Provadores de Teoremas e suas Aplicações

21/09/2018

Problema

- Os dois últimos exercícios do script `"Induction.v"` do livro Software Foundations do Benjamin Pierce:

  https://softwarefoundations.cis.upenn.edu/lf-current/Induction.html

- É necessário baixar o livro (opção “download” em https://softwarefoundations.cis.upenn.edu/lf-current/index.html) para ter acesso ao script `"Induction.v"` e conseqüentemente aos exercícios discutidos a seguir.

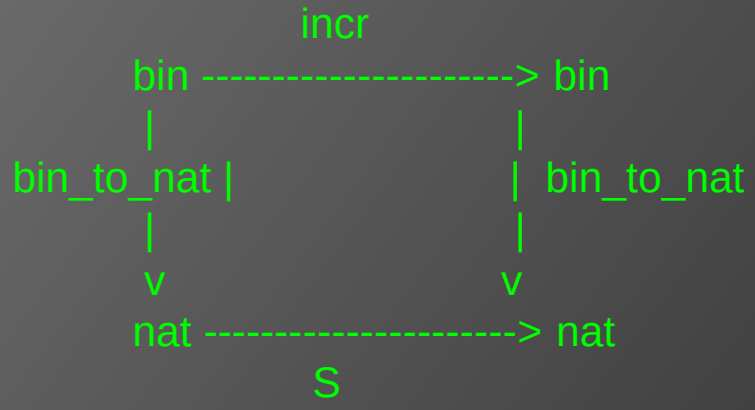
- São apresentadas cinco tentativas de solução, usando definições diferentes;

- As duas primeiras estão incompletas, a terceira está completa, a quarta é a solução oficial e a quinta usa a definição de normalização da quarta;

- A terceira e a quarta são similares, mudando apenas a definição da função de normalização; por causa disse, a solução oficial é mais simples do que a terceira tentativa. A quinta ilustra a simplicidade da quinta solução.

Problema

(** **** Exercise: 3 stars, recommended (binary_commute) *)
(** Recall the [incr] and [bin_to_nat] functions that you wrote for the [binary] exercise in the [Basics] chapter. Prove that the following diagram commutes:



That is, incrementing a binary number and then converting it to a (unary) natural number yields the same result as first converting it to a natural number and then incrementing.
Name your theorem [bin_to_nat_pres_incr] ("pres" for "preserves").

Before you start working on this exercise, copy the definitions from your solution to the [binary] exercise here so that this file can be graded on its own. If you want to change your original definitions to make the property easier to prove, feel free to do so! *)

Problema

(** **** Exercise: 5 stars, advanced (binary_inverse) *)
(** This exercise is a continuation of the previous exercise about binary numbers. You will need your definitions and theorems from there to complete this one; please copy them to this file to make it self contained for grading.

(a) First, write a function to convert natural numbers to binary numbers. Then prove that starting with any natural number, converting to binary, then converting back yields the same natural number you started with.

(b) You might naturally think that we should also prove the opposite direction: that starting with a binary number, converting to a natural, and then back to binary yields the same number we started with. However, this is not true! Explain what the problem is.

(c) Define a "direct" normalization function -- i.e., a function [normalize] from binary numbers to binary numbers such that, for any binary number b, converting to a natural and then back to binary yields [(normalize b)]. Prove it. (Warning: This part is tricky!)

Again, feel free to change your earlier definitions if this helps here. *)

Tentativas

- **Primeira:**
  Original, usando três tipos diferentes para representar números binários;
  Incompleta.

- **Segunda:**
  Original, usando um único tipo e uma função de normalização;
  Incompleta.

- **Terceira:**
  Original, usando um único tipo e uma função de normalização;
  Construtores renomeados;
  Completa.

- **Quarta**:
  Oficial,usando um único tipo e uma função de normalização;
  A função de normalização é diferente da terceira;

- **Quinta:**
  Idêntica à terceira, porém usando a função de normalização da quarta.

Primeira tentativa

- Usando tipos compostos:

Número positivo em binário:

```
Inductive pbin: Type:=
| x: pbin
| e: pbin -> pbin
| o: pbin -> pbin.
```

Número positivo em binário "opcional":

```
Definition optpbin: Type:= option pbin.
```

Número natural em binário:

```
Inductive bin: Type:=
| z: bin
| p: pbin -> bin.
```

Primeira tentativa

- OK para:

nat → bin → nat :

```
Lemma nat_to_bin_to_nat:
forall n: nat,
bin_to_nat (nat_to_bin n) = n.
```

- NOK para:

bin → nat → bin:

```
Lemma bin_to_nat_to_bin:
forall n: bin,
nat_to_bin (bin_to_nat n) = n.
```

- Extensa biblioteca de lemas e definições;
- Longa e complexa;
- Incompleta.

Segunda tentativa

- Definição única com função de normalização:

Número em binário:

```
Inductive bin: Type:=
| z: bin
| e: bin -> bin
| o: bin -> bin.
```

Função de normalização:

```
Fixpoint norm (n: bin): bin:=
match n with
| z => z
| e n' => match (norm n') with
          | z => z
          | _ => e (norm n')
          end
| o n' => o (norm n')
end.
```

Segunda tentativa

- OK para:

nat → bin → nat :

```
Lemma nat_to_bin_to_nat:
forall n: nat,
bin_to_nat (nat_to_bin n) = n.
```

- NOK para:

bin → nat → bin:

```
Lemma bin_to_nat_to_bin:
forall n: bin,
nat_to_bin (bin_to_nat n) = (norm n).
```

- Extensa biblioteca de lemas e definições;
- Incompleta.

Primeira e Segunda

- A representação de binários é unívoca na primeira;
- As provas, no entanto, se tornaram bastante complexas;
- A representação de binários na segunda não é unívoca;
- Necessidade de uma função de “normalização” que elimine zeros à esquerda;

```
Para o 0 (z):
r0 z
r0 (r0 z))
r0 (r0 (r0 z)))
...
Para o 1 (r1 z):
r1 (r0 z)
r1 (r0 (r0 z))
...
```

- A conversão de um binário para natural e depois de volta para binário produz representações normalizadas;
- Problema não acontece com a conversão de natural para binário e depois de volta para natural;
- Daí a necessidade de uso da função de normalização no enunciado do segundo lema.

Terceira tentativa

- Definição única com função de normalização:

Número em binário:

```
Inductive bin: Type:=
| z: bin
| r0: bin -> bin
| r1: bin -> bin.
```

Função de normalização:

```
Fixpoint norm (n: bin): bin:=
match n with
| z => z
| r0 n' => match (norm n') with
           | z => z
           | _ => r0 (norm n') end
| r1 n' => r1 (norm n')
end.
```

Terceira tentativa

- OK para:

nat → bin → nat :

```
Lemma nat_to_bin_to_nat:
forall n: nat,
bin_to_nat (nat_to_bin n) = n.
```

- OK para:

bin → nat → bin:

```
Lemma bin_to_nat_to_bin:
forall b: bin,
nat_to_bin (bin_to_nat b) = (norm b).
```

- Extensa biblioteca de lemas e definições;
- Depende da soma binária (bin_plus) e diversos lemas auxiliares;
- Completa.

Terceira tentativa

- Lemas auxiliares relacionados com propriedades da função de normalização e da adição binária:

```
Lemma norm_inc:
forall b: bin,
inc (norm b) = norm (inc b).

Lemma r0_norm_bin_plus:
forall b: bin,
norm (r0 b) = norm (bin_plus b b).

Lemma r1_norm_inc_bin_plus:
forall b: bin,
norm (r1 b) = norm (inc (bin_plus b b)).

Lemma norm_bin_plus:
forall b: bin,
bin_plus (norm b) (norm b) = norm (bin_plus b b).
```

Segunda e Terceira

- São similares;
- Mudança nos nomes dos construtores;
- Mesma função de normalização;
- Necessidade de definir soma binária e provar vários lemas auxiliares, entre eles os apresentados no slide anterior;
- Mais complexa e extensa;

Quarta e Quinta

Número em binário:

```
Inductive bin: Type:=
| z: bin
| r0: bin -> bin
| r1: bin -> bin.
```

Função de normalização:

```
Definition double (b: bin): bin :=
match b with
| z => z
| _ => r0 b end.

Fixpoint norm (b:bin): bin :=
match b with
| z => z
| r0 b => double (norm b)
| r1 b => r1 (norm b) end.
```

Quarta e Quinta

- São similares;
- Função de normalização da quarta é a mesma da quinta;
- Reescrita (quinta) para facilitar a leitura.

Terceira e Quarta

- São similares;
- Função de normalização é diferente;
- Não é necessário definir e usar soma binária nem os lemas do slide anterior;
- Quarta é mais simples e menor

- **A escolha das definições pode afetar consideravelmente a complexidade das provas!**
- **É importante avaliar bem as definições que serão empregadas!**