Systémy typovaných lambda kalkulů (část první)

LKFP přednáška 11, jaro 2024/25

Jan Laštovička

29. dubna 2025

1 Obecný rámec

Množina *nepravých výrazů* $\mathcal{T}$ je nejmenší množina, která splňuje následující podmínky.

1. $\mathcal{T}$ obsahuje spočetné množství proměnných $v_1, v_2, \ldots$;
2. $\mathcal{T}$ obsahuje konstanty $*$ a $\Box$;
3. jestliže $A, B \in \mathcal{T}$, pak $(AB) \in \mathcal{T}$;
4. jestliže x je proměnná a $A, B \in \mathcal{T}$, pak $(\lambda x : A.B) \in \mathcal{T}$;
5. jestliže x je proměnná a $A, B \in \mathcal{T}$, pak $(\Pi x : A.B) \in \mathcal{T}$.

Nepravé výrazy vzniklé pátým bodem se nazývají *součiny*. Přijímáme zaběhlé konvence o vynechávání závorek z čistého lambda kalkulu. Budeme používat $A, B, C, \alpha, \beta, \gamma, a, b, c, \ldots$ pro označování nepravých výrazů a $x, y, z, \ldots$ pro označování proměnných.

Stále používáme osnovu redukce

$$\beta = \{((\lambda x : A.B)C, B[x := C]) \mid A, B, C \in \mathcal{T} \text{ a } x \text{ je proměnná}\}.$$

Výroky mají tvar $A : B$, kde $A, B \in \mathcal{T}$. Nepravý výraz A se nazývá *subjekt* a B *predikát* výroku $A : B$. *Deklarace* mají tvar $x : A$, kde x je proměnná a $A \in \mathcal{T}$. *Nepravý kontext* je konečná posloupnost deklarací, jejichž subjekty jsou po dvou různé. Pokud $\Gamma = (x_1 : A_1 \ldots, x_n : A_n)$ je kontext a $y : B$ deklarace, pak

$$\Gamma, y : B = (x_1 : A_1 \ldots, x_n : A_n, y : B)$$

značí přidání deklarace do kontextu. Prázdný kontext () většinou nepíšeme. Fakt, že proměnná x není deklarovaná v nepravém kontextu Γ, zapisujeme $x \notin \Gamma$.

Níže uvedená pravidla definují pojem

$$\Gamma \vdash A : B$$

tvrdící, že výrok $A : B$ je odvozen z nepravého kontextu Γ; v takovém případě nazýváme A a B (*pravými*) *výrazy* a Γ (*pravým*) *kontextem*.

Konstanty $*$ a $\Box$ nazýváme *druhy*. Nechť $\mathrm{S} = \{*, \Box\}$. V níže uvedených pravidlech s, s_1 a s_2 označují nějaké prvky S. Začneme základními pravidly.

1. Axiom:

$$() \vdash * : \Box$$

2. Startovací pravidlo:

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \quad x \notin \Gamma$$

3. Oslabovací pravidlo:

$$\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B} \quad x \notin \Gamma$$

4. Pravidlo aplikace:

$$\frac{\Gamma \vdash F : (\Pi x : A.B) \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]}$$

5. Pravidlo abstrakce:

$$\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x : A.B) : s}{\Gamma \vdash (\lambda x : A.b) : (\Pi x : A.B)}$$

6. Pravidlo záměny (konverze):

$$\frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_\beta B'}{\Gamma \vdash A : B'}$$

Zvláštní pravidlo (s_1, s_2):

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x : A.B) : s_2}$$

Pokud $\Gamma \vdash A : \Box$, pak se A nazývá *formou* v kontextu Γ; pokud navíc $\Gamma \vdash B : A$, pak se B nazývá *konstruktorem formy* A v kontextu Γ. Pokud $\Gamma \vdash A : *$, pak se A nazývá *typem* v kontextu Γ; pokud navíc $\Gamma \vdash B : A$, pak říkáme, že B *je výraz typu* A v kontextu Γ.

Definujeme:

$A \to B \equiv \Pi x : A.B$, kde proměnná x se nevyskytuje v A ani v B.

Zápisem $\Gamma \vdash A : B : C$ myslíme konjunkci $\Gamma \vdash A : B$ a $\Gamma \vdash B : C$.

2 Systém $\lambda\!\rightarrow$

Systém používá základní pravidla a zvláštní pravidlo $(*, *)$. Můžeme odvodit:

$$\dfrac{\dfrac{}{\vdash * : \Box}}{A : * \vdash A : *} \qquad \dfrac{\dfrac{\dfrac{}{\vdash * : \Box}}{A : * \vdash A : *} \qquad \dfrac{\dfrac{}{\vdash * : \Box}}{A : * \vdash A : *}}{A : *, x : A \vdash A : *}}{A : * \vdash (\Pi x : A.A) : *}$$

Tedy $A : * \vdash (A \to A) : *$. Odvodili jsme, že pokud je A typ, pak $A \to A$ je typ. Dále odvodíme:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash * : \Box}}{A : * \vdash A : *}}{A : *, a : A \vdash a : A} \qquad \dfrac{\vdots}{A : * \vdash (\Pi x : A.A) : *}}{A : * \vdash (\lambda a : A.a) : (\Pi x : A.A)}$$

Odvodili jsme $A : * \vdash (\lambda a : A.a) : (A \to A)$. Tedy pokud je A typ, pak $(\lambda a : A.a)$ je abstrakce typu $(A \to A)$. Právě odvozenou abstrakci můžeme aplikovat na vhodný argument:

$$\dfrac{\dfrac{\vdots}{A : *, b : A \vdash (\Pi x : A.A)} \qquad \dfrac{\vdots}{A : *, b : A \vdash b : A}}{A : *, b : A \vdash ((\lambda a : A.a)b) : A}$$

Tedy pokud je A typ a b proměnná typu A, pak $(\lambda a : A.a)b$ je výraz typu A. Samozřejmě platí:

$$(\lambda a : A.a)b \to_\beta b$$

Dále lze například odvodit:

$$A : *, B : *, c : A, b : B \vdash ((\lambda a : A.b)c) : B;$$
$$A : *, B : * \vdash (\lambda a : A \lambda b : B.a) : (A \to (B \to A)).$$

Systém $\lambda\!\rightarrow$ odpovídá jednoduše typovanému lambda kalkulu.

3 Agda

Budeme používat programovací jazyk Agda[1], který vychází z Haskellu. Pro práci s jazykem použijeme rozšíření `agda-mode`[2] pro Visual Studio Code.

Výraz $\texttt{Set}_0$ zapisuje druh $*$ a výraz $\texttt{Set}_1$ druh $\Box$. Pro vložení $_0$ stačí napsat `\_0`, podobně pro $_1$. Programem mimo jiné definujeme kontext Γ. Definice:

[1] `https://wiki.portal.chalmers.se/agda/pmwiki.php`
[2] `https://marketplace.visualstudio.com/items/?itemName=banacorn.agda-mode`

```
data A : : Set₀ where
  a₁ : A₁
  ⋮
  aₙ : Aₙ
```

přidá do kontextu deklarace $A : *, a_1 : A_1, \ldots, a_n : A_n$. Například:

```
data N : Set₀ where
  zero : N
  succ : N → N
```

určí kontext $N : *, \texttt{zero} : N, \texttt{succ} : N \to N$. Symbol $\to$ vložíme napsáním `\to`. Definice:

```
x : A
x = b
```

ověří, že $\Gamma \vdash b : A$ a přidá do kontextu deklaraci $x : A$. Déle se výraz b pojmenuje x. .

Například:

```
x1 : N
x1 = (succ zero)
```

ověří $N : *, \texttt{zero} : N, \texttt{succ} : N \to N \vdash (\texttt{succ zero}) : N$.

Kompilací (stiskem kombinace `Ctrl+C Ctrl+L`) výraz zkompilujeme. Kombinací kláves `Ctrl+C Ctrl+N` můžeme zadat výraz a systém spočítá jeho normální formu. Například můžeme ověřit, že `x1` $\twoheadrightarrow$ (`succ zero`).

Abstrakci ($\lambda x : A.B$) zapíšeme výrazem `(λ x → B)`. Výraz A se snaží Agda odvodit. Obecně může při odvozování A selhat, protože se jedná o nerozhodnutelný problém. Symbol λ vložíme sekvencí `\lambda`. Například:

```
id : N → N
id = λ n → n
```

Jiná definice téhož:

```
id : N → N
id n = n
```

Jména stále můžeme definovat rozborem případů. Například:

```
+ : N →  N →  N
+ zero n = n
+ (succ m) n = succ (+ m n)
```

Normální forma výrazu `+ (succ zero) (succ zero)` je `(succ (succ zero))`. Přidáním deklarace:

```
{-# BUILTIN NATURAL N #-}
```

umožníme vkládání čísel v desítkové soustavě. Například `+ 1 1` je rovno výrazu `+ (succ zero) (succ zero)`.

Kombinátory `S` a `K` pro čísla:

```
K : N → N → N
K m n = m

S : (N → N → N) → (N → N) → N → N
S f g x = f x (g x)
```

Například:

$$\texttt{S + succ 3} \twoheadrightarrow \texttt{7}$$

3.1 Systém $\lambda 2$

Systém $\lambda 2$ používá základní pravidla a zvláštní pravidla $(*, *)$ a $(\Box, *)$. Můžeme v něm odvodit:

$$\dfrac{\vdash * : \Box \qquad \dfrac{\vdots}{\alpha : * \vdash (\alpha \to \alpha) : *}}{\vdash (\Pi\alpha : *.(\alpha \to \alpha)) : *}$$

Vynechané části důkazu již byly dokázány výše. Ukázali jsme, že $\vdash (\Pi\alpha : *.(\alpha \to \alpha))$ je typ. Najdeme hodnotu zadaného typu:

$$\dfrac{\dfrac{\vdots}{\alpha : * \vdash (\lambda a : \alpha.a) : (\alpha \to \alpha)} \qquad \dfrac{\vdots}{\vdash (\Pi\alpha : *.(\alpha \to \alpha)) : *}}{\vdash (\lambda\alpha : * \lambda a : \alpha.a) : (\Pi\alpha : *.(\alpha \to \alpha)) : *}$$

V Agdě typ $\Pi x : A.B$ zapíšeme výrazem `((`x `:` A`)` $\to$ B`)`. Předchozí výraz můžeme pojmenovat:

```
id : (α : Set₀) → (α → α)
id = λ α a → a
```

Písmeno α vložíme sekvencí `\alpha`.

Dále lze odvodit:

$$A : * \vdash (\lambda\alpha : * \lambda a : \alpha.a)A : (A \to A);$$
$$A : *, b : A \vdash (\lambda\alpha : * \lambda a : \alpha.a)Ab : A.$$

Samozřejmě platí:

$$(\lambda\alpha : *\lambda a : \alpha.a)Ab \to_\beta (\lambda a : A.a)b \to_\beta b.$$

Také lze odvodit $\vdash (\Pi\alpha : *.\alpha) : *$. Výraz si označíme $\bot \equiv (\Pi\alpha : *.\alpha)$. Neexistuje výraz A tak, aby $\vdash A : \bot$.

Definujeme kombinátory:

$$\mathrm{K} \equiv \lambda\alpha : *\lambda\beta : *\lambda x : \alpha\lambda y : \beta.x;$$
$$\mathrm{S} \equiv \lambda\alpha : *\lambda\beta : *\lambda\gamma : *\lambda f : (\alpha \to \beta \to \gamma)\lambda g : (\alpha \to \beta)\lambda z : \alpha.fz(gz).$$

Platí:

$$\vdash \mathrm{K} : (\Pi\alpha : *\Pi\beta : *.\alpha \to \beta \to \alpha);$$
$$\vdash \mathrm{S} : (\Pi\alpha : *\Pi\beta : *\Pi\gamma : *.((\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma)).$$

Ověření typů kombinátorů v Agdě:

```
K : (α : Set₀) → (β : Set₀) → (α → β → α)
K α β a b = a

S : (α : Set₀) → (β : Set₀) → (γ : Set₀) →
    (α → β → γ) → (α → β) → α → γ
S α β γ f g z = f x (g z)
```

Například výraz:

```
S N N N + succ 5
```

má normální formu `11`.