

Aula 3 - Recursão Primitiva em Coq

Rafael Castro - rafaelcgs10.github.io/coq

07/05/2018



Terminação em Coq

- Gallina é total. Logo, terminação é uma propriedade crucial da linguagem Gallina.
- Programas que não terminam são equivalentes à proposições falsa.
- Escrever um programa que não termina em Coq é provar o falso, é introduzir inconsistência.

Recursão Primitiva

- Recursão Estrita, Recursão primitiva e Recursão estrutural significam a mesma coisa.
- Um tipo especial de recursão.
- Todas as funções com recursão estrita terminam, portanto são totais.
- Mas nem todas as funções totais são de recursão primitiva!
- São como laços contados em linguagens imperativas, com limite de iteração fixo (*for*, etc. . .).
- As funções da Aritmética de Peano são exatamente essas com recursão primitiva (adição, subtração, multiplicação. . .).

Recursão Primitiva em Coq

- Em Gallina, funções com recursão primitiva são aquelas cuja chamada recursiva acontece apenas em subtermos sintáticos do argumento original.
- Ou seja, a chamada recursiva ocorre num subtermo que pode ser capturado por um casamento de padrão.
- Isso é feito por meio do comando *Fixpoint*, que é similar ao *Definition* mas permite recursão primitiva.
- Por exemplo, a função abaixo testa se um número é par.
- Note que n' é subtermo de n .

```
Fixpoint evenb (n:nat) : bool :=  
  match n with  
  | 0 => true  
  | S 0 => false  
  | S (S n') => evenb n'  
  end.
```

Apenas alguns testes

- Vamos aproveitar para definir ímpar como sendo não par.
- Vamos testar!

Definition oddb (n:nat) : bool := negb (evenb n).

Example test_oddb1: oddb 1 = true.

Proof. simpl. reflexivity. Qed.

Example test_oddb2: oddb 4 = false.

Proof. simpl. reflexivity. Qed.



Função soma de Peano

- A função soma abaixo recebe dois números.
- É feito o casamento de padrão no primeiro argumento:
 - ① Se for zero, então o resultado é o valor do segundo número.
 - ② Se for um sucessor, então o resultado é o sucessor da soma de n' com m . Adiciona um sucessor no resultado para cada sucessor do primeiro argumento.

```
Module NatPlayground2.
```

```
Fixpoint plus (n : nat) (m : nat) : nat :=
```

```
  match n with
```

```
    | 0 => m
```

```
    | S n' => S (plus n' m)
```

```
end.
```



Exemplo de soma

Compute (plus 3 2).

```
(* plus (S (S (S 0))) (S (S 0)))  
==> S (plus (S (S 0)) (S (S 0)))  
      by the second clause of the match  
==> S (S (plus (S 0) (S (S 0))))  
      by the second clause of the match  
==> S (S (S (plus 0 (S (S 0)))))  
      by the second clause of the match  
==> S (S (S (S (S 0))))  
      by the first clause of the match  
*)
```



Multiplicação de Peano

- Se os argumentos tiverem o mesmo tipo eles podem ser agrupados como $(n\ m : nat)$.
- A função multiplicação soma o segundo argumento uma vez para cada sucessor do primeiro argumento.

```
Fixpoint mult (n m : nat) : nat :=  
  match n with  
  | 0 => 0  
  | S n' => plus m (mult n' m)  
  end.
```

Example test_mult1: (mult 3 3) = 9.
Proof. simpl. reflexivity. Qed.



Subtração de Peano

- É possível casar dois valores ao mesmo tempo ao colocar uma vírgula entre eles.
- `_` é um padrão coringa.
- A função subtração funciona assim:
 - 1 Zero menos qualquer coisa dá zero. (Lembre-se que não tem negativo e a função precisa ser total).
 - 2 Um número (sucessor) menos zero resulta nesse número.
 - 3 Um número (sucessor) menos um número (sucessor) resulta na subtração dos predecessores.

```
Fixpoint minus (n m:nat) : nat :=  
  match n, m with  
  | 0 , _ => 0  
  | S _ , 0 => n  
  | S n' , S m' => minus n' m'  
end.
```

Algumas Notações para A.P.

Notation `"x + y" := (plus x y)`
 (at level 50, left associativity)
 : nat_scope.

Notation `"x - y" := (minus x y)`
 (at level 50, left associativity)
 : nat_scope.

Notation `"x * y" := (mult x y)`
 (at level 40, left associativity)
 : nat_scope.

Check `((0 + 1) + 1).`

Função fatorial

Exercício realizado em aula

$factorial(0) = 1$

$factorial(n) = n * factorial(n-1)$

Fixpoint factorial (n:nat) : nat

(* REPLACE THIS LINE WITH " := _your_definition_ ." *). Adm

Example test_factorial1: (factorial 3) = 6.

(* FILL IN HERE *) Admitted.

Example test_factorial2: (factorial 5) = (mult 10 12).

(* FILL IN HERE *) Admitted

Função teste de igualdade booleano

- Como funciona a seguinte função de igualdade booleana?

```
Fixpoint beq_nat (n m : nat) : bool :=  
  match n with  
  | 0 => match m with  
  | 0 => true  
  | S m' => false  
  end  
  | S n' => match m with  
  | 0 => false  
  | S m' => beq_nat n' m'  
  end  
end.
```

Função teste de menor ou igual booleano

- Como funciona a seguinte função de menor/igual booleana?

```
Fixpoint leb_nat (n m : nat) : bool :=  
  match n with  
  | 0 => true  
  | S n' =>  
    match m with  
    | 0 => false  
    | S m' => leb_nat n' m'  
    end  
  end.  
end.
```

Example test_leb1: (leb_nat 2 2) = true.

Proof. simpl. reflexivity. Qed.

Example test_leb2: (leb_nat 2 4) = true.

Proof. simpl. reflexivity. Qed.

Função teste menor booleano

Exercício realizado em sala

- Dica: $n < m \leftrightarrow ? \leq ?$

```
Definition ltb_nat (n m : nat) : bool
```

```
(* REPLACE THIS LINE WITH " := _your_definition_ ." *). Adm
```

```
Example test_ltb_nat1: (ltb_nat 2 2) = false.
```

```
(* FILL IN HERE *) Admitted.
```

```
Example test_ltb_nat2: (ltb_nat 2 4) = true.
```

```
(* FILL IN HERE *) Admitted.
```

```
Example test_ltb_nat3: (ltb_nat 4 2) = false.
```

```
(* FILL IN HERE *) Admitted.
```