

System Wspomagania Dowodzenia Coq

27.06.06

Zadanie 1.

Rozpatrujemy wyprowadzenia typowe w rachunku konstrukcji. Niech $\Delta \vdash \star : \square$. Udowodnij, że jeśli $x_1 : A_1; \dots; x_n : A_n \vdash A : B$ oraz dla każdego i , $\Delta \vdash x_i : A_i$, to $\Delta \vdash A : B$.

Zadanie 2.

Dany jest term $t =$

```
fun (A : *) (R R' : A -> A -> *) =>
  forall C : *,
    ( (forall x y : A, R' x y -> R x y) ->
      (forall x : A, R' x x -> ⊥) -> C) ->
    C
```

gdzie \perp to forall $X : *$, X .

Jakie reguły formowania produktów są potrzebne (i do czego konkretnie?), aby przypisać temu termowi typ $T =$

```
forall A : *, (A -> A -> *) -> (A -> A -> *) -> *
```

Któremu systemowi lambda-kostki odpowiada taki zestaw reguł formowania produktów?

Czy gdyby $*$ zamienić na `Prop`, można by przypisać termowi t typ T w Coqu? A gdyby zamienić $*$ na `Set`?

Zadanie 3.

Dana jest definicja tzw. Σ -typu:

```
Inductive sig (A:s1) (P:A -> s2) : s3 :=
  exist : forall x:A, P x -> sig A P.
```

Section `Subset_projections`.

```
Variable A : s1.
```

```
Variable P : A -> s2.
```

```
Definition proj1 (e:sig A P) := match e with
  | exist a b => a
end.
```

```
Definition proj2 (e:sig A P) :=
  match e return P (proj1 e) with
  | exist a b => b
end.
```

End `Subset_projections`.

W powyższej definicji `s1`, `s2` i `s3` to sorty `Prop` lub `Set`. Dla jakich układów sortów `s1`, `s2`, `s3` powyższe definicje są poprawne?

Odpowiedź wpisz w tabelkę:

s1	s2	s3	proj1	proj2
Set	Set	Set	+/-	+/-
Set	Set	Prop		
Set	Prop	Set		
		⋮		
Prop	Prop	Prop		

Zadanie 4.

W niniejszym zadaniu porównamy równość Leibniza z równością Johna Majora, która umie porównywać termy różnych typów.

Dla potrzeb tego zadania przyjmujemy następujące definicje stałych związanych z równością Leibniza:

Parameter `eq` : forall A:Type, A -> A -> Type.

Parameter `eq_refl` : forall (A:Type) (a:A), (eq A a a).

Parameter `eq_elim` : forall (A:Type) (a:A) (phi:forall n, eq A a n -> Type),
 phi a (eq_refl A a) -> forall (b:A) (q: eq A a b), phi b q.

Parameter `eq_subst` : forall (A:Type) (a:A) (phi : A -> Type),
 phi a -> forall b:A, eq A a b -> phi b.

Parameter `eq_unique` : forall (A:Type) (a:A) (phi : eq A a a -> Type),
 phi (eq_refl A a) -> forall q:eq A a a, phi q.

Wśród powyższych definicji, `eq_elim` to zależny eliminator dla `eq` (gdyby `eq` było zdefiniowane indukcyjnie), `eq_subst` to eliminator niezależny, a `eq_unique` to stwierdzenie, że dowód równości może być tylko jeden.

- Pokazać, że stałą `eq_elim` można zdefiniować używając `eq_subst` i `eq_unique`.¹

Wskazówka: Definicja ta może wyglądać jakoś tak:

$$(\text{fun } A \text{ a } \phi \text{ X } b \text{ q} \Rightarrow (\text{eq_subst } _ _ _ _ _ _) \text{ q})$$

Rozważmy następujące stałe związane z równością Johna Majora:

Parameter `JMeq` : forall (A:Type), A -> forall (B:Type), B -> Type.

Parameter `JMeq_refl` : forall (A:Type) (a:A), (JMeq A a A a).

Axiom `JMeq_elim` :
 forall (A:Type)(a:A)(phi : forall a', JMeq A a A a' -> Type),
 phi a (JMeq_refl A a) -> forall a' (l:JMeq A a A a'), phi a' l.

- Używając `JMeq_elim`, zdefiniuj termy: `JMeq_subst`, `JMeq_unique` o podanych niżej typach:

Parameter `JMeq_subst` : forall (A:Type) (a:A) (phi : A -> Type),
 phi a -> forall (b:A) (q: JMeq A a A b), phi b.

Parameter `JMeq_unique` : forall (A:Type) (a:A) (phi : JMeq A a A a -> Type),
 phi (JMeq_refl A a) -> forall q:JMeq A a A a, phi q.

¹Hofmann i Streicher w 1994 pokazali, że `eq_unique` nie jest definiowalne z `eq_elim`

Wskazówka: Dowód pierwszego lematu jest łatwy. Dowód drugiego może wyglądać tak:

```
fun A a phi H l =>
  (JMeq_elim A a
    (fun (a':A) (l:JMeq A a A a') =>
      forall (e:JMeq A a A a),
        JMeq A a' A a ->
          JMeq (JMeq A a A a') l (JMeq A a A a) e ->
            phi e)
    _ a l _ _ _).
```

W kolejnych podpunktach pokażemy, że `JMeq` jest równoważna `eq` wyposażonej w `eq_unique`.

- używając stałych `JMeq`, `JMeq_refl`, `JMeq_subst` i `JMeq_unique` zdefiniuj `eq`, `eq_refl`, `eq_subst` i `eq_unique`.
- używając stałych `eq`, `refl`, `eq_subst` i `eq_unique` zdefiniuj `JMeq`, `JMeq_refl`, `JMeq_elim`.
Wskazówka: Trzeba użyć typu `cell`

```
Inductive cell : Type := C : forall A:Type, A -> cell.
```

a przy definicji `JMeq_elim` wykorzystać istnienie stałej `sproj`, t.ż.: w kontekście, gdzie $(A : \text{Type}) (a:A) (e : \text{eq cell } (C A a) (C A a'))$ zachodzi:

```
sproj e (eq_refl Type A) : eq A a a'
```