l.

- intros x y *H1* *H2*. unfold *ge_all* in *. intros *y' H'*. inversion *H'*.
- intros x y *H1* *H2*. unfold *ge_all* in *. intros *y' H'*. apply *H2* in *H'*. unfold *ge* in *H1*. apply *Nat.le_trans* with y; assumption.

Qed.

Lemma *elt_max_aux_correto*: $\forall$ *l x*, *In x l* $\rightarrow$ *elt_max_aux l x* >= * *l* $\wedge$ *In* (*elt_max_aux l x*) *l*.

Proof.

induction *l*.

- intros *x H*. inversion *H*.
- intros *x H*. inversion *H*.
  + subst. split.
    × apply *elt_max_aux_correct_2*.
    × apply *elt_max_aux_head*.
  + clear *H*. split.
    × simpl. destruct (*x* <? *a*) *eqn*:*Hlt*.
      ** apply *elt_max_aux_correct_1*.
      ** replace (*elt_max_aux l x*) with (*elt_max_aux* (*a*::*l*) *x*).
        *** apply *ge_ge_all* with (*elt_max_aux* (*a*::*l*) *a*).
          **** apply *Nat.ltb_ge* in *Hlt*. unfold *ge* in *. apply *elt_max_aux_le*. assumption.
          **** apply *elt_max_aux_correct_2*.
        *** simpl. rewrite *Hlt*. reflexivity.
    × simpl *elt_max_aux*. destruct (*x* <? *a*) *eqn*:*Hlt*.
      ** apply *elt_max_aux_in*.
      ** apply *in_cons*. apply *IHl*. assumption.

Qed.

Agora podemos definir a função principal:

Definition *elt_max* (*l*: *list nat*) :=
  match *l* with
  | *nil* ⇒ *None*
  | *h*::*tl* ⇒ *Some* (*elt_max_aux tl h*)
  end.

Theorem *elt_max_correto*: $\forall$ *l k*, *elt_max l* = *Some k* $\rightarrow$ *k* >= * *l* $\wedge$ *In k l*.

Proof.

induction *l*.

- intros *k H*. + subst. split.
    × unfold *ge_all* in *. intros *y' H'*. inversion *H'*.
    × inversion *H*.
- intros *k H*. inversion *H*.
  + subst. split.
    × apply *elt_max_aux_correct_1*.

× `apply` *elt_max_aux_in*.

`Qed`.

Referências

[ARdM17] M. Ayala-Rincón and F.L.C. de Moura. *Applied Logic for Computer Scientists - computational deduction and formal proofs.* UTiCS, Springer, 2017.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Electrical Engineering and Computer Science Series. MIT press, third edition, 2009.