Aula 2 - Programação Funcional em Coq

Rafael Castro - rafaelcgs10.github.io/coq

02/05/2018

O que é Programação Funcional?

- O paradigma de programação funcional é baseado no simples conceito: se funções não têm efeitos colaterais, então tudo o que precisamos saber sobre funções são os seus mapeamentos. - (se ignorarmos propriedades não funcionais como a eficiência!).
- O termo funcional enfatiza que funções são *first class values*, ou seja, elas podem ser passadas como argumento para funções e retornadas por funções.

Baby's First Functional Program

- Um programa funcional é apenas um mapeamento!
- Um programa (não recursivo) é definido com *Definition*, seguido pelo identificador, a sequência de argumentos, o tipo do seu retorno e a sua definição.
- Casamento de padrão desconstroi um identificador em seus possíveis termos. A seta => informa o respectivo resultado.

```
Definition next_weekday (d:day) : day :=
  match d with
  | monday => tuesday
  | tuesday => wednesday
  | wednesday => thursday
  | thursday => friday
  | friday => monday
  | saturday => monday
  | sunday => monday
  end.
```

Executando computações em Coq

```
Compute (next_weekday friday).

Compute (next_weekday (next_weekday saturday)).
```

Baby's First Proof about some Function

- *Example* é o mesmo que *Theorem*, que também é o mesmo que *Lemma*.

```
Example test_next_weekday:
  (next_weekday (next_weekday saturday)) = tuesday.
Proof.
  simpl.
  reflexivity.
Qed.
```

Tipo booleano

- O tipo padrão booleano é definido em Coq por:

```
Inductive bool : Type :=
  | true : bool
  | false : bool.
```

- Como isso, de alguma, maneira representa os booleanos? De forma alguma!
- Booleanos tem sua semântica dada pelas suas operações (tabela verdade).

Funções sobre booleanos

```
Definition negb (b:bool) : bool :=
  match b with
  | true => false
  | false => true
  end.
Definition andb (b1:bool) (b2:bool) : bool :=
  match b1 with
  | true => b2
  | false => false
  end.
Definition orb (b1:bool) (b2:bool) : bool :=
  match b1 with
  | true => true
  | false => b2
  end.
```

Pequenas provas como testes

- Apenas alguns testes de nossas funções.

```
Example test_orb1: (orb true false) = true.
Proof. reflexivity. Qed.
Example test_orb2: (orb false false) = false.
Proof. reflexivity. Qed.
Example test_orb3: (orb false true) = true.
Proof. reflexivity. Qed.
Example test_orb4: (orb true true) = true.
Proof. reflexivity. Qed.
```

Notações para definições

```
Notation "x && y" := (andb x y).
Notation "x || y" := (orb x y).

Example test_orb5: false || false || true = true.
Proof. simpl. reflexivity. Qed.
```

Módulos em Coq

- Coq tem um sistema de módulos para organizar o desenvolvimento de sistemas grandes.
- Cerca-se entre *Module X* e *End X*, onde *X* é o nome do módulo.
- Depois do *End X* as definições desse módulo são referidas como *X.foo*.

Números Naturais

- A definição é como na Aritmética de Peano. Todos os 8 axiomas são contemplados pela definição indutiva abaixo. ~ *Falaremos mais disso depois!*
- 0 é um número natural
- *S* define o sucessor de um número natural. Se *p* é um número natural, então *S p* também é.
- Cuidado! *S* é uma função que apenas define (indutivamente) os elementos de nat e não representa uma computação. *S* define o sucessor, ele não computa o sucessor (não é (+1)).

```
Module NatPlayground.

 Inductive nat : Type :=
  | O : nat
  | S : nat -> nat.
```

Primeira função sobre naturais

```
Definition pred (n : nat) : nat :=
  match n with
    | O => O
    | S n' => n'
  end.

End NatPlayground.

Check (S (S (S (S O)))).

Compute (pred 4).

Check S.
Check pred.
```