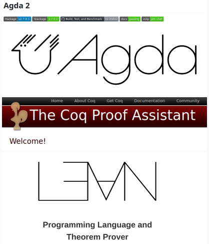
Hogyan működnek a bizonyításasszisztensek?

Molnár Zoltán Gábor (BME, Algebra és Geometria Tsz.)

2025. Február 25.

Miért érdekesek a bizonyításasszisztensek?

- Típus- és bizonyításelmélészeknek: Agda, Coq
- Gyakorló matematikusok számára: Lean (Mathlib), Coq
- Alapjuk: függő típuselmélet (Dependent Type Theory, DTT)



Miért „ismeretlen” nyelvek a proverek?

- A legközelebbi ismert: Haskell (tisztán funcionális, típusos, polimorf)
- (Továbbiak: OCaml, Scala, Rust, ... (nem tisztán funkcionális))
- Hasonló: TypeScript (JS + Types), Matlab, Python + mypy (nem tisztán funkcionális)

Példa függvénydefinícióra TS-ben (típusfüggés)

```
// TypeScript
function
    getLength<T>([head, ...tail]: [T, ...T[]] | []): number
{
    return head ? 1 + getLength(tail) : 0;
}
```

Amiben utolérhetetlenek?

a típusok termfüggése bizonyítás

```
(* Coq *)
Fixpoint length {A : Type} (l : list A) : nat :=
  match l with
  | [] => 0
  | a :: t => 1 + length t
  end.

Lemma length_cons : forall (A : Type) (a : A) (l : list A),
  length (a :: l) = 1 + length l.
Proof.
  intuition.
Qed.
```

Dependent Types

$$\frac{\vdash A : \mathrm{Type} \quad x : A \vdash B(x) : \mathrm{Type} \quad x : A \vdash b(x) : B(x)}{\vdash \lambda(x : A).b(x) : \prod_{x:A} B(x)}$$

$$\frac{\vdash A : \mathrm{Type}}{x : A \vdash \mathrm{refl}_A\, x : x = x}$$

Egyszerű típuselmélet (STT)

STT: Programozási és logikai megközelítés

- Minden **programnyelv,** amely tartalmaz funkcionális részt, rendelkezik STT elemekkel...
- ... és a legtöbb esetben ez elég is.
- Logikai megközelítés: STT az implikációs (ha-akkor) logika **természetes levezetési rendszere,** ...
- ... de csak a propozícionális és véges dimenziós logikára kiterjeszthető.

STT: a szabályok

- **Típusok:**

$$\frac{}{\iota : \mathrm{Type}} \qquad \frac{A : \mathrm{Type} \qquad B : \mathrm{Type}}{A \to B : \mathrm{Type}}$$

- Változódeklarációk, kontextusok:

  $\Gamma = [A, B, ...]$ (értsd: $\Gamma = (x : A, y : B, ...)$)

- **Operációk,** levezetési szabályok:

$$\frac{A, \Gamma \vdash t : B}{\Gamma \vdash \lambda A t : A \to B} \qquad (\text{értsd } \frac{x : A, \Gamma \vdash t : B}{\Gamma \vdash \lambda(x : A).t : A \to B})$$

$$\frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash s : A}{\Gamma \vdash \mathrm{app}\ t\ s : B} \qquad (\text{értsd } \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash s : A}{\Gamma \vdash t(s) : B})$$

- Strukturális szabályok

$$\frac{}{A, \Gamma \vdash \mathrm{hyp}\, 0 : A} \qquad \frac{\Gamma \vdash \mathrm{hyp}\ i : B}{A, \Gamma \vdash \mathrm{hyp}\,(i + 1) : B}$$

Beta redukció és helyettesítés

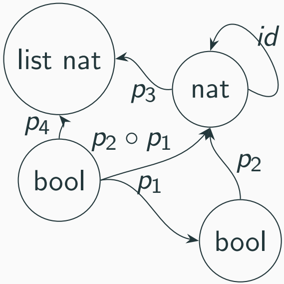
- Mint programnyelv, az STT rendelkezik a programfuttatás fogalmával.
- Logikában, ez az a bizonyítások redukciója, a vargabetűk levágása.
- **Egyenlőségi** vagy konverziós szabályok:

$$\beta : \quad \mathrm{app}\,(\lambda\, A\; t)\; s = t[s], \qquad \lambda\, A\,(\mathrm{app}\; t\; \mathrm{hyp}\; 0) = t \quad : \eta$$

ahol .[.] a legkülső szabad változóba való behelyettesítés.

STT modellje: CCC (és CwF)

- A családos kategóriák **CwF** modell azon alapul, hogy a kontextusok (objektumok), helyettesítések (mofizmusok), termek-típusok (kontravariáns funktor a halmazcsaládok kategóriájába) mind alapfogalmak.
- A kartéziusi zárt kategória **CCC** modell szerint a típusok az objektumok, a morfizmusok a programok.
  $A \wedge B \mapsto A \times B \quad (A \to B) \mapsto B^A \qquad ((AB)^C = A^C B^C)$



Coq implementációk

Coq bizonyítások: CCC helyessége és teljessége

- A proof irrelevance feltételezésével a CCC helyes és teljes szemantikája STT-nek.
- Nincs szükség a helyettesítésre és a beta-eta redukció csak néhány esete kell.

Lean példák

Összegzés

Összegzés

- STT alapfogalmai és megközelítései.
- CCC mint modell.
- Beta redukció és helyettesítés.
- Coq implementációk és részeredmények.
- Lean alkalmazása matematikai bizonyításokhoz.