Funkcionální programování

Typovaný lambda kalkulus a Hindley–Milnerův typový systém

Petr Pudlák

5. října 2011

Osnova

Motivace

Chceme se dopředu dozvědět něco o lambda termu (o programu).
Například:

1. Zda jsou zabudované (či odvozené) funkce použity korektně. Například, zda neaplikujeme $+$ na dvě funkce.
2. Jaký bude výsledek programu (pokud program je normalizující)? Například, bude to číslo, nebo funkce?
3. Je program normalizující? (Některé typové systémy to dovedou rozhodnout.)

Typované lambda kalkuly

- Kromě termů konstruujeme další syntaktické objekty *typy*.
- Definujeme relaci mezi termy a typy: „term M je typu τ".
  Značíme obvykle $M : \tau$.
- Relaci „:" definujeme pomocí odvozovacích pravidel.
- Rozlišujeme 2 základní přístupy:

  à la Church: termy obsahují explicitní anotace, kterými přiřazujeme typy proměnným v lambda abstrakcích. Podle anotací pak odvozujeme typ termu.

  à la Curry: pro daný term (bez anotací) odvozujeme jeho (možný) typ.

Osnova

Systém F

Další názvy:

- $\lambda 2$
- (Girard–Reynolds) polymorphic lambda calculus
- the second-order lambda calculus

Typy v systému F

Definition (Typy systému F)

Buď $\mathbb{V}$ nekonečná spočetná množina typových proměnných.
Množina typů $\mathbb{T}$ systému F je definována induktivně:

$$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \to \mathbb{T} \mid \forall \mathbb{V} \mathbb{T}$$

- „$\to$" konstruuje funkční typy – funkce z prvního typu do druhého.
- „$\forall$" konstruuje polymorfní typy.
- Konvence:
  - Typové proměnné označujme řeckými písmeny ze začátku abecedy (α, β atd.).
  - Typy označujme řeckými písmeny z prostředku abecedy (τ, σ atd.).
  - „$\to$" asociuje doprava: $\sigma_1 \to \sigma_2 \to \sigma_3$ píšeme místo $\sigma_1 \to (\sigma_2 \to \sigma_3)$
  - „$\forall$" asociuje doprava: $\forall\alpha_1 \ldots \forall\alpha_n.\sigma$ píšeme místo $(\forall\alpha_1(\forall\alpha_2 \ldots (\forall\alpha_n(\sigma)) \ldots))$.

Tvrzení v typovaných lambda kalkulech

Definition (tvrzení)

1. *Tvrzení* říká, že daný term $M \in \Lambda$ je daného typu $\sigma \in \mathbb{T}$.
   Značení $M : \sigma$.
   Typ σ je *predikátem* tvrzení a term M je *subjektem* tvrzení.
2. *Deklarace* je tvrzení, ve kterém je subjektem proměnná.
3. *Báze* (nebo též *kontext*) je množina deklarací, které mají za subjekty různé proměnné. Značíme obvykle velkými řeckými písmeny (Γ, Δ).

Poznámka

V Haskellu relaci „:" zapisujeme pomocí znaku dvou dvojteček :: .
Tvrzení $M : \sigma$ *tedy zapíšeme jako* $M :: \sigma$.

Přiřazení typů v systému F

Definition

Přiřazení typů v systému F je definováno následujícím systémem přirozené dedukce:

Pravidla (axiomy) pro přiřazení typů v systému F

$$\text{(startovací pravidlo)}\ \frac{(x:\sigma)\in\Gamma}{\Gamma\vdash x:\sigma}$$

$$\text{(eliminace }\rightarrow\text{)}\ \frac{\Gamma\vdash M:(\sigma\rightarrow\tau)\qquad\Gamma\vdash N:\sigma}{\Gamma\vdash(MN):\tau}$$

$$\text{(zavedení }\rightarrow\text{)}\ \frac{\Gamma,x:\sigma\vdash M:\tau}{\Gamma\vdash(\lambda x.M):(\sigma\rightarrow\tau)}$$

$$\text{(eliminace }\forall\text{)}\ \frac{\Gamma\vdash M:(\forall\alpha.\sigma)}{\Gamma\vdash M:(\sigma[\alpha:=\tau])}$$

$$\text{(zavedení }\forall\text{)}\ \frac{\Gamma\vdash M:\sigma}{\Gamma\vdash M:(\forall\alpha.\sigma)}\ ,\ \alpha\notin FV(\Gamma)$$

Věta o substituci v systému F

Lemma (o substituci v systému F)

1. $\Gamma \vdash M : \sigma$ *pak* $\Gamma[\alpha := \tau] \vdash M : \sigma[\alpha := \tau]$.
2. *Pokud* $\Gamma, x : \sigma \vdash M : \tau$ *a* $\Gamma \vdash N : \sigma$ *pak* $\Gamma \vdash M[x := N] : \tau$.

Důkaz.

[Barendregt92]. □

Věta o redukci subjektů v systému F

Theorem (o redukci subjektů v systému F)

Buď $M \twoheadrightarrow_\beta M'$. Pak v systému F platí, že pokud $\Gamma \vdash M : \sigma$ pak $\Gamma \vdash M' : \sigma$.

Důkaz.

[Barendregt92]. □

Silná normalizace v systému F

Theorem (o silné normalizaci pro systém F)

Jestliže v systému F $\Gamma \vdash M : \sigma$ *pak M je silně normalizující.*

Důkaz.

[Barendregt92]. □

Důsledek

- *Systém F není turingovsky úplný, protože obsahuje jen totální funkce.*
- *Systém F neobsahuje obecnou rekurzi.*
  Pokud nějaký systém obsahuje obecnou rekurzi (např. je v něm typovatelný kombinátor Y) tak obsahuje funkce, které nejsou totální (například YI nemá nf). Takový systém proto není ani silně ani slabě normalizující.

Kontrola typu, typovatelnost, obydlenost typu

Definition

Kontrola typu (type checking): Dáno M a σ, platí $\vdash M : \sigma$?
Značení $M : \sigma?$.

Typovatelnost (typability): Dáno M, existuje σ taková, že $\vdash M : \sigma$?
Značení $M :?$.

Obydlenost (inhabitation): Dáno σ, existuje M takový, že $\vdash M : \sigma$?
Značení $? : \sigma$.

Typy a nerozhodnutelnost I

Theorem (Joe B. Wells)

V systému F nelze rozhodnout:

1. *kontrolu typu,*
2. *typovatelnost,*
3. *obydlenost.*

Typy a nerozhodnutelnost II

Důkaz.

- Důkaz, že v F je typovatelnost ekvivalentní kontrole typu byl znám už dříve, viz. [**Barendregt92**].
- Důkaz nerozhodnutelnosti obydlenosti viz. [**Barendregt92**].
- Důkaz, že typovatelnost v F je nerozhodnutelná viz. [**Wells-TypabilityL2**].

□

Poznámka

Nerozhodnutelnost obydlenosti a ekvivalence typovatelnosti a kontroly typu byly známy již poměrně dlouho, ale chyběl stále důkaz (ne)rozhodnutelnosti typovatelnosti a kontroly typu F. Tyto dva problémy byly proto nazývány „embarrassing open problems", než je Wells vyřešil.

Osnova

Rank-k polymorfismus

- V systému F se mohou kvantifikátory typů objevovat libovolně v typech.
- Ukázalo se, že to ovlivňuje rozhodnutelnost typovatelnosti.

Rank-k polymorfismus

- V systému F se mohou kvantifikátory typů objevovat libovolně v typech.
- Ukázalo se, že to ovlivňuje rozhodnutelnost typovatelnosti.

Definition

Buď dáno $k \in \mathbb{N}$. Pak typ je *ranku* k (často psáno *rank-k*) pokud se žádný jeho kvantifikátor neobjevuje nalevo od více než $k - 1$ šipek (nakreslíme-li typ jako strom).

Rank-k polymorfismus

Example

$\forall\alpha\beta.\alpha \to \beta \to \alpha$ je rank-1.
$(\forall\alpha.\alpha \to \alpha) \to (\forall\beta.\beta \to \beta)$ je rank-2.

Example

- Typ $\forall\alpha.\alpha \to (\forall\beta.\beta)$ (rank-1) není obydlený totální funkcí; (je ekvivalentní typu $\forall\alpha\beta.\alpha \to \beta$).
- Typ $(\forall\alpha.\alpha) \to (\forall\beta.\beta)$ (rank-2) je obydlený totální funkcí $\lambda x.x$.

Rank-k polymorfismus

Odvoditelnost typů

- Pro rank-1 typy lze odvozovat typy (Hindley–Milnerův algoritmus [**damas1982principal**]).
- Pro rank-2 lze také ještě odvozovat typy [**kfoury1992type**, **kfoury1994direct**].
- Pro rank-k, $k > 2$ je odvoditelnost nerozhodnutelná.

Osnova

Jazyk Hindley–Milnerova typového systému

Termy:

$$e := x \mid e_1\, e_2 \mid \lambda x.e \mid \text{let } x = e_1 \text{ in } e_2$$

Typy:

$$\tau := \alpha \mid \iota \mid \tau \to \tau$$

Typová schémata:

$$\sigma := \tau \mid \forall \alpha.\sigma$$

Konvence značení

Budeme používat následující konvenci značení:

- x pro lambda proměnné,
- e pro lambda termy,
- α pro typové proměnné,
- τ pro typy,
- ι pro primitivní typy (jako např. *Int* apod.),
- σ pro typová schémata.
- Sekvenci objektů značme pro zjednodušení zápisu čarou, například: $\lambda\overline{x}.M$ místo $\lambda x_1, \ldots, x_n.M$, $\forall\overline{\alpha}.\tau$ místo $\forall\alpha_1 \ldots \forall\alpha_n.\tau$ atd. Sekvence může být pokaždé jiná a jinak dlouhá.

Volné proměnné

$$
\begin{aligned}
\mathsf{FTV}(\alpha) &= \{\alpha\} \\
\mathsf{FTV}(\tau \to \tau') &= \mathsf{FTV}(\tau) \cup \mathsf{FTV}(\tau') \\
\mathsf{FTV}(\forall \alpha \tau) &= \mathsf{FTV}(\tau) \setminus \{\alpha\}
\end{aligned}
$$

Uspořádání na typových schématech

Definition

$$\frac{\beta_i \notin \mathsf{FTV}(\forall\overline{\alpha}.\tau) \qquad \tau' = [\overline{\tau}/\overline{\alpha}]\tau}{\forall\overline{\alpha}.\tau > \forall\overline{\beta}.\tau'}$$

Slovy:

- Dosadíme za kvantifikované proměnné nějaké typy $\tau_1, \ldots, \tau_n$.
- Kvantifikujeme libovolně ty proměnné, které nebyly volné v původním typovém schématu $\forall\overline{\alpha}.\tau$.

Uspořádání na typových schématech

Definition

$$\frac{\beta_i \notin \mathrm{FTV}(\forall\overline{\alpha}.\tau) \qquad \tau' = [\overline{\tau}/\overline{\alpha}]\tau}{\forall\overline{\alpha}.\tau > \forall\overline{\beta}.\tau'}$$

Slovy:

- Dosadíme za kvantifikované proměnné nějaké typy $\tau_1, \ldots, \tau_n$.
- Kvantifikujeme libovolně ty proměnné, které nebyly volné v původním typovém schématu $\forall\overline{\alpha}.\tau$.

Pozorování:

- Relace $>$ je reflexivní a tranzitivní, je to tedy uspořádání
- Intuitivně $\sigma > \sigma'$ znamená, že σ je obecnější typ než σ'.
- Je-li $\sigma > \sigma'$ pak $\mathrm{FTV}(\sigma) \subseteq \mathrm{FTV}(\sigma')$.

Example

$$\forall\alpha.\alpha \to \alpha \;>\; \forall\beta.(\beta \to \beta) \to (\beta \to \beta) \;>\; (\gamma \to \gamma) \to (\gamma \to \gamma)$$

Substituce

- Substituce na typech je definována obvyklým způsobem.
- U typových schémat substituce pracuje pouze na volných proměnných, vázané proměnné nechává.
- Značení $S = [\tau_1/\alpha_1 \ldots, \tau_n/\alpha_n]$, nebo jen $S = [\overline{\tau}/\overline{\alpha}]$.

Substituce

- Substituce na typech je definována obvyklým způsobem.
- U typových schémat substituce pracuje pouze na volných proměnných, vázané proměnné nechává.
- Značení $S = [\tau_1/\alpha_1 \ldots, \tau_n/\alpha_n]$, nebo jen $S = [\overline{\tau}/\overline{\alpha}]$.
- Substituce si všímá jen volných proměnných, a
- „$>$” si všímá jen vázaných proměnných.

Substituce

- Substituce na typech je definována obvyklým způsobem.
- U typových schémat substituce pracuje pouze na volných proměnných, vázané proměnné nechává.
- Značení $S = [\tau_1/\alpha_1 \ldots, \tau_n/\alpha_n]$, nebo jen $S = [\overline{\tau}/\overline{\alpha}]$.
- Substituce si všímá jen volných proměnných, a
- „$>$" si všímá jen vázaných proměnných.
- Platí proto, že $\sigma > \sigma'$ pak $S\sigma > S\sigma'$.

Kontexty

Definition

Buď $\Gamma = \langle x_1 : \sigma_1, \ldots, x_n : \sigma_n \rangle$. Definujme

- $\mathrm{FTV}(\Gamma) := \bigcup_i \mathrm{FTV}(\sigma_i)$
- $S\Gamma := \langle x_1 : S\sigma_1, \ldots, x_n : S\sigma_n \rangle$
- $\overline{\Gamma}(\tau) := \forall \overline{\alpha}.\tau$, kde $\alpha = \mathrm{FTV}(\tau) \setminus FTP(\Gamma)$.
  Intuitivně, kvantifikujeme všechny volné proměnné, které nejsou vázány kontextem.
- Γ_x buď kontext Γ, z kterého odebereme deklarace týkající proměnné x.

Hindley-Milnerův typový systém

Definition

V Hindley-Milnerově typovém systému píšeme, že $\Gamma \vdash e : \sigma$ jestliže to lze odvodit následujícími pravidly:

$$\text{TAUT}\ \frac{}{\Gamma \vdash x : \sigma}\ (x : \sigma \in \Gamma)$$

$$\text{COMB}\ \frac{\Gamma \vdash e : \tau' \to \tau \qquad \Gamma \vdash e' : \tau'}{\Gamma \vdash (ee') : \tau}$$

$$\text{INST}\ \frac{\Gamma \vdash e : \sigma}{\Gamma \vdash e : \sigma'}\ (\sigma > \sigma')$$

$$\text{ABS}\ \frac{\Gamma_x, x : \tau' \vdash e : \tau}{\Gamma \vdash (\lambda x.e) : \tau' \to \tau}$$

$$\text{GEN}\ \frac{\Gamma \vdash e : \sigma}{\Gamma \vdash e : \forall \alpha.\sigma}\ (\alpha \notin \text{FTV}(\Gamma))$$

$$\text{LET}\ \frac{\Gamma \vdash e : \sigma \qquad \Gamma_x, x : \sigma \vdash e' : \tau}{\Gamma \vdash (\text{let}\, x = e\, \text{in}\, e') : \tau}$$

Hindley-Milnerův algoritmus

Dáno Γ a e, algoritmus W hledá substituci S a typ τ takové, že $S\Gamma \vdash e : \tau$. Dále platí:

Theorem (Korektnost W)

Jestliže $W(\Gamma, e)$ najde (S, τ) pak existuje odvození $S\Gamma \vdash e : \tau$.

Theorem (Úplnost W)

Mějme Γ, e, a buď Γ' instance Γ a σ takové, že $\Gamma' \vdash e : \sigma$. Pak:

1. *$W(\Gamma, e)$ uspěje; buď $W(\Gamma, e) = (S, \tau)$; a*
2. *existuje substituce R taková, že $\Gamma' = RS\Gamma$ a $R\overline{S\Gamma}(\tau) > \sigma$.*

Takový typ τ nazýváme principiální typ termu e.

Důkaz.

Viz. originální článek [**damas1982principal**], nebo http://www.cs.ucla.edu/~jeff/docs/hmproof.pdf. □

Unifikace

Definition

- *Unifikátor* typů τ a τ' je substituce S taková, že $S\tau = S\tau'$.
- *Nejobecnější unifikátor* typů τ a τ' je unifikátor S takový, že pokud pro substituci R platí, že $R\tau = R\tau'$, existuje T taková, že $R = T \circ S$.

Unifikace

Definition

- *Unifikátor* typů τ a τ' je substituce S taková, že $S\tau = S\tau'$.
- *Nejobecnější unifikátor* typů τ a τ' je unifikátor S takový, že pokud pro substituci R platí, že $R\tau = R\tau'$, existuje T taková, že $R = T \circ S$.

Theorem (Robinson 1965)

Existuje-li unifikátor dvou typů τ a τ' pak existuje jejich nejobecnější unifikátor (zn. $\mathcal{U}(\tau, \tau')$) a lze ho efektivně počítat.

Důkaz.

Viz. [**robinson1965machine**]. (Udělat příklad.) □

Hindley-Milnerův algoritmus

$W(\Gamma, e) = (S, \tau)$, kde:

- $e = x$ a $x : \forall\overline{\alpha}.\tau' \in \Gamma$, pak S je identická substituce a $\tau = [\overline{\beta}/\overline{\alpha}]\tau'$, kde β_i jsou nové proměnné.
- $e = e_1\, e_2$: buď
  $W(\Gamma, e_1) = (S_1, \tau_1)$,
  $W(S_1\Gamma, e_2) = (S_2, \tau_2)$ a
  $V = \mathcal{U}(S_2\tau_1, \tau_2 \to \beta)$, kde β je nová proměnná.
  Pak $S = V \circ S_2 \circ S_1$ a $\tau = V\beta$.
- $e = \lambda x.e_1$: nechť β je nová proměnná, a buď
  $W(\Gamma_x, x : \beta, e_1) = (S_1, \tau_1)$.
  Pak $S = S_1$ a $\tau = S_1(\beta) \to \tau_1$.
- $e = (\text{let } x = e_1 \text{ in } e_2)$: buď
  $W(\Gamma, e_1) = (S_1, \tau_1)$ a buď
  $W(S_1\Gamma_x, x : \overline{S_1\Gamma}(\tau_1), e_2) = (S_2, \tau_2)$.
  Pak $S = S_2 \circ S_1$ a $\tau = \tau_2$.

Není-li splněná některá z podmínek daného bodu, algoritmus selže.

let výrazy

Nutnost *let* výrazů

Example

Term $(\lambda x \to M)\ N$ není z hlediska typovacího systému ekvivalentní **let** $x = N$ **in** M (i když semanticky je):

1. Výraz **let** $x = \lambda z \to z$ **in** $x\ x$ má typ $(a \to a)$.
2. Výraz $(\lambda x \to x\ x)\ (\lambda z \to z)$ nemá typ.

V prvním případě má x polymorfní typ $(a \to a)$, který je ve výrazu $x\ x$ instancován v prvním x na $((a \to a) \to (a \to a))$ a v druhém na $(a \to a)$. V druhém případě nelze takto instanciovat typ x.

Osnova

Rekurzivní typy

- Hindley-Milnerův typový systém neumožňuje rekurzivní **let** výrazy.
- Například **let** $x = x$ **in** x nejde typovat. (Cvičení: proč?)

Řešení 1: Kombinátor Y

- Můžeme přidat kombinátor Y jako primitivní funkci s typem $\forall\alpha.(\alpha \to \alpha) \to \alpha$.
- Problém: Není zřejmé, jak se bude typový systém a algoritmus odvození chovat.
- Rekurze není součástí typového systému.

Řešení 2: Pravidlo pro uniformní rekurzi

- Do jazyka přidáme další syntaktickou konstrukci pro *uniformní rekurzi* (též zvanou *monomorfní*):

$$\mathsf{fix}\, x.e$$

- Konstrukce $\mathsf{fix}\, x.e$ nahrazuje $Y(\lambda x.e)$.
- Přidáme pravidlo:[1]

$$(\text{FIX-U})\ \frac{\Gamma_x, x : \tau \vdash e : \tau}{\Gamma \vdash \mathsf{fix}\, x.e : \tau}$$

[1] Symbol τ značí podle konvence typy, ne typová schémata!

Řešení 2 – problémy I

Rekurzivní funkce nemohou volat samy sebe s jiným typem v rekurzi. Například:

$$\textbf{data}\ \mathit{Seq}\ a = \mathit{Nil} \mid \mathit{Seq}\ a\ (\mathit{Seq}\ (a, a))$$

$$\mathit{size} :: \mathit{Seq}\ a \to \mathit{Int}$$
$$\mathit{size}\ \mathit{Nil} = 0$$
$$\mathit{size}\ (\mathit{Seq}\ \_\ s) = 1 + 2 * \mathit{size}\ s$$

nelze zapsat pomocí fix.

Řešení 2 – problémy II

- Při některých vícenásobných rekurzích vycházejí méně obecné typy. Například:

$$
\begin{aligned}
f = \textbf{let}\\
\quad f'\ x = x\\
\quad g'\ y = f'\ 3\\
\quad \textbf{in}\ f'
\end{aligned}
$$

Řešení 2 – problémy II

- Při některých vícenásobných rekurzích vycházejí méně obecné typy. Například:

$$
\begin{aligned}
f = \textbf{let}\\
\quad f'\ x = x\\
\quad g'\ y = f'\ 3\\
\quad \textbf{in}\ f'
\end{aligned}
$$

- Zkracujme typ páru jako (τ, τ') a zkracujme pair $e_1\, e_2$ jako (e_1, e_2) .
- Při přepisu dvojnásobné rekurze pomocí páru dostáváme:

$$(f, g) = (\text{fix}\, p.\ \underbrace{((\lambda x.x),\ (\lambda y.(\text{first}\ p)3))})$$

- Označený term je typu $(\alpha \to \alpha, \beta \to \gamma)$.
- První funkce musí být aplikovatelná na 3, takže α musí být *Int* a zároveň γ musí být α. Takže typ abychom mohli použít FIX-U, musí typ být $(\mathit{Int} \to \mathit{Int}, \beta \to \mathit{Int})$.
- Výsledný typ f tak bude $\mathit{Int} \to \mathit{Int}$.

Řešení 3: Polymorfní pravidlo pro rekurzi

- Polymorfní rekurzi rozpracoval poprvé [**mycroft1984polymorphic**].
- *Polymorfní rekurzi* definuje pravidlo:[2]

$$\text{(FIX-P)}\ \frac{\Gamma_x, x : \sigma \vdash e : \sigma}{\Gamma \vdash \text{fix}\, x.e : \sigma}$$

- K algoritmu pro odvození principiálního typu přidáme:

$e = \text{fix}\, x.e_1$ buď

$\sigma_0 = \forall\beta.\beta$

$\Gamma_0 = \Gamma_x \cup \{x : \sigma_0\}$

opakujeme:

- $(S_{i+1}, \tau_{i+1}) = W(\Gamma_i, e_1)$
- $\sigma_{i+1} = \overline{S_{i+1}\Gamma_i}(\tau_{i+1})$
- $\Gamma_{i+1} = (S_{i+1}\Gamma_i)_x \cup \{x : \sigma_{i+1}\}$

dokud $S_{i+1}\sigma_i \neq \sigma_{i+1}$.

Pak $S = S_{i+1} \ldots S_1$ a $\tau = \tau_{i+1}$.

[2] Rozdíl: zde už pravidlo obsahuje typová schémata.

Polymorfní rekurze v našem příkladu

Hledejme typ výrazu $\mathsf{fix}\, p.((\lambda x.x), (\lambda y.(\mathsf{first}\ p)3))$.
V našem případě je vždy $\Gamma_i = \{p : \sigma_i\}$ a σ_i nemají volné proměnné.
Proto $S_{i+1}\Gamma_i = \Gamma_i$ a $\overline{S_{i+1}\Gamma_i}(\tau_{i+1})$ okvantifikuje všechny proměnné τ_{i+1}.

1. $\sigma_0 = \forall\omega.\omega$,
2. $S_1 = [(\mathsf{Int} \to \alpha, \beta)/\omega]$,
   $\tau_1 = (\gamma \to \gamma, \delta \to \alpha)$,
   $\sigma_1 = \forall\alpha\gamma\delta.(\gamma \to \gamma, \delta \to \alpha)$,
3. $S_2 = [\mathit{Int}/\gamma]$,
   $\tau_2 = (\epsilon \to \epsilon, \zeta \to \mathit{Int})$,
   $\sigma_2 = \forall\epsilon\zeta.(\epsilon \to \epsilon, \zeta \to \mathit{Int})$,
4. $S_3 = [\mathit{Int}/\epsilon]$,
   $\tau_3 = (\eta \to \eta, \theta \to \mathit{Int})$,
   $\sigma_3 = \forall\eta\theta.(\eta \to \eta, \theta \to \mathit{Int}) = S_3\sigma_2$,

Kýžený principiální typ výrazu $\mathsf{fix}\, p. \ldots$ je tedy $\tau = \tau_3 = (\eta \to \eta, \theta \to \mathit{Int})$.
(Výsledná substituce je nezajímavá, neboť $\mathrm{FTV}(\Gamma) = \{\}$.)

Rozhodnutelnost odvození typů s polymorfní rekurzí

- Na některých termech se algoritmus zacyklí.
  Například na $\mathrm{fix}\, f.\lambda x.f$, protože bude vycházet
  $\sigma_n = \forall \alpha_0 \ldots \alpha_n.\alpha_0 \to \ldots \to \alpha_n$.

Rozhodnutelnost odvození typů s polymorfní rekurzí

- Na některých termech se algoritmus zacyklí.
  Například na $\mathrm{fix}\, f.\lambda x.f$, protože bude vycházet
  $\sigma_n = \forall \alpha_0 \ldots \alpha_n.\alpha_0 \to \ldots \to \alpha_n$.
- Přidání polymorfní rekurze činí Hindley-Milnerův systém nerozhodnutelný henglein1993type.

Rozhodnutelnost odvození typů s polymorfní rekurzí

- Na některých termech se algoritmus zacyklí.
  Například na $\mathrm{fix}\, f.\lambda x.f$, protože bude vycházet
  $\sigma_n = \forall \alpha_0 \ldots \alpha_n.\alpha_0 \to \ldots \to \alpha_n$.
- Přidání polymorfní rekurze činí Hindley-Milnerův systém nerozhodnutelný henglein1993type.
- Proto je nutné v některých případech explicitně typovat.
  Například pro

$$\textbf{data}\ \mathit{Seq}\ a = \mathit{Nil} \mid \mathit{Seq}\ a\ (\mathit{Seq}\ (a, a))$$

$$\mathit{size} :: \mathit{Seq}\ a \to \mathit{Int}$$
$$\mathit{size}\ \mathit{Nil} = 0$$
$$\mathit{size}\ (\mathit{Seq}\ \_\ s) = 1 + 2 * \mathit{size}\ s$$

Haskell vyžaduje explicitní typ pro *size*.

Poznámky

- Vztah mezi uniformní a polymorfní rekurzí je podobný jako mezi $(\lambda x.e)e'$ a $\mathsf{let}\, x = e' \,\mathsf{in}\, e$.
- Jakmile přidáme kterýkoliv ze způsobů zavádějící obecnou rekurzi, nemůže být typový systém ani silně ani slabě normalizující.

Osnova

Unifikační algoritmus

- Popíšeme algoritmus, který vždy najde nejobecnější unifikátor (MGU) dané množiny výrazů, existuje-li.
- Více viz. [**baader2001unification**].
- (Efektivnější algoritmus byl navržen v [**martellimontanari1982**].)

Nesouhlasící množina

Definition (nesouhlasící množina)

Buď $\mathbf{W} = \{E_1, \ldots, E_n\}$ množina různých výrazů a buď k délka úseku zleva, který je stejný u všech E_i. Buď

$$\mathbf{D} = \{D_i \mid 1 \leqslant i \leqslant n,\ D_i \text{ je pod-výraz } E_i \text{ začínající na } k+1. \text{ pozici}\}$$

D nazveme *nesouhlasící množinou* **W**.

Example

Pro

$$\mathbf{W} = \{P(x, \underbrace{f(y,z)}), P(x, \underbrace{a}), P(x, \underbrace{g(h(k(x)))})\}$$

bude

$$\mathbf{D} = \{f(y,z), a, g(h(k(x)))\}$$

Unifikační algoritmus

1. Volme $k = 0$, $\mathbf{W}_0 = \mathbf{W}$ a $S_0 = []$.
2. Je-li $|\mathbf{W}_k| = 1$, skončíme; S_k je MGU $\mathbf{W}$.
   V opačném případě najdeme nesouhlasící množinu $\mathbf{D}_k$ pro $\mathbf{W}_k$.
3. Jestliže existuje proměnná $x_k \in \mathbf{D}_k$ a term $t_k \in \mathbf{D}_k$ takové, že x_k se nevyskytuje v t_k, pokračujeme následujícím krokem.
   Pokud ne, skončíme; $\mathbf{W}$ není unifikovatelná.
4. Volme $S_{k+1} = [t_k/x_k] \circ S_k$ a $\mathbf{W}_{k+1} = [t_k/x_k]\mathbf{W}_k$
   (čili $\mathbf{W}_{k+1} = S_{k+1}\mathbf{W}$).
5. Položme $k = k + 1$ a jděme do ??.

Example

Najděte MGU $\mathbf{W} = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}$.

Věty o unifikačním algoritmu

Theorem

Pro konečnou neprázdnou množinu výrazů $\mathbf{W}$ *algoritmus vždy skončí.*

Důkaz.

V k-tém kroku množina $S_{k+1}\mathbf{W}$ obsahuje o jednu proměnnou méně než množina $S_k\mathbf{W}$. □

Theorem (Věta o unifikaci)

Je-li $\mathbf{W}$ *neprázdná unifikovatelná množina výrazů, pak unifikační algoritmus najde MGU této množiny.*

Důkaz věty o unifikaci

Důkaz.

- Buď T libovolný unifikátor $\mathbf{W}$. Ukážeme, že v každém $T = T \circ S_k$, tedy $T\mathbf{W} = TS_k\mathbf{W}$.
- Zkoumejme ??. krok algoritmu v k-té iteraci. Jelikož $T = T \circ S_k$ unifikuje $\mathbf{W}$, unifikuje T množinu $\mathbf{D}_k$ (nesouhlasící množina pro $S_k\mathbf{W}$). $\mathbf{D}_k$ tedy musí obsahovat nějakou proměnnou x_k a term t_k různý od x_k a $Tx_k = Tt_k$. Pokud by t_k obsahoval x_k pak by Tt_k obsahoval Tx_k, což nelze. Algoritmus tedy neskončí a pokračuje do kroku ??.
- Nechť algoritmus volí $S_{k+1} = [t_k/x_k] \circ S_k$. Pak

$$T \circ S_{k+1} = (T \circ [t_k/x_k]) \circ S_k = T \circ S_k$$

neboť $Tx_k = Tt_k$ a t_k neobsahuje x_k.

Osnova

Hindley-Milnerův typový systém v Haskellu

- Haskell 98 používá HM typový systém s polymorfní rekurzí.
- Všechy typy jsou implicitně univerzálně kvantifikované.
- Kontext není uživateli přístupný.[3]
  Nelze proto specifikovat typ s proměnnou vázanou v kontextu.
  Například:

$$
\begin{aligned}
&\mathit{function} :: (\mathit{Int} \to a) \to (a, a) \\
&\mathit{function}\ f = \\
&\quad \textbf{let} \\
&\qquad t = f\ 4 \\
&\quad \textbf{in}\ (t, t)
\end{aligned}
$$

Zde nelze explicitně otypovat vnořenou funkci t. GHC hlásí:

> *Couldn't match expected type 'a1' against inferred type 'a'*
> *'a1' is a rigid type variable bound by the type signature for 't' at ...*
> *'a' is a rigid type variable bound by the type signature for 'function' at ...*

[3] V GHC 7 to půjde pomocí pragma *ScopedTypeVariables*.

Syntaxe jazyka Haskell

Kompletní syntaxi Haskellu popisuje Haskell 98 Language and Libraries, AKA *The Haskell 98 Report*. **TODO** odkaz

Osnova

Cvičení I

Cvičení

Odvoďte z axiomů Hindley-Milnerova typového systému principiální typy těchto výrazů:

1. $\lambda x \to x$ *(kombinátor I)*
2. $\lambda x\ y \to x$ *(kombinátor K)*
3. $\lambda x\ y\ z \to (x\ z)\ (y\ z)$ *(kombinátor S)*
4. **let** $x = \lambda z \to z$ **in** $x\ x$
5. $(\lambda x \to x\ x)\ (\lambda z \to z)$

popřípadě ukažte, proč typ nemají.

Cvičení

Implementujte Hindler-Milnerův algoritmus.

Osnova

Literatura I