

Ćwiczenia teoretyczne

Wojciech Jedynek

Paweł Wieczorek

21 września 2011

Meta notatki

Proponuję wybrać jakąś notację dla teorii typów Martina-Löfa. Może tę z książki „Programming in...”?

Czy chcemy mieć narysowane drzewa w proponowanych rozwiązaniach? W książce Martina-Löfa on czasem rysuje drzewa by coś pokazać. Musielibyśmy się zdecydować by było konsekwentnie.

1 Zadania na koniec

Zadanie 1. Skonstruuj negację bitową, tzn term $negb : N_2 \rightarrow N_2$ a następnie skonstruuj term o typie:

$$(\Pi b \in N_2) \neg (b \equiv negb\ b)$$

Wskazówka: Trudność to dowód że 0_2 jest różne od 1_2 , ponieważ nie mamy typów indukcyjnych to to nie jest oczywiste, trzeba spróbować jak z czwartym aksjomatem Peano na seminarium.

Zadanie 2. Udowodnij, że nie istnieje funkcja z liczb naturalnych do ciągów zero-jedynkowych, która ma odwrotność. To jest skonstruuj term thm o następującym typie:

$$\neg (\Sigma f \in N \rightarrow BinSeq) (\Sigma g \in BinSeq \rightarrow N) (\Pi s \in BinSeq) f(g\ s) \equiv s$$

gdzie $BinSeq$ oznacza $N \rightarrow N_2$.

Wskazówka: Dowód tego twierdzenia to standardowy przykład metody przekątniowej, można znaleźć rozwiązanie w Whitebooku. Trudność polega na przeniesieniu tego na naturalną dedukcję (Zaskakujące może być, że to twierdzenie jest konstruktywne!).

Propozycja rozwiązania. Zanim sformalizujemy metodę przekątniową przypomnijmy sobie ten dowód, by ustalić co konkretnie chcemy uzyskać.

Pokażmy, że dla dowolnych funkcji f i g potrafimy dojść do sprzeczności o ile g jest odwrotnością f . Stwórzmy „przekątniowy” ciąg zero-jedynkowy $h : BinSeq$:

$$h = \lambda n. negb\ (f\ n\ n)$$

Element tego ciągu o numerze n jest równy negacji n -tego elementu w n -tym ciągu w wyznaczonej numeracji przez funkcję f . Za pomocą funkcji g możemy uzyskać numer tego ciągu, niech $n_h = g\ h$. Teraz, zauważmy że

$$h\ n_h = negb\ (f\ n_h\ n_h) = negb\ (f\ (g\ h)\ n_h) = negb\ (h\ n_h)$$

czyli sprzeczność.

Możemy teraz zacząć zastanawiać się jak przenieść powyższe rozumowanie na naturalną dedukcję, musimy spróbować skonstruować funkcję *diagonal* o następującym typie:

$$(\Pi f \in N \rightarrow BinSeq) (\Pi g \in BinSeq \rightarrow N) \rightarrow ((\Pi s \in BinSeq) f(g\ s) \equiv s) \rightarrow N_0$$

Chcemy aby odpowiadała ona przedstawionemu rozumowaniu. Term zacząć pisać prosto:

$$\text{diagonal} = \lambda f. \lambda g. \lambda C. \boxed{?}$$

W miejscu $\boxed{?}$ chcemy skonstruować absurdalną wartość. Ale jak? Sprzeczność uzyskaliśmy pokazując, że $h\ n_h = \text{negb}\ (h\ n_h)$, bo wiemy że dla dowolnego b zachodzi $b \neq \text{negb}\ b$.

Wykorzystajmy te szczegóły w praktyce: z poprzedniego zadania mamy term $\text{Hnegb} : (\Pi b \in N_2) \neg (b \equiv \text{negb}\ b)$, czyli pamiętając co rozumiemy jako negację - jesteśmy w posiadaniu metody, która z dowodu $b \equiv \text{negb}\ b$ konstruuje absurdalną wartość. Wykorzystajmy tę metodę dla $h\ n_h$.

$$\text{diagonal} = \lambda f. \lambda g. \lambda C. \text{Hnegb}\ (h\ n_h)\ \text{Heq}$$

gdzie

$$\text{Heq} = \boxed{?} : h\ n_h \equiv \text{negb}\ (h\ n_h)$$

By skonstruować świadka tej równości musimy przeanalizować ciąg równości w oryginalnym rozumowaniu - pierwsze dwie

$$h\ n_h = \text{negb}\ (f\ n_h\ n_h) = \text{negb}\ (f\ (g\ h)\ n_h)$$

mamy z definicji h oraz n_h . Czyli interesuje nas jedynie term typu

$$\text{negb}\ (f\ (g\ h)\ n_h) \equiv \text{negb}\ (h\ n_h)$$

...

Dowód twierdzenia, to funkcja która ze świadka istnienia takich funkcji f i g ma konstruować wartość absurdalną, która jedyne co musi zrobić to rozpakować dane z argumentu i zaaplikować do nich funkcję *diagonal*.

$$\text{thm} = \lambda H. \Sigma\text{-elim}\ (\lambda f. \lambda H'. \Sigma\text{-elim}\ (\lambda g. \lambda H''. \text{diagonal}\ f\ g\ H'')\ H')\ H$$

Wersja w Agdzie (musze wybadać jak UTF w Verbatim robić)

module Cantor where

```
open import Data.Product
open import Data.Nat
open import Data.Empty
open import Data.Bool
open import Relation.Nullary
open import Relation.Binary.PropositionalEquality
```

```
negb : Bool -> Bool
negb true  = false
negb false = true
```

```
Hnegb : (b : Bool) -> ~b == negb b
Hnegb true ()
Hnegb false ()
```

```
BinSeq : Set
BinSeq = N -> Bool
```

```
E-elim : {A : Set} {B : A -> Set} {P : Ex A B -> Set}
        -> ( (a : A) -> (b : B a) -> P (a , b) )
```

```

      -> (p : Ex A B) -> P p
E-elim H (a , b) = H a b

thm : ~ Ex \ (f : N -> BinSeq) -> Ex \ (g : BinSeq -> N)
      -> ( (s : BinSeq) -> f (g s) == s )
thm H = Ex-elim (\f H' -> Ex-elim (\g H'' -> diagonal f g H'') H') H
where

diagonal : (f : N -> BinSeq) (g : BinSeq -> N)
           -> ( (s : BinSeq) -> f (g s) == s )
           -> False
diagonal f g C = Hnegb (h nh) Heq
where
  h : BinSeq
  h n = negb (f n n)

  nh = g h

Heq : negb (f (g h) nh) == negb (h nh)
Heq = subst (\ p -> negb (p nh) == (negb (h nh))) (sym (C h)) refl

```

□