# Typové systémy polymorfního lambda kalkulu

Pavel Dvořák (172770)

IA038: jaro 2011

## Úvod

Polymorfní lambda kalkul rozšiřuje jednoduše typovaný lambda kalkul o typové proměnné. K jeho otypování se používají dva obecně známé typové systémy: Girardův-Reynoldsův (také uváděn jako System F – viz [1, kapitola 11]) a jeho oslabená verze, nazvaná Hindleyho-Milnerův. Při implementacích typové kontroly a typového odvození se však v praxi (například u funkcionálních programovacích jazyků jako jsou Haskell a ML, což bylo popsáno v [2, kapitola 8]) využívá pouze druhá jmenovaná varianta. Ukážeme si důvody, jež za tímto výběrem stojí.

## Definice

### Girardův-Reynoldsův typový systém

Kompletní definici polymorfního lambda kalkulu, jenž je postaven na Girardově-Reynoldsovu typovém systému, můžeme nalézt v [3, strana 343].

#### Syntaxe

| | | |
|---|---|---|
| $t ::=$ | | *Termy* sestávají z: |
| | $x$ | proměnných |
| | $\lambda x : T.t$ | abstrakcí |
| | $t\,t$ | aplikací |
| | $\lambda X.t$ | typových abstrakcí |
| | $t\,[T]$ | typových aplikací |

| | | |
|---|---|---|
| $v ::=$ | | *Hodnoty* sestávají z: |
| | $\lambda x : T.t$ | abstraktních hodnot |
| | $\lambda X.t$ | typových abstraktních hodnot |

| | | |
|---|---|---|
| $T ::=$ | | *Typy* sestávají z: |
| | $X$ | typových proměnných |
| | $T \to T$ | typových funkcí |
| | $\forall X.T$ | univerzálních typů |

| | | |
|---|---|---|
| $\Gamma ::=$ | | *Typový kontext* sestává z: |
| | $\varnothing$ | prázdného kontextu |
| | $\Gamma, x : T$ | navázání proměnných |
| | $\Gamma, X$ | navázání typových proměnných |

Syntaxe kalkulu nám poskytuje elementární prvky, s nimiž můžeme dále pracovat.

#### Vyhodnocení $t \to t'$

$$\frac{t_1 \to t_1'}{t_1\,t_2 \to t_1'\,t_2} \quad \text{(E-App1)}$$

$$\frac{t_2 \to t_2'}{v_1\,t_2 \to v_1\,t_2'} \quad \text{(E-App2)}$$

$$(\lambda x : T_{11}.t_{12})v_2 \to [x \mapsto v_2]\,t_{12} \quad \text{(E-AppAbs)}$$

$$\frac{t_1 \to t_1'}{t_1[T_2] \to t_1'[T_2]} \quad \text{(E-TApp)}$$

$$(\lambda X.t_{12})[T_2] \to [X \mapsto T_2]\,t_{12} \quad \text{(E-TappTabs)}$$

Podle této sady pravidel probíhá v kalkulu vyhodnocení výrazů.

**Otypování** $\Gamma \vdash t : T$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \text{(T-VAR)}$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1 . t_2 : T_1 \to T_2} \quad \text{(T-ABS)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1 t_2 : T_{12}} \quad \text{(T-APP)}$$

$$\frac{\Gamma, X \vdash t_2 : T_2}{\Gamma \vdash \lambda X . t_2 : \forall X . T_2} \quad \text{(T-TABS)}$$

$$\frac{\Gamma \vdash t_1 : \forall X . T_{12}}{\Gamma \vdash t_1 [T_2] : [X \mapsto T_2] T_{12}} \quad \text{(T-TAPP)}$$

V typovaném lambda kalkulu má každý výraz přiřazen svůj typ, případně by mělo být možné jeho typ odvodit. Otypování nám stanovuje, jak toto odvození má probíhat a jak můžeme mezi sebou typy kombinovat.

Tento typový systém odpovídá dle Curryho-Howardova izomorfismu části formálního systému intuicionistické logiky druhého řádu, která obsahuje pouze univerzální kvantifikátor.

### Hindleyho-Milnerův typový systém

Oslabená variace výše definovaného systému mu plně odpovídá, až na definici typů. Rozdíl spočívá v tom, že se univerzální kvantifikátor může vyskytovat pouze na začátku typu:

| | |
|---|---|
| $T ::=$ | *Typy* sestávají z: |
| $U$ | těla typu |
| $\forall X . T$ | univerzálních typů |

| | |
|---|---|
| $U ::=$ | *Tělo typu* sestává z: |
| $X$ | typových proměnných |
| $U \to U$ | typových funkcí |

# Typové odvození

Jak jsme již uvedli, v typovaném lambda kalkulu musí mít každý výraz svůj typ. Toto se týká i podvýrazů. Jelikož je velice nepraktické psát ke každému výrazu a jeho součástem explicitně veškeré typy, používá se k tomuto účelu odvození (inference), které je součástí typové kontroly některých překladačů.

### Girardův-Reynoldsův typový systém

Po objevení polymorfního lambda kalkulu na začátku 70. let 20. století dlouho nebylo jasné, jak to s jeho otypováním je. Nikdo nebyl s to sestrojit algoritmus, který by fungoval spolehlivě pro všechny výrazy. Teprve o více než dvacet let později přišel pan Wells na to, že typové odvození a typová kontrola v Girardově-Reynoldsovu typovém systému jsou totéž a že se jedná o nerozhodnutelný problém. Tuto ideu dokázal a popsal ve své práci [4].

V tomto typovém systému tím pádem není možné odvození provést a v praxi musíme použít jinou variantu typového systému polymorfního lambda kalkulu.

### Hindleyho-Milnerův typový systém

Pokud však přijmeme restrikci možného výskytu univerzálního kvantifikátoru, můžeme aplikovat jednoduchý postup, pomocí kterého typové odvození spolehlivě proběhne. Do detailu je popsán v [3, kapitola 22]. Tato restrikce naštěstí není příliš svazující, univerzální kvantifikátor v těle typu je nutný jen zřídka.

#### Stanovení omezení

Začneme tím, že vezmeme veškerý neotypovaný kód a stanovíme množinu typových omezení. Na tu se lze dívat jako na soustavu rovnic k vyřešení.

Kromě obecných pravidel otypování potřebujeme pravidla pro jednoduché typy. Kdybychom v kalkulu měli zavedeny typy $Nat$ (konstanta $zero$ představující nulu a funkce $succ$, $pred$ a $iszero$, které znamenají následníka, předchůdce a test na nulu), $Bool$ (konstanty $true$ a $false$ znázorňující logickou pravdu a nepravdu) a konstrukci $if-then-else$, vypadaly by v kontextu $\Gamma \vdash t : T$ definice takto:

$$\Gamma \vdash zero : Nat \qquad \text{(T-Zero)}$$

$$\frac{C' = C \cup \{T = Nat\}}{\Gamma \vdash succ\ t_1 : Nat} \qquad \text{(T-Succ)}$$

$$\frac{C' = C \cup \{T = Nat\}}{\Gamma \vdash pred\ t_1 : Nat} \qquad \text{(T-Pred)}$$

$$\frac{C' = C \cup \{T = Nat\}}{\Gamma \vdash iszero\ t_1 : Bool} \qquad \text{(T-IsZero)}$$

$$\Gamma \vdash true : Bool \qquad \text{(T-True)}$$

$$\Gamma \vdash false : Bool \qquad \text{(T-False)}$$

$$\frac{C' = C_1 \cup C_2 \cup C_3 \cup \{T_1 = Bool, T_2 = T_3\}}{\Gamma \vdash if\ t_1\ then\ t_2\ else\ t_3 : T_2} \qquad \text{(T-If)}$$

Po aplikaci těchto a obecných definic na výraz lambda kalkulu zjistíme jeho typová omezení a můžeme se pustit do jejich řešení.

### Unifikace

Jakmile máme k dispozici množinu všech typových omezení, stačí ji prohnat tzv. unifikačním algoritmem. Ten vezme všechna omezení a pokusí se pomocí nahrazení nalézt nejobecnější unifikaci, tedy takový sled nahrazení, který vyhovuje množině typových omezení a pro každé jeho nahrazení platí, že je méně specifické, než ostatní možná. V případě námi prezentovaného polymorfního lambda kalkulu se jedná o odvození, u něhož je nejvyšší možný počet typových proměnných oproti typům $Nat$ a $Bool$.

Unifikační algoritmus

```
1:  function UNIFY(C)
2:      if C = ∅ then
3:          []                          ▷ všechno jsme zpracovali
4:      else
5:          let {S = T} ∪ C' ← C in
6:          if S = T then
7:              UNIFY(C')
8:          else if S = X ∧ X ∉ FV(T) then
9:              UNIFY([X ↦ T]C') ∘ [X ↦ T]
10:         else if T = X ∧ X ∉ FV(S) then
11:             UNIFY([X ↦ S]C') ∘ [X ↦ S]
12:         else if S = S1 → S2 ∧ T1 → T2 then
13:             UNIFY(C' ∪ {S1 = T1, S2 = T2})
14:         else
15:             ⊥                       ▷ nelze unifikovat
16:         end if
17:     end if
18: end function
```

Kód na pátém řádku značí výběr omezení $S = T$ z množiny $C$ a uložení zbylých do proměnné $C'$. V podmínkách na osmém a desátém řádku je využita funkce $FV$, jež představuje všechny typové proměnné v konkrétním omezení. Pomocí této funkce se provádí *occurs check*, což je kontrola nahrazování. Může se totiž stát, že by některé z nahrazení vedlo k sestrojení nekonečného typu (příklad: $T \mapsto T \to T$). V případě, že by odvozený typ byl nekonečný, a tehdy, pokud typy nesedí, algoritmus skončí chybou.

## Závěr

Představili jsme si dvě nejběžnější varianty typových systémů pro polymorfní lambda kalkul a předvedli si jejich rozdíly. Dále jsme si vysvětlili, proč není možné použít Girardův-Reynoldsův

typový systém k implementaci spolehlivé typové kontroly. Také jsme si ukázali algoritmus pro typové odvození v Hindleyho-Milnerově typovém systému.

V teorii typů je oblast polymorfismu poměrně dobře zmapována a v programovacích jazycích využita, na rozdíl od kupříkladu hodnotově závislých typů, které své praktické uplatnění stále ještě nenalezly.

## Reference


[1] Jean-Yves Girard. *Proofs and Types*. Cambridge University Press, 1990.

[2] Simon Peyton Jones, Philip Wadler, Peter Hancock, and David Turner. *The Implementation of Functional Programming Languages*. Prentice Hall, 1987.

[3] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.

[4] Joe B. Wells. Typability and type checking in System F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98(1–3):111–156, 1999.

[5] John C. Reynolds. *Theories of Programming Languages*. Cambridge University Press, 1998.