Ćwiczenia z Agdy - Lista 1.

Wojciech Jedynak   Paweł Wieczorek

18 października 2011

```
module Exercises where
```

1 Podstawy Izomorfizmu Curry'ego-Howarda

Fałsz zdefiniowaliśmy w Agdzie jako typ pusty:

```
data ⊥ : Set where
⊥-elim : {A : Set} → ⊥ → A
⊥-elim ()
```

Możemy teraz wyrazić negację w standardowy sposób: jako funkcję w zbiór pusty.

```
¬_ : Set → Set
¬ A = A → ⊥
```

Zadanie 1 *Udowodnij, że $p \Rightarrow \neg\neg p$, czyli dokończ poniższą definicję:*

```
pnnp : {A : Set} → A → ¬ ¬ A
pnnp = {!!}
```

Czy potrafisz udowodnić implikację w drugą stronę?

Zadanie 2 *Udowodnij prawo kontrapozycji, czyli pokaż że $(A \Rightarrow B) \Rightarrow (\neg\ B \Rightarrow \neg\ A)$. Czy potrafisz udowodnić twierdzenie odwrotne?*

Zadanie 3 *Zdefiniuj typ, który będzie odpowiadał formule atomowej $\top$. Jakiemu typowi z języków programowania on odpowiada?*

Zadanie 4 *Polimorficzne pary, czyli odpowiednik koniukcji możemy zdefiniować następująco:*

$$\textbf{\textit{data}}\ \_\wedge\_\ (A\ B\ :\ Set)\ :\ Set\ \textbf{\textit{where}}$$
$$pair\ :\ (a\ :\ A) \to (b\ :\ B) \to A \wedge B$$

Udowodnij reguły eliminacji oraz prawo przemienności tj. zdefiniuj funkcje fst, snd i swap:

$$fst\ :\ \{A\ B\ :\ Set\} \to A \wedge B \to A$$
$$fst\ =\ \{!!\}$$
$$snd\ :\ \{A\ B\ :\ Set\} \to A \wedge B \to B$$
$$snd\ =\ \{!!\}$$
$$swap\ :\ \{A\ B\ :\ Set\} \to A \wedge B \to B \wedge A$$
$$swap\ =\ \{!!\}$$

Zadanie 5 *Zdefiniuj typ sumy rozłącznej (czyli Haskellowy typ Either). Jakiemu spójnikowi logicznemu odpowiada ten typ? Udowodnij przemienność tego spójnika.*

Zadanie 6 *Korzystając z typów z poprzednich dwóch zadań, sformułuj i spróbuj udowodnić prawa deMorgana znane z logiki klasycznej. Które z nich zachodzą w logice kostruktywnej?*

2 Typ identycznościowy, czyli równość w języku

Aby wydawać i dowodzić sądy o równości różnych bytów, zdefiniowaliśmy tzw. typ identycznościowy. Typ ustalonych a,b $\in$ A, a $\equiv$ b jest zamieszkały wtw, gdy a i b obliczają się do tego samego elementu kanonicznego (wartości).

$$\textbf{infix}\ 5\ \_\equiv\_$$
$$\textbf{data}\ \_\equiv\_\ \{\mathsf{A}\ :\ \mathsf{Set}\}\ :\ \mathsf{A} \to \mathsf{A} \to \mathsf{Set}\ \textbf{where}$$
$$\mathsf{refl}\ :\ \{\mathsf{a}\ :\ \mathsf{A}\} \to \mathsf{a} \equiv \mathsf{a}$$

Różność elementów definiujemy jako... zaprzeczenie równości:

$$\_\not\equiv\_\ :\ \{\mathsf{A}\ :\ \mathsf{Set}\} \to \mathsf{A} \to \mathsf{A} \to \mathsf{Set}$$
$$\mathsf{a} \not\equiv \mathsf{b}\ =\ \neg\ (\mathsf{a} \equiv \mathsf{b})$$

Zadanie 7 *Pokazaliśmy już jak udowodnić niektóre własności równości:*

$$symm\ :\ \{A\ :\ Set\} \to (a\ b\ :\ A) \to a \equiv b \to b \equiv a$$
$$symm\ a\ .a\ refl\ =\ refl$$
$$subst\ :\ \{A\ :\ Set\} \to (P\ :\ A \to Set) \to (a\ b\ :\ A) \to a \equiv b \to P\ a \to P\ b$$
$$subst\ P\ a\ .a\ refl\ Pa\ =\ Pa$$

Udowodnij dwie dodatkowe (bardzo przydatne) własności:

$$\mathit{trans} : \{A : \mathit{Set}\} \to (a\ b\ c : A) \to a \equiv b \to b \equiv c \to a \equiv c$$
$$\mathit{trans} = \{!!\}$$
$$\mathit{cong} : \{A\ B : \mathit{Set}\} \to (P : A \to B) \to (a\ b : A) \to a \equiv b \to P\ a \equiv P\ b$$
$$\mathit{cong} = \{!!\}$$

3 Wartości boolowskie

Wartości boolowskie zdefiniowaliśmy następująco:

```
data Bool : Set where
  false : Bool
  true : Bool
```

Zadanie 8 *Uzupełnij poniższą definicję tak, aby otrzymać funkcję negacji boolowskiej.*

$$\mathit{not} : \mathit{Bool} \to \mathit{Bool}$$
$$\mathit{not}\ \mathit{false} = \{!!\}$$
$$\mathit{not}\ \mathit{true} = \{!!\}$$

Udowodnij, że Twoja definicja spełnia następujące własności:

$$\mathit{not\text{-}has\text{-}no\text{-}fixed\text{-}points} : (b : \mathit{Bool}) \to \mathit{not}\ b \not\equiv b$$
$$\mathit{not\text{-}has\text{-}no\text{-}fixed\text{-}points} = \{!!\}$$
$$\mathit{not\text{-}is\text{-}involutive} : (b : \mathit{Bool}) \to \mathit{not}\ (\mathit{not}\ b) \equiv b$$
$$\mathit{not\text{-}is\text{-}involutive} = \{!!\}$$

4 Liczby naturalne

Na wykładzie zdefiniowaliśmy liczby naturalne z dodawaniem następująco:

```
data ℕ : Set where
  zero : ℕ
  suc : ℕ → ℕ
 {-# BUILTIN NATURAL ℕ #-}
 {-# BUILTIN ZERO zero #-}
 {-# BUILTIN SUC suc #-}
infix 6 _+_
_+_ : ℕ → ℕ → ℕ
zero + m = m
suc n + m = suc (n + m)
```

Zadanie 9 *Pamiętając, że wg ICH indukcja = rekursja, udowodnij następujące własności dodawania:*

```
plus-right-zero : (n : ℕ) → n + 0 ≡ n
plus-right-zero = {!!}
plus-suc-n-m : (n m : ℕ) → suc (n + m) ≡ n + suc m
plus-suc-n-m = {!!}
```

Zadanie 10 *Korzystając z poprzedniego zadania, udowodnij przemienność dodawania:*

```
plus-commutative : (n m : ℕ) → n + m ≡ m + n
plus-commutative = {!!}
```

Zadanie 11 *Zdefiniuj mnożenie i potęgowanie dla liczb naturalnych.*

```
infix 7 _*_
infix 8 _^_
_*_ : ℕ → ℕ → ℕ
n * m = {!!}
_^_ : ℕ → ℕ → ℕ
n ↑ m = {!!}   -- tutaj też ma być daszek :-)
```

Jeśli masz ochotę, to udowodnij przemienność mnożenia.

Zadanie 12 *Udowodnij, że zero* $\not\equiv$ *suc (zero).*

Zadanie 13 *Udowodnij następującą własność ("prawo skracania"):*

```
strip-suc : (n m : ℕ) → suc n ≡ suc m → n ≡ m
strip-suc = {!!}
```

5 Wektory

Przypomnijmy definicję wektorów:

```
data Vec (A : Set) : ℕ → Set where
  [] : Vec A 0
  _::_ : {n : ℕ} → (x : A) → (xs : Vec A n) → Vec A (suc n)
```

Zdefiniowaliśmy już m.in. konkatenację wektorów:

```
_++_ : {A : Set} → {n m : ℕ} → Vec A n → Vec A m → Vec A (n + m)
[] ++ v2 = v2
(x :: v1) ++ v2 = x :: (v1 ++ v2)
```

Zadanie 14 *Zaprogramuj funkcje vmap, która jest wektorowym odpowiednikiem map dla list. Jaka powinna być długość wynikowego wektora?*

Zadanie 15 *W Haskellu bardzo często używamy funkcji zip, która jest zdefiniowana następująco:*

```
zip :: [a] -> [b] -> [(a,b)]
zip (x:xs) (y:ys) = (x,y) : zip xs ys
zip _ _ = []
```

Jak widać, przyjęto tutaj, że jeśli listy są różnej długości, to dłuższa lista jest ucinana. Nie zawsze takie rozwiązanie jest satysfakcjonujące. Wymyśl taką sygnaturę dla funkcji zip na wektorach, aby nie dopuścić (statycznie, za pomocą systemu typów) do niebezpiecznych wywołań.

Zadanie 16 *Zaprogramuj wydajną funkcję odwracającą wektor. Użyj funkcji subst, jeśli będziesz chcieć zmusić Agdę do stosowania praw arytmetyki.*

6 Zbiory skończone - typ fin

Przypomnijmy definicję typu fin:

```
data Fin : ℕ → Set where
  zero : {n : ℕ} → Fin (suc n)
  suc  : {n : ℕ} → (i : Fin n) → Fin (suc n)
```

Dla dowolnego n ∈ ℕ typ Fin n ma dokładnie n mieszkańców (w szczególności Fin 0 jest pusty). Własność ta sprawia, że typ Fin świetnie nadaje się do indeksowania wektorów. Indeksowanie to jest bezpieczne, gdyż system typów wyklucza nam możliwość stworzenia indeksu, który wykraczałby poza dozwolony zakres.

```
_!_ : {A : Set} {n : ℕ} → Vec A n → Fin n → A
[] ! ()
(x :: xs) ! zero = x
(x :: xs) ! suc i = xs ! i
```

Zadanie 17 *Zauważ, że typ Fin przypomina strukturą liczby naturalne, choć zawiera więcej informacji.*
Napisz funkcję konwersji, która "zapomina" te dodatkowe informacje.
Np. jeśli Fin 2 = $\{ 0_2, 1_2 \}$, to chcemy mieć
*to*ℕ $0_2 \equiv 0$
*to*ℕ $1_2 \equiv 1$

$$\mathit{to}\mathbb{N} : \{n : \mathbb{N}\} \to \mathit{Fin}\ n \to \mathbb{N}$$
$$\mathit{to}\mathbb{N} = \{!!\}$$

Zadanie 18 *Napisz funkcje, która dla danego n, zwraca największy element zbioru Fin (suc n). Przez największy rozumiemy tutaj ten element, który jest zbudowany z największej liczby konstruktorów.*
Przykładowo, dla $n \equiv 3$ *mamy* $Fin\ 3 \equiv \{\ 0_3,\ 1_3,\ 2_3\ \}$ *i jako wynik chcemy otrzymać* 2_3.

$$\mathit{fmax} : (n : \mathbb{N}) \to \mathit{Fin}\ (\mathit{suc}\ n)$$
$$\mathit{fmax} = \{!!\}$$

Zadanie 19 *Jeśli udało Ci się zrobić poprzednie dwa zadania, to pokaż, że ich złozenie w jedną ze stron daje identyczność. Czy potrafisz sformułować twierdzenie o złożeniu w drugą stronę?*

$$\mathit{lemma\text{-}max} : (n : \mathbb{N}) \to \mathit{to}\mathbb{N}\ (\mathit{fmax}\ n) \equiv n$$
$$\mathit{lemma\text{-}max} = \{!!\}$$

Zadanie 20 *W matematyce mamy* $\{\ 0,\ 1,\ 2,\ ..\ n\text{-}1\ \} \subseteq \{\ 0,\ 1,\ 2,\ ..\ n\text{-}1,\ n\ \}$. *Zainspirowani tym zawieraniem chcemy teraz pokazać, że typ Fin n można osadzić w Fin (suc n).*
Uwaga. Jednym ze sposobów byłoby po prostu użycie konstruktora suc, ale chcemy zrobić odwzorowanie, w którym k_n *przechodzi na* k_{n+1}.

$$\mathit{fweak} : \{n : \mathbb{N}\} \to \mathit{Fin}\ n \to \mathit{Fin}\ (\mathit{suc}\ n)$$
$$\mathit{fweak} = \{!!\}$$

Po zdefiniowaniu funkcji udowodnij jej poprawność:

$$\mathit{fweak\text{-}correctness} : \{n : \mathbb{N}\} \to (i : \mathit{Fin}\ n) \to \mathit{to}\mathbb{N}\ i \equiv \mathit{to}\mathbb{N}\ (\mathit{fweak}\ i)$$
$$\mathit{fweak\text{-}correctness} = \{!!\}$$

Zadanie 21 *Pokaż, że dla każdego* $n \in \mathbb{N}$, *możemy stablicować wszystkie elementy typu Fin n w n-elementowym wektorze.*

$$\mathit{tabFins} : (n : \mathbb{N}) \to \mathit{Vec}\ (\mathit{Fin}\ n)\ n$$
$$\mathit{tabFins} = \{!!\}$$