Elemento máximo de uma lista de naturais

Primeira atividade no assistente de provas Coq

April 12, 2023

Prof. Flávio L. C. de Moura

Prazo: 30 de abril de 2023 (5 pontos).

Neste trabalho formalizaremos a correção de um algoritmo que retorna o maior elemento de uma lista qualquer de números naturais. A construção do algoritmo será feita por partes, iniciando com a função auxiliar *elt_max_aux*, que foi estudada durante as aulas 02 e 03. A função principal que representa o algoritmo será definida no final deste arquivo com o nome *elt_max*. Na definição a seguir, a função *elt_max_aux l max* recebe uma lista *l* de números naturais, e o natural *max*. O papel do segundo argumento desta função é armazenar o maior elemento encontrado à medida em que a lista é percorrida:

```
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.
```

A palavra reservada `Fixpoint` diz para o Coq que a função a ser definida é recursiva. A construção é feita considerando os dois construtores possíveis para uma lista. Esta construção é conhecida com *pattern matching*. O construtor *nil* representa o caso em que a lista *l* é a lista vazia, e o construtor *h*::*tl* representa o caso em que a lista *l* é a lista que tem *h* como primeiro elemento (cabeça da lista), e cauda *tl* (o restante da lista). Assim, quando a lista *l* é vazia, a função *elt_max_aux l max* retorna o natural *max*. Se *l* tem a forma *h*::*tl* então verificamos se *max* é estritamente menor do que *h*, e em caso afirmativo, recursivamente continuamos a busca pelo maior elemento da cauda *tl* sabendo que o maior elemento encontrado até o momento é *h*. Caso contrário, isto é, se *max* é maior ou igual a *h* então recursivamente continuamos a busca na cauda *tl* sabendo que *max* ainda é o maior elemento encontrado até o momento. Podemos verificar o comportamento da função *elt_max_aux* em casos específicos. A seguir, pedimos ao Coq que avalie a função *elt_max_aux* com argumentos (1::2::3::*nil*) e 0:

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

O retorno dado pelo Coq é:

```
= 3
: nat
```

Ou seja, inicializando o segundo argumento com 0, obtemos 3 como resultado. Se o segundo argumento for inicializado com 7 então o retorno obtido é 7:

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

```
= 7
: nat
```

A seguir definimos as funções *ge_all* e *le_all* que expressam o fato de um natural ser, respectivamente, maior ou igual e menor ou igual a todos os elementos de uma lista:

`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`).

Os lemas a seguir foram comentados durante as aulas, e não serão detalhados aqui, com exceção do lema *elt_max_aux_large* a seguir. Explicaremos a prova feita em Coq usando linguagem natural para servir como exemplo. O lema *elt_max_aux_large* diz que $a \leq$ *elt_max_aux l a*, quaisquer que sejam a lista *l*, e o natural *a*:

`Lemma` *elt_max_aux_large*: ∀ *l a*, *a* ≤ *elt_max_aux l a*.

Prova. A prova é feita por indução na estrutura da lista *l*. Então temos dois casos a considerar:

1. Se *l* for a lista vazia então *elt_max_aux nil a* retorna *a*, e concluímos, uma vez que $a \leq a$.

2. Se *l* tem a forma *h* :: *tl* então precisamos provar que *a* ≤ *elt_max_aux h* :: *tl a*, qualquer que seja o natural *a*. De acordo com a definição da função *elt_max_aux*, precisamos comparar *a* com *h*, o que nos dá 2 subcasos:

   (a) Se $a < h$ então a chamada recursiva retorna *elt_max_aux tl h*, que por hipótese de indução é maior ou igual a *h*, ou seja, $h \leq$ *elt_max_aux tl h*. Logo $a \leq h \leq$ *elt_max_aux tl h* = *elt_max_aux h* :: *tl a*.

   (b) Se $h \leq a$ então a chamada recursiva retorna *elt_max_aux tl a*, e portanto $a \overset{h.i.}{\leq}$ *elt_max_aux tl a* = *elt_max_aux h* :: *tl a*. □

`Lemma` *elt_max_aux_le*: ∀ *l a a'*, *a* ≤ *a'* → *elt_max_aux l a* ≤ *elt_max_aux l a'*.

`Lemma` *elt_max_aux_correct_1*: ∀ *l a*, *elt_max_aux l a* ¿=* *a*::*l*.

`Lemma` *elt_max_aux_head*: ∀ *l d*, *In* (*elt_max_aux* (*d*::*l*) *d*) (*d*::*l*).

`Lemma` *elt_max_aux_correct_2*: ∀ *l d*, *elt_max_aux* (*d*::*l*) *d* ¿=* *d*::*l*.

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

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

`Lemma` *ge_ge_all*: ∀ *l x y*, *x* ≥ *y* → *y* ¿=* *l* → *x* ¿=* *l*.

`Lemma` *elt_max_aux_correto*: ∀ *l x*, *In x l* → *elt_max_aux l x* ¿=* *l* ∧ *In* (*elt_max_aux l x*) *l*.

A função *elt_max* definida a seguir, recebe uma lista qualquer como argumento, e retorna *None* quando a lista é vazia, e *Some* (*elt_max_aux tl h*) quando a lista tem a forma *h*::*tl*. O tipo *option* é utilizado para expressar fatos que envolvem a presença opcional de elementos de um dado tipo. Ou seja, esperamos que a função *elt_max l* retorne o maior elemento da lista *l*, mas quando *l* é a lista vazia, não existe maior elemento a ser retornado, e o construtor *None* é então utilizado. Quando a lista é não vazia, utilizamos a função auxiliar *elt_max_aux*. Nexte caso, o retorno da função é *Some* (*elt_max_aux tl h*), assumindo que a lista tem a forma $h :: tl$:

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

O objetivo deste trabalho é provar que a função *elt_max* é correta, ou seja, que retorna o maior elemento da lista dada como argumento. O teorema a seguir caracteriza esta noção de correção:

```
Theorem elt_max_correto: ∀ l k, elt_max l = Some k → k ¿=* l ∧ In k l.
```

Para provar o teorema *elt_max_correto* pode ser interessante enunciar (e provar!) resultados auxiliares, assim como fizemos para a função *elt_max_aux*.

Observações importantes

1. O prazo para finalização desta atividade é **30 de abril de 2023**.
2. Crie um repositório no GitHub a partir do link disponibilizado no SIGAA.
3. Esta atividade vale 5 pontos (dos 30 disponíveis para o projeto de acordo com o plano de ensino). Soluções parciais serão consideradas.
4. No prazo especificado, o repositório deverá conter os seguintes arquivos:
   (a) `paa_2023_1_elemento_maximo.v` com a prova completa da correção da função *elt_max*;
   (b) `paa-2023-1-elemento-maximo.pdf` contendo um relato do trabalho que foi feito. Faça um texto que destaque os aspectos importantes do desenvolvimento das provas, como por exemplo, as dificuldades encontradas e as soluções adotadas.