Extenzionális Martin-Löf Típuselmélet

1. Definíció

1.1. Függő Függvénytípus (Π-típus)

Típusalkotás (Formation) A szabály azt mondja meg, hogyan hozhatunk létre egy új függő függvénytípust.

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma, x : A \vdash B(x) \text{ type}}{\Gamma \vdash (\Pi x : A.B(x)) \text{ type}} \quad (\Pi\text{-form})$$

```
Pi : forall (A : Ty), (Tm A -> Ty) -> Ty;
```

Bevezetés (Introduction) λ-absztrakció:

$$\frac{\Gamma, x : A \vdash t(x) : B(x)}{\Gamma \vdash (\lambda x : A.t(x)) : (\Pi x : A.B(x))} \quad (\Pi\text{-intro})$$

```
lam : forall A (B : Tm A -> Ty),
  (forall (x : Tm A), Tm (B x)) -> Tm (Pi A B);
```

Kiküszöbölési szabály (Elimination)

$$\frac{\Gamma \vdash f : (\Pi x : A.B(x)) \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B(a)} \quad (\Pi\text{-elim})$$

```
app : forall A (B : Tm A -> Ty),
  Tm (Pi A B) -> forall (x : Tm A), Tm (B x);
```

Számítási szabály (β-redukció) Ez a szabály definiálja, hogyan viselkedik a függvényalkalmazás komputációs szempontból.

$$(\lambda x : A.t(x))(a) \equiv t(a) \quad (\beta\text{-redukció})$$

```
beta_Pi : forall A B f,
  app A B (lam A B f) = f;
```

Egyértelműségi szabály (η-konverzió)

$$f \equiv (\lambda x : A.f(x)) \quad (\eta\text{-konverzió})$$

```
eta_Pi : forall A B t,
  lam A B (app A B t) = t;
```

1.2. Azonosságtípus (Id-típus)

Típusalkotás (Formation) Létrehozhatunk egy típust, ami az azonosság bizonyítékait tartalmazza.

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma \vdash x : A \quad \Gamma \vdash y : A}{\Gamma \vdash \mathrm{Id}_A(x, y) \text{ type}} \quad (\text{Id-form})$$

```
Id : forall (A : Ty) (x y : Tm A), Ty;
```

Bevezetés (Introduction) Az azonosság bizonyításának módja a reflexivitás.

$$\frac{\Gamma \vdash x : A}{\Gamma \vdash \mathrm{refl}(x) : \mathrm{Id}_A(x, x)} \quad (\text{Id-intro})$$

```
refl : forall A (x y : Tm A),
  x = y -> Tm (Id A x y);
```

1.3. Kiküszöbölés (Elimination) / spéci: egyenlőség-tükrözés

Ez a szabály teszi a típuselméletet **extenzionálissá**.

$$\frac{\Gamma \vdash p : \mathrm{Id}_A(x, y)}{\Gamma \vdash x \equiv y : A} \quad (\text{Id-elim})$$

```
equality_reflection : forall A (x y : Tm A),
  Tm (Id A x y) -> x = y;
```

Számítási és egyértelműségi szabályok Ezek a szabályok biztosítják, hogy a `refl` és az `equality_reflection` egymás inverzei.

`beta_Id:`

$$\mathrm{refl}_{x \equiv y}(\mathrm{equality\_reflection}(t)) \equiv t$$

```
beta_Id : forall A (x y : Tm A) t,
  refl A x y (equality_reflection A x y t) = t;
```

```
eta_Id:
```

$$\text{equality\_reflection}(\text{refl}_{x\equiv y}(p)) \equiv p$$

```
eta_Id : forall A (x y : Tm A) p,
  equality_reflection A x y (refl A x y p) = p;
```

2. Coq mint az EMLTT direkt modellje

A Coq natív típuselmélete, kiegészítve két axiómával(!), egy természetes és direkt modelljét adja a fent definiált extenzionális Martin-Löf típuselméletnek (EMLTT).

Állítás. A Coq rendszerének megfelelő töredéke, kiegészítve a **funkcionális extenzionalitás** és a **bizonyítás-irrelevancia** axiómáival EMLTT.

Bizonyítás A modellt úgy hozzuk létre, hogy az EMLTT absztrakt fogalmait megfeleltetjük a Coq saját, konkrét konstrukcióinak:

- **Típusok** (`Ty`): Az elmélet típusait a Coq saját típusai, azaz a `Type` univerzumban élő elemek feleltetik meg.
- **Termek** (`Tm(A)`): Egy `A` típus termjei maguk a Coq `A` típusának elemei.
- **Pi-típus** (`Pi A B`): A függő függvénytípust a Coq beépített függő függvénytípusa, a `forall (x : A), B x` valósítja meg.
- **Id-típus** (`Id A x y`): Az azonosságtípust a Coq propozicionális egyenlősége, az `x = y` valósítja meg.
- **Lambda** (`lam f`): A λ-absztrakció a Coq-beli függvényekre az identitás-függvény.
- **Applikáció** (`app f x`): A függvényalkalmazás a Coq natív függvényalkalmazása.
- **Reflexivitás** (`refl H`): Az azonosság bevezető szabálya egy `x = y` bizonyításra az identitás-függvény.
- **Egyenlőség-reflexió** (`eq_refl p`): Az azonosság kivezető szabálya szintén az identitás-függvény a bizonyításokon.

Az alábbi Coq kódrészlet definiálja ezt a modellt:

```
Section Direct_model_of_EMLTT.

Require Import Coq.Logic.FunctionalExtensionality.
Require Import Coq.Logic.ProofIrrelevance.

Instance Coq_as_EMLTT_model : ExtensionalMartinLofTypeTheory.
Proof.
  apply mk_ExtensionalMartinLofTypeTheory with
    (Ty := Type)
    (Tm := fun T : Type => T)
    (Pi := (fun A B => forall (x : A), B x))
```

```
    (Id := (fun A x y => x = y))
    (lam := (fun A B f => f))
    (app := (fun A B f x => f x))
    (refl := (fun A x y H => H))
    (equality_reflection := (fun A x y p => p)).
  (* A bizonyítások innentől következnek *)
Defined.
```

A négy számítási/egyértelműségi szabály teljesül ezzel az interpretációval.

1-2. β-Pi és η-Pi szabályok. A `beta_Pi` szabály azt követeli meg, hogy `app (lam f) = f` legyen. Az interpretációnban ez a `(fun x => f x) = f` egyenlőséggé válik. Hasonlóan, az `eta_Pi` szabály (`lam (app t) = t`) szintén a `(fun x => t x) = t` egyenlőséget követeli meg. Mindkét állítás a **funkcionális extenzionalitás** axiómájának közvetlen következménye, ami kimondja, hogy két függvény akkor és csak akkor egyenlő, ha minden argumentumra azonos eredményt adnak.

```
Axiom functional_extensionality_dep : forall {A} {B : A -> Type},
  forall (f g : forall x : A, B x),
  (forall x, f x = g x) -> f = g.
```

Ez az axióma független a Coq levezetési rendszerétől, azaz nem vezethető le belőle. (És a EMLTT-ra lefordított alakja is független az EMLTT-től: van olyan modell, amiben igaz és van olyan modell, amiben hamis.)

```
(* beta_Pi: app (lam f) = f  <-->  (fun x => f x) = f *)
intros A B f. extensionality x. reflexivity.

(* eta_Pi: lam (app t) = t  <-->  lam (fun x => t x) = t *)
intros A B t. extensionality x. reflexivity.
```

3-4. β-Id és η-Id szabályok. A `beta_Id` szabály szerint `refl (equality_reflection t) = t`, ahol `t` egy bizonyítása az `x = y` azonosságnak. Az `eta_Id` szabály pedig azt, hogy `equality_reflection (refl p) = p`. Mivel a `refl` és az `equality_reflection` is az identitás-függvényként van definiálva a bizonyításokon, mindkét egyenlet egyszerűen `t = t` illetve `p = p` alakú lesz. Ezek az egyenlőségek a **bizonyítás-irrelevancia** axiómájából következnek. Ez az axióma kimondja, hogy egy állítás bármely két bizonyítása azonos. Mivel az egyenlet két oldalán álló termek ugyanannak az `x = y` propozíciónak a bizonyításai, ezért egyenlőnek kell lenniük.

```
(* beta_Id: refl (eq_refl t) = t *)
intros A x y t. apply proof_irrelevance.

(* eta_Id: eq_refl (refl p) = p *)
intros A x y p. apply proof_irrelevance.
```