

Ćwiczenia z Agdy - Lista 1.

Wojciech Jedynek Paweł Wieczorek

18 października 2011

1 Uwaga

Lista zadań jest dostępna w dwu wariantach: .pdf i .lagda. Wersję .lagda można otworzyć w edytorze tekstu i uzupełniać brakujące fragmenty bez przepisywania wszystkiego.

module Exercises **where**

2 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 kontrpozycji, 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 koniunkcji możemy zdefiniować następująco:

```
data _^_ (A B : Set) : Set where
  pair : (a : A) → (b : B) → A ^ B
```

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

```
fst : {A B : Set} → A ^ B → A
fst = {!!}

snd : {A B : Set} → A ^ B → B
snd = {!!}

swap : {A B : Set} → A ^ B → B ^ 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 konstruktywnej?

3 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).

```
infix 5 _≡_
data _≡_ {A : Set} : A → A → Set where
  refl : {a : A} → a ≡ a
```

Różność elementów definiujemy jako... zaprzeczenie równości:

```
_≠_ : {A : Set} → A → A → Set
a ≠ b = ¬ (a ≡ b)
```

Zadanie 7 Pokazaliśmy już jak udowodnić niektóre własności równości:

```
symm : {A : Set} → (a b : A) → a ≡ b → b ≡ a
symm a .a refl = refl

subst : {A : Set} → (P : A → Set) → (a b : A) → a ≡ b → P a → P b
subst P a .a refl Pa = Pa
```

Udowodnij dwie dodatkowe (bardzo przydatne) własności:

```
trans : {A : Set} → (a b c : A) → a ≡ b → b ≡ c → a ≡ c
trans = {!!}
cong : {A B : Set} → (P : A → B) → (a b : A) → a ≡ b → P a ≡ P b
cong = {!!}
```

4 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.*

```
not : Bool → Bool
not false = {!!}
not true = {!!}
```

Udowodnij, że Twoja definicja spełnia następujące własności:

```
not-has-no-fixed-points : (b : Bool) → not b ≠ b
not-has-no-fixed-points = {!!}
not-is-involutive : (b : Bool) → not (not b) ≡ b
not-is-involutive = {!!}
```

5 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 $zero \neq suc\ zero$.

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

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

6 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. konkatencję 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 funkcję *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 chciał zmusić Agdę do stosowania praw arytmetyki.

7 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 $n \in \mathbb{N}$ 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 strukturę liczby naturalne, choć zawiera więcej informacji.

Napisz funkcję konwersji, która "zapomina" te dodatkowe informacje.

Np. jeśli $\text{Fin } 2 = \{ 0_2, 1_2 \}$, to chcemy mieć

$\text{to}\mathbb{N} \ 0_2 \equiv 0$

$\text{to}\mathbb{N} \ 1_2 \equiv 1$

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

Zadanie 18 Napisz funkcję, 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 .

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

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

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

Zadanie 20 W matematyce mamy $\{0, 1, 2, \dots, n-1\} \subseteq \{0, 1, 2, \dots, n-1, n\}$. 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} .

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

Po zdefiniowaniu funkcji udowodnij jej poprawność:

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

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

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