Aula 8 - Mais provas sobre listas de naturais

Rafael Castro - rafaelcgs10.github.io/coq

21/05/2018

Fatos simples sobre listas de naturais

- Simplificação pela reflexividade é suficiente para provar fatos simples sobre listas de naturais.
- No exemplo abaixo a *reflexivity* é capaz de simplificar pois a definição de *app* tem um caso do *pattern-matching* considera o caso do primeiro argumento ser uma lista vazia.

```
Theorem nil_app : forall l:natlist,
  [] ++ l = l.
Proof. reflexivity. Qed.
```

Análise de caso sobre listas de naturais

- Caso o valor da lista não seja conhecido (vazio ou não-vazio), então a análise de caso pode de útil.
- Note que os nomes fornecidos na análise de caso são *n* e *l'*. Note, também, que não é necessário fornecer nomes para caso a lista seja vazia.
- Isso funciou pois definimos *tl [] = []*.

```
Theorem tl_length_pred : forall l:natlist,
  pred (length l) = length (tl l).
Proof.
  intros l. destruct l as [| n l'].
  - (* l = nil *)
    reflexivity.
  - (* l = cons n l' *)
    reflexivity.
Qed.
```

Indução sobre listas de naturais

- Uma das maravilhas dos tipos definidos com *inductive* (tipos indutivos) é a possibilidade de fazer indução.
- Logo, pode-se fazer indução sobre listas assim como sobre números naturais!
- Lembre-se, cada definição num tipo indutivo define um conjunto de valores.
- Assim como um natural ou é zero ou é um sucessor aplicado a um natural, uma lista ou é uma lista vazia ou é um *cons* aplicado a um natural e uma lista.
- A prova por indução sobre lista de naturais é:
  1. Mostrar que a propriedade P é verdadeira quando l é *nil*.
  2. Mostrar que a proposition P é verdadeira quando l é *cons* n l' para algum número n e uma outra lista l', assumindo que P é verdadeiro para l'.

app é associativa

- Não seria possível provar a associativade da função *app* por análise de caso (exercício deixado para o leitor =D).
- A indução abaixo faz o que desejamos.

```
Theorem app_assoc : forall l1 l2 l3 : natlist,
  (l1 ++ l2) ++ l3 = l1 ++ (l2 ++ l3).
Proof.
  intros l1 l2 l3. induction l1 as [| n l1' IHl1'].
  - (* l1 = nil *)
    reflexivity.
  - (* l1 = cons n l1' *)
    simpl. rewrite -> IHl1'. reflexivity. Qed.
```

Revertendo uma lista

- Uma função interessante para provas por indução sobre listas é a função *rev*.
- Diz-se que função *rev* inverte uma lista ou, então, reverte uma lista.

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

Example test_rev1:            rev [1;2;3] = [3;2;1].
Proof. reflexivity.  Qed.
Example test_rev2:            rev nil = nil.
Proof. reflexivity.  Qed.
```

Propriedades de *rev*

- Reverter uma lista não altera o seu comprimento *length*.
- A prova por indução abaixo não consegue avançar devido a impossibilidade de computar o tamanho *length (rev l' ++ [n])*.

```
Theorem rev_length_firsttry : forall l : natlist,
  length (rev l) = length l.
Proof.
  intros l. induction l as [| n l' IHl'].
  - (* l = [] *)
    reflexivity.
  - (* l = n :: l' *)
    simpl.
    rewrite <- IHl'.
Abort.
```

Uma propriedade auxiliar envolvendo *app* e *length*.

- O tamanho da concatenação é a soma dos tamanhos!

```
Theorem app_length : forall l1 l2 : natlist,
  length (l1 ++ l2) = (length l1) + (length l2).
Proof.
  (* WORKED IN CLASS *)
  intros l1 l2. induction l1 as [| n l1' IHl1'].
  - (* l1 = nil *)
    reflexivity.
  - (* l1 = cons *)
    simpl. rewrite -> IHl1'. reflexivity.  Qed.
```

Segunda tentativa de rev _length

- Usando rewrite com app _length podemos finalizar a prova anterior!

```
Theorem rev_length : forall l : natlist,
  length (rev l) = length l.
Proof.
  intros l. induction l as [| n l' IHl'].
  - (* l = nil *)
    reflexivity.
  - (* l = cons *)
    simpl. rewrite -> app_length, plus_comm.
    simpl. rewrite -> IHl'. reflexivity.  Qed.
```

Dica quente: comando *Search*

- Algumas provas necessitam de outras provas. Para se referir à outras provas é necessário lembrar de seus nomes, o que pode ser bem difícil.
- O comando *Search* permite buscar por teoremas/lemmas/programas.

```
Search rev.
```