

# Ćwiczenia

Wojciech Jedynek

Paweł Wieczorek

14 października 2011

**Zadanie 1.** Dodaj nowe zbiory do naszego systemu - listę parametryzowaną zbiorem  $A$  oraz drzewo binarne parametryzowane zbiorem  $A$ , tzn sformułuj wszystkie potrzebne reguły aby takie zbiory wprowadzić.

**Zadanie 2.** Na seminarium nie wprowadziliśmy zbioru odpowiadającego spójnikowi  $\vee$ , pokaż jak dodać taki zbiór. Zadbaj aby jego reguły odpowiadały regułom z systemu naturalnej dedukcji.

**Zadanie 3.** Ustalmy typy  $A$  oraz rodzinę  $B(a)$  dla każdego  $a \in A$  oraz  $C(a, a')$  dla każdych  $a, a' \in A$ . Zbuduj termy o następujących typach:

$$\begin{aligned} & ((\Pi x \in A) (\Pi y \in A) C(x, y)) \rightarrow (\Pi y \in A) (\Pi x \in A) C(x, y) \\ & ((\Sigma x \in A) (\Pi y \in A) C(x, y)) \rightarrow (\Pi y \in A) (\Sigma x \in A) C(x, y) \\ & ((\Pi x \in A) (\Pi y \in A) B(x) \rightarrow B(y)) \rightarrow (\Pi x \in A) (\Pi y \in A) \neg B(y) \rightarrow \neg B(x) \\ & (\neg(\Sigma x \in A) B(x)) \rightarrow (\Pi x \in A) \neg B(x) \end{aligned}$$

**Zadanie 4.** Stwórz wyrażenie *compose*, za pomocą którego stworzymy następującą regułę:

$$\frac{g \in A \rightarrow B \quad f \in B \rightarrow C}{\text{compose}(f, g) \in A \rightarrow C}$$

Następnie zaproponuj regułę dla wersji z typami zależnymi i zdefiniuj odpowiednie wyrażenie *composeDep*.

$$\frac{g \in (\Pi x \in A) B(x) \quad f \in (\Pi x \in A) (\Pi b \in B(x)) C(x, b)}{\text{composeDep}(f, g) \in ?}$$

Zdefiniuj też pomocnicze wyrażenie *apply2*:

$$\frac{f \in (\Pi x \in A) (\Pi y \in B(x)) C(x, y) \quad x \in A \quad y \in B(x)}{\text{apply2}(f, x, y) \in C(x, y)}$$

**Zadanie 5.** Ustalmy typ  $A$ , wyprowadź termy o następujących typach.

$$\begin{aligned} & (\Pi x \in A) [x =_A x] \\ & (\Pi b \in Bool) (\Pi c \in A) [\text{if } b \text{ then } c \text{ else } c =_A c] \end{aligned}$$

**Zadanie 6.** Wyprowadź samodzielnie regułę dla prawa Leibniza (*subst*). Następnie posługując się tą regułą pokaż jak stworzyć reguły dla symetrii i przechodniości.

$$\frac{P(x) \text{ set } [x \in A] \quad a \in A \quad b \in A \quad c \in [a =_A b] \quad p \in P(a)}{\text{subst}(c, p) \in P(b)}$$

Oraz pokaż jak stworzyć *cong* dla poniższej reguły.

$$\frac{f \in A \rightarrow B \quad a \in A \quad b \in A \quad c \in [a =_A b]}{\text{cong}(c, f) \in [\text{apply}(f, a) =_B \text{apply}(f, b)]}$$

**Zadanie 7.** Zdefiniuj funkcję dodawania  $\text{add} \in \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat}$ , a następnie stwórz termy o następujących typach:

$$(\Pi a \in \text{Nat}) [\text{apply2}(\text{add}, 0, a) =_{\text{Nat}} a]$$

$$(\Pi a \in \text{Nat}) [\text{apply2}(\text{add}, a, 0) =_{\text{Nat}} a]$$

$$(\Pi a \in \text{Nat})(\Pi b \in \text{Nat}) [\text{apply2}(\text{add}, \text{succ}(a), b) =_{\text{Nat}} \text{succ}(\text{apply2}(\text{add}, a, b))]$$

$$(\Pi a \in \text{Nat})(\Pi b \in \text{Nat}) [\text{apply2}(\text{add}, a, b) =_{\text{Nat}} \text{apply2}(\text{add}, b, a)]$$

Powyższe równości mogą być traktowane jako specyfikacja dodawania, można teraz założyć że nasz system z typami zależnymi posłużył jednocześnie do zdefiniowania funkcji, zdefiniowania specyfikacji (za pomocą typów identycznościowych) oraz udowodnił że funkcja spełnia te równości – co oznacza że wszystko zostało zweryfikowane w obrębie jednego systemu.

**Zadanie 8.** Niech  $f \in (\Pi x \in A)B(x)$ . Wyprowadź  $[f =_{(\Pi x \in A)B(x)} \lambda x. \text{apply}(f, x)]^1$ . Następnie zapoznaj się z typem *Eq* zdefiniowanym w książce (to typ identycznościowy z silną regułą eliminacji), następnie używając tego typu wyprowadź  $f = \lambda x. \text{apply}(f, x) \in (\Pi x \in A)B(x)$ .

Potrzebna jest dodatkowa reguła eliminacji dla typu funkcyjnego, jest to reguła eliminacji podobna do reguł dla typów indukcyjnych. Mamy rodzinę zbiorów  $C$  poindeksowaną funkcjami.

$$\frac{\begin{array}{l} f \in (\Pi x \in A)B(x) \\ C(v) \text{ set } [v \in (\Pi x \in A)B(x)] \\ d(y) \in C(\lambda y) [y(x) \in B(x) [x \in A]] \end{array}}{\text{funsplit}(f, d) \in C(f)} \quad \frac{\begin{array}{l} b(x) \in B(x) [x \in A] \\ C(v) \text{ set } [v \in (\Pi x \in A)B(x)] \\ d(y) \in C(\lambda y) [y(x) \in B(x) [x \in A]] \end{array}}{\text{funsplit}(\lambda b, d) = d(b) \in C(\lambda b)}$$

Spróbuj zdefiniować *apply* za pomocą *funsplit*.

**Zadanie 9.** Udowodnij następujące twierdzenia:

$$\left( (\Pi x \in A) \text{Eq}(B(x), \text{apply}(f, x), \text{apply}(g, x)) \right) \rightarrow \text{Eq}((\Pi x \in A)B(x), f, g)$$

$$\text{Eq}((\Pi x \in A)B(x), f, g) \rightarrow \left( (\Pi x \in A) \text{Eq}(B(x), \text{apply}(f, x), \text{apply}(g, x)) \right)$$

$$[f =_{(\Pi x \in A)B(x)} g] \rightarrow \left( (\Pi x \in A) [\text{apply}(f, x) =_{B(x)} \text{apply}(g, x)] \right)$$

**Zadanie 10.** Mogąc robić wyrażenia obliczające typy zobaczyliśmy jak można definiować predykaty, przykładowo: zdefiniowaliśmy predykat  $\text{IsZero}(n) = \text{Set}(\text{natrec}(n, \hat{\top}, (x, y) \hat{\perp}))$  który był prawdziwy tylko dla 0. Osiągnęliśmy to zwracając  $\top$  dla  $n = 0$  i  $\perp$  dla innych wartości. Spróbuj powtórzyć dowód że 0 jest różne od  $\text{succ}(n)$ .

<sup>1</sup>skrótowy zapis:  $\lambda x. e \equiv \lambda((x)e)$

**Zadanie 11.** Zrób predykat *Empty* dla list z pierwszego zadania. Następnie, mając ustalony *A*, zdefiniuj wyrażenie *isEmpty*:

$$isEmpty \in (\Pi x s \in List\ A). Empty(xs) \vee \neg Empty(xs)$$

Zastanów się nad różnicą pomiędzy taką funkcją a poniższą, przypomnij sobie interpretację BHK dla alternatywy.

$$isEmpty' \in List\ A \rightarrow Bool$$

Następnie, korzystając z *Empty*, zaprogramuj bezpieczne wersje procedur *head* oraz *tail*.

**Zadanie 12.** Skonstruuj negację bitową, tzn funkcję  $negb \in Bool \rightarrow Bool$  a następnie skonstruuj term o typie:

$$(\Pi b \in Bool) \neg [b =_{Bool} apply(negb, b)]$$

Dodatkowo, zbuduj termy o następujących typach:

$$[apply(negb, true) =_{Bool} false]$$

$$[apply(negb, false) =_{Bool} true]$$

**Zadanie 13.** Udowodnij, że nie istnieje funkcja z liczb naturalnych w ciągi zero-jedynkowe, która ma funkcję odwrotną. To jest skonstruuj term o następującym typie:

$$\neg (\Sigma f \in N \rightarrow BinSeq) (\Sigma g \in BinSeq \rightarrow N) (\Pi s \in BinSeq) [s =_{BinSeq} apply(f, apply(g, s))]$$

gdzie *BinSeq* oznacza  $N \rightarrow Bool$ .

Wskazówka: Dowód tego twierdzenia to standardowy przykład metody przekątniowej, można znaleźć rozwiązanie w *Whitebooku*. Trudność zadania polega na przeniesieniu rozwiązania do naszego systemu.