Přírodovědecká fakulta Masarykovy univerzity v Brně
Katedra matematické informatiky

Funkcionální programování

dr. Libor Škarvada

říjen 1993 — leden 1994

Obsah

1 Úvod

Programovací jazyky:

imperativní (*Turingův stroj*) — výpočet je posloupností příkazů (instrukcí)

logické (*Predikátová logika*) — výpočet je posloupností splňování predikátů

funkcionální (λ*–kalkul*) — výpočet je posloupností vzájemně ekvivalentních výrazů, které se postupně zjednodušují

- **netypované** (*netypovaný λ–kalkul, Alonzo Church 1941*) — např. *LISP*, *Scheme*
- **typované** (*typovaný λ–kalkul, B. Curry*) — jazyky ML (Metalanguage), stále obsahují přiřazení, např. *Miranda* (Evropa), *Haskell* (Glasgow)

volání hodnotou	$\longrightarrow$	striktní vyhodnocení
volání jménem	$\longrightarrow$	líné vyhodnocení

Příklad 1.1
faktoriál (Haskell)

`facts=1: zipWith (*) facts` $\underbrace{[1\ldots]}_{(=[1,2,3,4,\ldots])}$

$$\begin{array}{rcccccccc} \text{čili} & [& 1 & 1 & 2 & 6 & 24 & 120 & \ldots\ ] \\ * & [& 1 & 2 & 3 & 4 & 5 & 6 & \ldots\ ] \\ \hline & [& 1 & 2 & 6 & 24 & 120 & 720 & \ldots\ ] \end{array}$$

Podobně lze zapsat celý Pascalův trojúhelník.

2 Netypovaný λ–kalkul

Jazyk se skládá z proměnných a termů. Označujeme ho Λ. Dále zavádíme několik speciálních symbolů (např.(,), λ).

Definice 2.1
Termy jazyka Λ lze získat právě konečným počtem aplikací těchto postupů:

(i). $x, y, \ldots \in \Lambda$ *proměnné*

(ii). $M, N \in \Lambda \implies M\,N \in \Lambda$ *aplikace*; pro M funkce a N hodnoty

(iii). $M \in \Lambda \implies \lambda x.M \in \Lambda$ *abstrakce*; M je term závislý na x, pak $\lambda x.M$ je funkce zobrazující A na term $M[A/x]$, který vznikne tak, že každý výskyt x v M se nahradí A

Příklad 2.2
Nechť

$$x^2 + 1$$

je term závislý na x. Pak

$$(\lambda x.(x^2 + 1))c \;=\; c^2 + 1$$

Mezi termy nadefinujeme relaci redukce $M \hookrightarrow N$ (např. $3+4 \hookrightarrow 7$). K tomu budeme potřebovat pojem *volné proměnné*:

Definice 2.3

- `x` je volná v `x`
- `x` je volná v `MN`, je-li volná v `M` nebo v `N`
- `x` je volná v λ`y.M`, pokud je volná v `M` a současně je různá od `y`

Definice 2.4
Redukce $\hookrightarrow \subseteq \Lambda^2$:

(i). *α–redukce*:

$$\lambda x.M \ \hookrightarrow^\alpha \ \lambda y.M[y/x]$$

pokud y není volná v M. Příklady:

$$(\lambda x.x^2+1) \ \hookrightarrow^\alpha \ (\lambda y.y^2+1)$$
$$(\lambda x.x^2+y) \ \not\hookrightarrow^\alpha \ (\lambda y.y^2+y)$$

(ii). *β–redukce*:

$$(\lambda x.M)N \ \hookrightarrow^\beta \ M[N/x]$$

Příklady:

$$(\lambda x.x^2+1)8 \ \hookrightarrow^\beta \ (x^2+1)[8/x] \ = \ (8^2+1)$$

(iii). *η–redukce*:

$$\lambda x.Mx \ \hookrightarrow^\eta \ M$$

pokud x není volná v M.

(iv). $\hookrightarrow$ je *kompatibilní uzávěr* relací $\hookrightarrow^\alpha$, $\hookrightarrow^\beta$, $\hookrightarrow^\eta$, což znamená:

$$\frac{M \hookrightarrow N}{PM \hookrightarrow PN}, \frac{M \hookrightarrow N}{\lambda x.M \hookrightarrow \lambda x.N}$$

kde $\frac{A}{B}$ značí $A \Rightarrow B$.

Definice 2.5
$\hookrightarrow^*$ je reflexivní a tranzitivní uzávěr relace $\hookrightarrow$.

Tedy $M \hookrightarrow^* N$ znamená, že N je výsledek M. Pokud $\neg\exists\, P : N \hookrightarrow P$, pak N je *normální formou (nf)* vzhledem k relaci $\hookrightarrow$.

Definice 2.6
Relace $\rightarrow \subseteq A^2$ je *silně normalizující* (*fundovaná*), když neexistuje nekonečná posloupnost $a_1 \ \rightarrow \ a_2 \ \rightarrow \ a_3 \ \rightarrow \dots$

Definice 2.7
Relace $\rightarrow$ je *Church—Rösserova* (*konfluentní*), jestliže $\forall\, a, b, c \in A$ takové, že $a \rightarrow^* b$, $a \rightarrow^* c \quad \exists\, d \in A : \ b \rightarrow^* d, c \rightarrow^* d$.

Jinými slovy: $\leftarrow^* \circ \rightarrow^* \subseteq \rightarrow^* \circ \leftarrow^* \quad (\leftarrow \,=\, \rightarrow^{-1})$ $\hookrightarrow$ není silně normalizující, je Church—Rösserova

Poznámka 2.8
proč není SN:
třeba: $(\lambda x.xx)(\lambda x.xx) \ \hookrightarrow \ xx[(xx.xx)/x] \ = \ (\lambda x.xx)(\lambda x.xx)$ a zacyklilo se to.

Definice 2.9
$\approx \,:=\, (\hookrightarrow \cup \hookleftarrow)^*$

3 Rekurze, věta o pevném bodě

Věta 3.1
$\forall$ term $F \quad \exists!$ term $X : \quad FX \approx X$

Důkaz Vytvoříme term

$$\varphi = \lambda f(\lambda x.f(xx))(\lambda x.f(xx))$$

a položme

$$X = \varphi F$$

. Pak $X = \varphi F = (\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))F \hookrightarrow^{\beta} (\lambda x.F(xx))(\lambda x.F(xx)) \hookrightarrow F((\lambda x.F(xx))(\lambda x.F(xx))) \hookleftarrow F((\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))F) = F(\varphi F) = FX$ za použití β–redukce a pravidla $\frac{N \hookrightarrow P}{MN \hookrightarrow MP}$. Celkem tedy

$$X \approx FX$$

Věta 3.2
Jinak zapsáno, je to: $\exists$ term φ : $\forall$ term F je $F(\varphi F) \approx \varphi F$.

Definice 3.3
φ z předchozí věty se nazývá *kombinátor pevného bodu*.

Poznámka 3.4
$\xi = N \ldots N$ je term obsahující ξ, v λ–kalkulu $X \approx N$. Platí $N \approx (\lambda x.M)X$, kde $M = N[x/X]$ a $x \notin F \cup (N) \ldots$ je volná proměnná. Chceme, aby $X \approx (\lambda x.M)X$ — platí z věty o pevném bodě.

$$X = \varphi(\lambda x.M)$$

Příklad 3.5

```
fakt = λm.if m=0 then 1 else m*fakt(m-1)
fakt = φ(λf.(λm.if m=0 then 1 else m*f(m-1)))
```

4 Typovaný λ–kalkul

Typy i je typ (Bool, int, char). Jsou-li σ, τ typy, pak $\sigma \to \tau$ je typ.

Termy

- $x^\sigma, y^\sigma, \ldots \in \Lambda^\sigma$ ($x \ldots$ je typu σ)
- je-li F typu $\sigma \to \tau$ a A je typu σ, pak $FA \in \Lambda^\tau \ldots$ *aplikace*
- je-li M typu τ a x proměnná typu σ, pak $\lambda x.M \in \Lambda^{\sigma \to \tau} \ldots$ *abstrakce*

Definice Relace *redukce*

$$\hookrightarrow: \bigcup_{\sigma\ je\ typ} \hookrightarrow^\sigma$$

$\hookrightarrow^\sigma \subseteq (\Lambda^\sigma)^2$ a relace $\hookrightarrow^\sigma$ jsou definovány stejně jako v netypovém λ–kalkulu

Věta Redukce $\hookrightarrow$ je CR. Redukce $\hookrightarrow$ je SN.

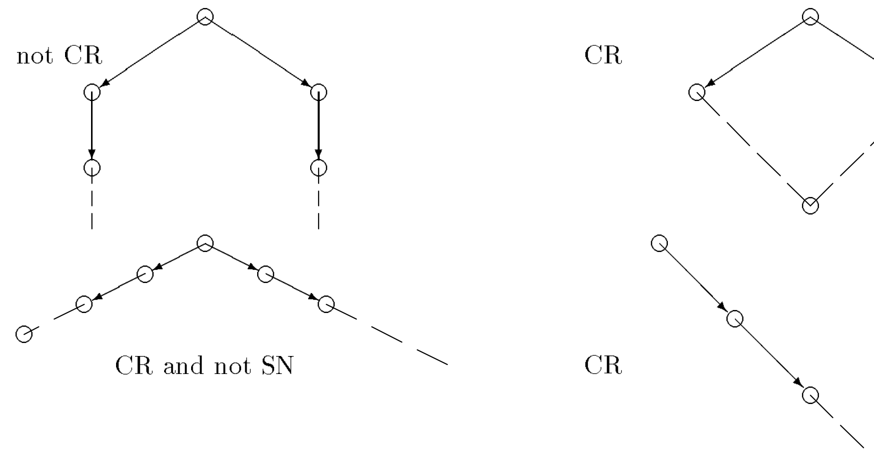
Definice Rozšíření typového λ–kalkulu o *konstanty* $c_1^\sigma, c_2^\sigma, \ldots \in \Lambda^\sigma$: Pro každý typ σ definujeme konstantu Y_σ — term typu $(\sigma \to \sigma) \to \sigma$. Přidáme σ–*pravidlo*:

$$Y_\sigma F^{\sigma \to \sigma} \hookrightarrow F(Y_\sigma F)$$

Nedeterminismus:

vnější — výsledek výpočtu je nejednoznačný

vnitřní — výsledek výpočtu je jednoznačný, ale výpočet není dán jednoznačně



Obr. 1: Ukázky větvení programů při různých typech determinismu

slabý vnitřní — i když se výpočet může zacyklit, vždy existuje způsob, jak se dostat k výsledku

silný vnější — výpočet v každém případě buď skončí a dá vždy stejný výsledek, nebo nikdy neskončí pro stejné vstupy

konečný — neexistují nekonečné větve ($CR \wedge SN$)

Deterministický programovací jazyk se liší od λ–kalkulu v tom, že je vnitřně deterministický.

5 Jazyk PCF

(*Programming language for Computable Functions*)

- je vnitřně deterministický
- má normální pořadí vyhodnocení
- jednoduše typovaný funkcionální jazyk, vnitřně deterministický, s normálním vyhodnocováním termů (redukcí)
- není SN, protože obsahuje konstanty: $Y_\sigma (\forall$ typ $\sigma)$ a δ–pravidla: $YF \hookrightarrow F(YF)$, (čili $\hookrightarrow F(F(YF)) \hookrightarrow F(F(F(YF))) \hookrightarrow \ldots$)

Odstranění prvního nebo druhého termu		
MN:	$\frac{M \hookrightarrow N}{MN \hookrightarrow PN}$	$\frac{N \hookrightarrow P}{MN \hookrightarrow MP}$
vyhodnocení:	normální líné	aplikativní striktní
volání:	jménem	hodnotou

Tabulka 1: Větvení v λ–kalkulu

Typy PCF

- základní: *bool*, *nat* ... *nulární typový konstruktor*
- $\forall$ typ σ, τ je $\sigma \to \tau$ typ ... $\to$ je *binární typový konstruktor*

Termy PCF

- $\forall$ typ σ jsou $x^\sigma, y^\sigma, \ldots$ termy typu σ (term M typu σ značíme $M : \sigma$)
- $M : \tau$, $x^\sigma : \sigma$; pak $\lambda x^\sigma.M : \sigma \to \tau$
- $M : \sigma \to \tau$, $N : \sigma$; pak $MN : \tau$

Konstanty PCF

- je-li $n \in \mathcal{N}$, pak $\overline{n}$ je term typu nat: $\overline{n} : nat$
- $true : bool$, $false : bool$
- $succ : nat \to nat$, $pred : nat \to nat$
- $zero? : nat \to bool$
- $\forall$ typ σ je $if_\sigma : (bool \to (\sigma \to (\sigma \to \sigma)))$
- $\forall$ typ σ je $Y_\sigma : (\sigma \to \sigma) \to \sigma$

Možné doplnění o *kartézský součin*

- **typy**: $\forall$ typy σ, τ je $\sigma \times \tau$ typ
- **termy**: $M : \sigma$, $N : \tau$; pak $(M, N) : \sigma \times \tau$
- **if**: $if'_\sigma : bool \times (\sigma \times \sigma) \to \sigma$
- Kartézský součin je v podstatě zbytečný:

$add : nat \to nat \to nat$	$add' : (nat \times nat) \to nat$
$add\ x : nat \to nat$	
$(add\ x)\ y : nat$	
$(add\ x)\ y = x + y$	$add'(x, y) = x + y$

Obecně: máme-li $f' : \alpha \times \beta \to \gamma$, pak $\exists\ f : \alpha \to \beta \to \gamma$ tak, že $f'(x, y) = f\ x\ y$.
$f\ x = \lambda y.f'(x, y)$, neboli $f = \lambda x.\lambda y.f'(x, y)$ Tomuto postupu se říká *karifikace* (*currying*):

$$curry_{\alpha,\beta,\gamma} : (\alpha \times \beta \to \gamma) \to (\alpha \to \beta \to \gamma)$$

Příklad

$$add = curry_{nat,nat,nat}\ add'$$

Má smysl, je-li povolen kartézský součin. Není-li povolen, pak jsou všechny funkce karifikované.

Redukce v PCF

- β*–redukce*:$(\lambda x^\sigma.M)N \hookrightarrow^\beta M[N/x]$
- $Y_\sigma M \hookrightarrow M(Y_\sigma M)$, je-li $M : \sigma \to \sigma$
- $if_\sigma\ true\ M\ N \hookrightarrow M$, je-li $M : \sigma, N : \sigma$
- $if_\sigma\ false\ M\ N \hookrightarrow N$, je-li $M : \sigma, N : \sigma$
- $M \hookrightarrow N \Longrightarrow if_\sigma\ M\ P\ Q \hookrightarrow if_\sigma\ N\ P\ Q \ldots$ striktní vyhodnocení
- $succ\ \overline{n} \hookrightarrow \overline{n+1}$
- $M \hookrightarrow N \Longrightarrow succ\ M \hookrightarrow succ\ N \ldots$ striktní vyhodnocení (pro $M, N : nat$)
- $pred\ \overline{n+1} \hookrightarrow \overline{n}$
- $M \hookrightarrow N \Longrightarrow pred\ M \hookrightarrow pred\ N \ldots$ striktní vyhodnocení (pro $M, N : nat$)
- $pred\ \overline{0} \hookrightarrow pred\ \overline{0}$
- $zero?\ \overline{0} \hookrightarrow true$
- $zero?\ \overline{n+1} \hookrightarrow false$
- $M \hookrightarrow N \Longrightarrow zero?\ M \hookrightarrow zero?\ N \ldots$ striktní vyhodnocení (pro $M, N : nat$)
- $M \hookrightarrow N \Longrightarrow MP \hookrightarrow NP \ldots$ líné vyhodnocení

Definice *Programem PCF* je term bez volné proměnné (uzavřený) základního typu ($M : nat$ nebo $M : bool$).

Věta Je-li M program v PCF, pak nastane právě jedna z následujících možností:

(a). M se v konečném počtu kroků redukuje na term $\overline{n}$. $M \hookrightarrow^* \overline{n}$ pro vhodné $n \in \mathcal{N}$ a $M : nat$

(b). $M : bool$ a $M \hookrightarrow^* false$

(c). $M : bool$ a $M \hookrightarrow^* true$

(d). existuje jednoznačně určená nekonečná posloupnost termů základního, ale téhož typu $M_1, M_2, \ldots$ tak, že $M \hookrightarrow M_1 \hookrightarrow M_2 \hookrightarrow \ldots$

Důkaz Jestliže $M \hookrightarrow N$, pak N je určeno jednoznačně termem M. Pro program M existuje term N tak, že $M \hookrightarrow N$ vždy kromě případů, kdy $M = true$ nebo $M = false$ nebo $M = \overline{n}$ pro nějaké $n \in \mathcal{N}$.

Příklad

$$
\begin{array}{ll}
add: & nat \to nat \to nat \\
add =^{(df)} & \lambda x^{nat}\ \lambda y^{nat}\ if_{nat}\ (zero?\ x)\ y \\
 & (add(pred\ x)(succ\ y)) \ \hookleftarrow^{\beta} \\
add =^{(df)} & (\lambda f^{nat\to nat\to nat} \lambda x^{nat}\ \lambda y^{nat}\ if_{nat}(zero?\ x)\ y \\
 & (f(pred\ x)(succ\ y)))add \ \hookleftarrow^{\delta}
\end{array}
$$

ale máme větu o pevném bodě, takže:

$$
\begin{array}{ll}
add = & Y_{nat\to nat\to nat}(\lambda f^{nat\to nat\to nat}\ \lambda x^{nat}\ \lambda y^{nat} \\
 & if_{nat}(zero?\ x)\ y(f(pred\ x)(succ\ y)))
\end{array}
$$

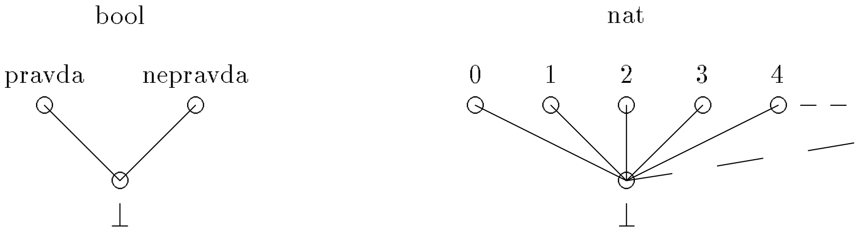
5.1 Operační sémantika PCF

je funkce, která každému programu přiřazuje jeho význam — může to být číslo, $true/false$, `nedefinováno` ($\perp$).

Definice *Operační sémantika* je $\{\!\!|\ \_\ |\!\!\} = \{\{\!\!|\ \_\ |\!\!\}_{bool}, \{\!\!|\ \_\ |\!\!\}_{nat}\}$, kde

$$\begin{array}{lcccl} \{\!\!|\ \_\ |\!\!\}_{bool} & : & \overline{\Lambda}^{bool} & \longrightarrow & \{\perp, pravda, nepravda\} \\ \{\!\!|\ \_\ |\!\!\}_{nat} & : & \overline{\Lambda}^{nat} & \longrightarrow & \{\perp\} \cup \mathcal{N} \end{array}$$

a $\overline{\Lambda}^{\sigma}$ je množina programů typu σ. Uspořádané množiny tohoto typu — *ploché domainy* takové, že pro každý program



Obr. 2: Uspořádání množin výsledků operační sémantiky

M:

$$\{\!\!|\ M\ |\!\!\} = \begin{cases} n, & M \hookrightarrow^{*} \overline{n} \\ pravda, & M \hookrightarrow^{*} true \\ nepravda, & M \hookrightarrow^{*} false \\ \perp, & M \hookrightarrow M_1 \hookrightarrow M_2 \hookrightarrow \ldots \end{cases}$$

	uzavřené	s volnými proměnnými
základní ($bool, nat$)	programy	(xx)
funkcionální ($\sigma \rightarrow \tau$)	$\lambda x(xx)$	

Tabulka 2: Rozdělení termů

5.2 Denotační sémantika PCF

Definice *Denotační sémantika* je $[\![\ \_\ ]\!] = \{[\![\ \_\ ]\!]_{\sigma} \mid \sigma \text{ je typ } \}$, kde

$$[\![\ \_\ ]\!]_{\sigma} : \Lambda^{\sigma} \longrightarrow (Env \longrightarrow D_{\sigma})$$

a $Env \longrightarrow D_{\sigma}$ je přiřazení volným proměnným; D_{σ} je doména (obor hodnot):

$$\begin{array}{lcl} D_{nat} & = & \mathcal{N}_{\perp} \\ D_{bool} & = & \mathcal{B}_{\perp} \\ D_{\tau \rightarrow \upsilon} & = & [D_{\tau} \longrightarrow D_{\upsilon}] \end{array}$$

$D_{\tau \rightarrow \upsilon}$ ... doména všech spojitých funkcí.

Definice Nechť f, g jsou funkce. Pokud $f(x) = g(x)$ nebo $f(x) = \perp \wedge g(x) \neq \perp$, pak g je *v bodě x definovanější*; $(f\ x \sqsubseteq g\ x)$.

Jestliže $\forall x :\ f\ x \sqsubseteq g\ x$, pak $f \sqsubseteq g$; g je *definovanější* než f.

Poznámka Pro každou posloupnost funkcí $f_1 \sqsubseteq f_2 \sqsubseteq f_3 \sqsubseteq \ldots$ existuje suprémum $\bigsqcup_{sup}\{f_i \mid i \in \mathcal{N}\}$.

Definice $Env = \{Env_\sigma : \sigma \text{ je typ }\}$; $Env_\sigma : Var_\sigma \longrightarrow D_\sigma$; Var_σ je množina volných proměnných. Pro každé přiřazení $\varrho \in Env_\sigma$ a každou proměnnou $x^\sigma \in Var_\sigma$ hodnota $d \in D_\sigma$ je $\varrho[x^\sigma \mapsto d] \in Env_\sigma$, které vznikne z ϱ změnou funkční hodnoty v x^σ z $\varrho\, x^\sigma$ na d, tj.

$$\varrho[x^\sigma \mapsto d]y = \begin{cases} d, & x^\sigma = y \\ \varrho y, & jinak \end{cases} \quad (\forall y \in Var_\sigma)$$

Sémantická funkce $[\![\ \_\ ]\!]_\sigma$ je definována:

1). konstanty:

- $[\![\ \overline{n}\ ]\!]_{nat}\varrho = n \quad \forall n \in \mathcal{N}$
- $[\![\ true\ ]\!]_{bool}\varrho = pravda$
- $[\![\ false\ ]\!]_{bool}\varrho = nepravda$
- $[\![\ if_\sigma\ ]\!]_{D\to\sigma\to\sigma\to\sigma}\varrho : D_{bool} \longrightarrow D_\sigma \longrightarrow D_\sigma \longrightarrow D_\sigma$ taková, že pro $b \in D_{bool}; x, y \in D_\sigma$ je

$$[\![\ if_\sigma\ ]\!]_{bool\to\sigma\to\sigma\to\sigma}\varrho\ b\ x\ y = \begin{cases} x, & b = pravda \\ y, & b = nepravda \\ \bot, & b = \bot \end{cases}$$

- $[\![\ succ\ ]\!]_{nat\to nat}\varrho : D_{nat} \longrightarrow D_{nat} : \forall n \in D_{nat}$:

$$[\![\ succ\ ]\!]_{nat\to nat}\varrho\ n = \begin{cases} n+1, & n \in \mathcal{N} \\ \bot, & n = \bot \end{cases}$$

- $[\![\ pred\ ]\!]_{nat\to nat}\varrho : D_{nat} \longrightarrow D_{nat} : \forall n \in D_{nat}$:

$$[\![\ pred\ ]\!]_{nat\to nat}\varrho\ n = \begin{cases} n-1, & n \in \mathcal{N} - \{0\} \\ \bot, & n = \bot \vee n = 0 \end{cases}$$

- $[\![\ zero?\ ]\!]_{nat\to bool}\varrho : D_{nat} \longrightarrow D_{bool} : \forall n \in D_{nat}$:

$$[\![\ zero?\ ]\!]_{nat\to bool}\varrho\ n = \begin{cases} pravda, & n = 0 \\ nepravda, & n \in \mathcal{N} - \{0\} \\ \bot, & n = \bot \end{cases}$$

- $[\![\ Y_\sigma\ ]\!]_{(\sigma\to\sigma)\to\sigma}\varrho : D_{\sigma\to\sigma} \longrightarrow D_\sigma$. Pro spojitou funkci $f \in D_{\sigma\to\sigma}$ je

$$[\![\ Y_\sigma\ ]\!]_{(\sigma\to\sigma)\to\sigma}\varrho\ f = \bigsqcup_{(sup)} \{f^n \bot \mid n \in \mathcal{N}\}$$

Tedy $\begin{matrix} f^0 x = \bot \\ f^{n+1}x = f(f^n x) \end{matrix}\ \forall x \in D_\sigma, n \in \mathcal{N}$

2). proměnné: $[\![\ x^\sigma\ ]\!]_\sigma \varrho = \varrho x^\sigma$

3). term je *aplikace*: $[\![\ MN\ ]\!]_\sigma\varrho = [\![\ M\ ]\!]_{\tau\to\sigma}\varrho([\![\ N\ ]\!]_\tau\varrho)$ pro $M \in \Lambda^{\tau\to\sigma}, N \in \Lambda^\tau$

4). $[\![\ \lambda x^\tau.M\ ]\!]_{\tau\to\upsilon}\varrho : D_\tau \longrightarrow D_\upsilon$, kde pro libovolné $d \in D_\tau$ je

$$[\![\ \lambda x^\tau.M\ ]\!]_{\tau\to\upsilon}\varrho\ d = [\![\ M\ ]\!]_\upsilon(\varrho[x^\tau \mapsto d])$$

Věta *o korespondenci*
Pro všechny programy M platí:

$$[\![\ M\ ]\!] = \{\!|\ M\ |\!\}$$

Poznámka Je-li M uzavřený, pak $[\![M]\!]\varrho = [\![M]\!]\varrho'$ pro každou dvojici přiřazení ϱ, ϱ' (environment). Proto značíme $[\![M]\!] = [\![M]\!]\varrho$.

Definice *Kontext* je term s jedinou volnou proměnnou. Kontext typu *nat* nebo *bool* je tzv. *základní kontext*. Je-li C kontext a x^σ je jeho volná proměnná a M je term typu σ, pak místo $C[M/x]$ píšeme $C[M]$.

Definice Říkáme, že denotační sémantika $[\![\_]\!]$ je *plně abstraktní*, jestliže pro každé dva uzavřené termy M, N typu σ platí ekvivalence

$$[\![M]\!] \sqsubseteq [\![N]\!] \iff \forall C: \{\!| C[M] |\!\} \sqsubseteq \{\!| C[N] |\!\}$$

kde C je základní kontext s volnou proměnnou typu σ.

Poznámka Ekvivalentně: Říkáme, že denotační sémantika $[\![\_]\!]$ je *plně abstraktní*, jestliže pro každé dva uzavřené termy M, N typu σ platí ekvivalence

$$[\![M]\!] \sqsubseteq [\![N]\!] \iff \forall C: [\![C[M]]\!] \sqsubseteq [\![C[N]]\!]$$

kde C je základní kontext s volnou proměnnou typu σ.

Otázka Je PCF plně abstraktní?

Odpověď Ne.

Důkaz Implikace $\Longrightarrow$ platí, protože C je funkce, a je-li tedy $[\![N]\!]$ definovanější než $[\![M]\!]$, pak $[\![C[N]]\!]$ musí být definovanější než $[\![C[M]]\!]$.

Lze však nalézt dva termy nesrovnatelné v relaci $\sqsubseteq$, ale u odpovídajících kontextů nastane rovnost.

Řešení

a). změnit definici denotační sémantiky tak, aby PCF byl plně abstraktní

b). zavedení do PCF $p - or : bool \to bool \to bool$ a δ–pravidla pro $p - or$ (viz obrázek 3 a níže).

$p - or$	$\perp$	T	F
$\perp$	$\perp$	T	$\perp$
T	T	T	T
F	$\perp$	T	F

$s - or$	$\perp$	T	F
$\perp$	$\perp$	$\perp$	$\perp$
T	$\perp$	T	T
F	$\perp$	T	F

$l - or$	$\perp$	T	F
$\perp$	$\perp$	$\perp$	$\perp$
T	T	T	T
F	$\perp$	T	F

$r - or$	$\perp$	T	F
$\perp$	$\perp$	T	$\perp$
T	$\perp$	T	T
F	$\perp$	T	F

Obr. 3: Paralelní, striktní, levý a pravý *or*

Chceme tedy ukázat, že

$$\exists M, N : \forall C : [\![C[M]]\!] \sqsubseteq [\![C[N]]\!] \wedge [\![M]\!] \not\sqsubseteq [\![N]\!]$$

Důkaz pomocí *or*–funkcí, které se liší ve výsledcích na vstupech (T,$\perp$) a ($\perp$,T) — viz obrázek 3:

striktní $(s - or)$ — v obou výše uvedených případech je výsledek $\perp$

paralelní $(p - or)$ — výsledek je vždy T, jde o monotónní rozšíření předchozí funkce

levý $(l - or)$ — funkční hodnota v bodě (T,$\perp$) je T, v bodě ($\perp$,T) $\perp$

pravý $(r-or)$ — opačně než $l-or$

Je zřejmé, že platí následující vztah:

$$s-or \sqsubseteq \begin{matrix} l-or \\ r-or \end{matrix} \sqsubseteq p-or$$

Vyjádření funkcí v PCF:

$$\begin{array}{rl} l-or: & \lambda x.\lambda y.if\ x\ true\ y \\ r-or: & \lambda x.\lambda y.if\ y\ true\ x \\ p-or: & \text{nelze vyjádřit} \\ s-or: & \lambda x.\lambda y.if\ x\ (if\ y\ x\ x)\ y \end{array}$$

$p-or$ nelze vyjádřit, protože if nemůže mít první parametr $\perp$.

$\Omega = Y_{bool}(\lambda x^{bool}.x)$
$YM \hookrightarrow M(YM)$
$\Omega \hookrightarrow (\lambda x.x)\Omega \hookrightarrow \Omega \hookrightarrow \ldots$
$[\![\Omega]\!] = \perp$
$M = \lambda f^{bool \to bool \to bool}.if(f\ \Omega\ true)(if(f\ true\ \Omega)(if(f\ false\ false)\ \Omega\ \underline{true})\ \Omega)\ \Omega$
$N = \lambda f^{bool \to bool \to bool}.if(f\ \Omega\ true)(if(f\ true\ \Omega)(if(f\ false\ false)\ \Omega\ \underline{false})\ \Omega)\ \Omega$
$[\![M]\!]\ p-or = [\![true]\!] = T$ (pravda)
$[\![N]\!]\ p-or = [\![false]\!] = F$ (nepravda)

$$\left.\begin{array}{rcccc} [\![M]\!]\ p-or & = & [\![true]\!] & = & T \\ [\![N]\!]\ p-or & = & [\![false]\!] & = & F \end{array}\right\} \Longrightarrow [\![M]\!] \begin{matrix} \not\sqsubseteq \\ \not\sqsupseteq \end{matrix} [\![N]\!]$$

Závěr

- PCF není plně abstraktní
- Funkce $p-or$ není vyjádřitelná žádným termem
- Plná abstraktnost lze dosáhnout, jestliže pro každou funkci existuje term, tedy je ekvivalentní množina termů a funkcí
- Přidáme-li do PCF $p-or$ a odpovídající pravidlo, dosáhneme plné abstraktnosti

Věta V PCF neexistuje term $\underline{p-or}$ takový, že $[\![\underline{p-or}]\!] = p-or$.

Důkaz Kdyby existoval, pak by

$$\underline{p-or}\ true\ \Omega \hookrightarrow^* true\ \wedge\ \underline{p-or}\ \Omega\ true \hookrightarrow^* true$$

$\exists M:\ p-or: bool \to bool \to bool = \lambda x.\lambda y.M$
Pak ale $(\lambda x.\lambda y.M)\ true\ \Omega\ \hookrightarrow^\beta\ (\lambda y.M[true/x])\ \Omega \hookrightarrow^\beta$
$\hookrightarrow^\beta\ M[true/x][\Omega/y]\ \hookrightarrow^*\ N[true/x][\Omega/y] \ldots$ pro $M \hookrightarrow^* N$
a také $(\lambda x.\lambda y.M)\ \Omega\ true\ \hookrightarrow^*\ N[\Omega/x][true/y]$

pro nějaký základní term N. Je tedy třeba začít redukovat argumenty. To ale nejde; alespoň jeden z nich se vždy zacyklí.

6 Funkcionální jazyky

6.1 Rysy reálných funkcionálních jazyků

Definice

$$\texttt{let}\ \underline{x}\ \texttt{be}\ \underline{e_1}\ \texttt{in}\ \underline{e_2}\ \equiv\ (\lambda x.e_2)e_1$$

Rekurzivní definice

$$\texttt{letrec } \underline{x} \texttt{ be } \underline{e_1} \texttt{ in } \underline{e_2} \equiv (\lambda x.e_2)(Y\ \lambda x.e_1)$$

6.2 Haskell

- seznamy:

$$\begin{array}{rcl} [0,1,2] & :: & [Int] \\ [false, true, true] & :: & [Bool] \\ [\lambda x.x+1, \lambda x.2*x] & :: & [Int \to Int] \\ \left.\begin{array}{l} length\ [\,] = 0 \\ length[x:s] = 1 + length\ s \end{array}\right\} & & \text{má typ } [\alpha] \to Int \\ \alpha & & typová\ proměnná \\ length & & polymorfní\ funkce \end{array}$$

  Seznamy jsou uloženy klasicky:

$$[a,b,c] \equiv a:b:c:[\,]$$

  V seznamu musí mít všechny prvky týž typ

- Funguje tam skript — obsahuje předdefinované operace a funkce:

$$\begin{array}{rcl} (+),(-),(*),(/) & :: & Int \to Int \to Int \\ (<),(>),\ldots & :: & Int \to Int \to Bool \end{array}$$

- Má také nadefinován typ $Char$
- Infix a prefix:

$$\begin{array}{rcl} \texttt{g 0 false} & \equiv & 0\,'g'\,false \\ \texttt{14 + 815} & \equiv & (+)\ 14\ 815 \end{array}$$

- e_1 `where` x=e_2 … když e_2 obsahuje x, je to rekurze
- `let` x=e `in` e' … nerekurzivní
- Operátor rovnosti:

$$(==) \quad :: \quad \alpha \to \alpha \to Bool \quad \forall \alpha \in E_q$$

  (Kvůli vyčíslitelnosti to nejde na funkce)

- Faktoriál:

```
fact 0 = 1
fact n = n * fact (n-1)
```

  Při vyčíslování to zkouší postupně, jestli se to hodí — *pattern matching*.

$$\begin{array}{rcl} fact & = & \lambda m.if \begin{array}{c} match(m,0) \\ m == 0 \end{array} then\ 1 \\ & & else\ if\ match(m,n)\ then\ m * fact(m-1) \end{array}$$

- Minimum:

$$\begin{array}{rcl} min & :: & Int \to Int \to Int \\ min\ x\ y & = & \begin{cases} x, & x < y \\ y, & jinak \end{cases} \end{array}$$

- dvojice, trojice, vícetice: (α, β) — dvojice
- Karifikace:

$$\begin{aligned} curry &:: ((\alpha,\beta) \to \gamma) \to \alpha \to \beta \to \gamma \\ curry\ f\ x\ y &= f(x,y) \end{aligned}$$

  curry karifikuje, tj. z funkce dvojice udělá klasickou funkci dvou argumentů

$$\begin{aligned} uncurry &:: (\alpha \to \beta \to \gamma) \to (\alpha,\beta) \to \gamma \\ uncurry\ f(x,y) &= f\ x\ y \end{aligned}$$

  $\Longrightarrow$ $(uncurry\ min)$ je funkce dvojice $(uncurry\ min)(2,4)$

- Funkce *zip* spojí dva seznamy do jednoho tak, že prvky výsledného seznamu budou dvojice prvků vždy z jedoho a druhého seznamu. Jestliže je jeden ze seznamů delší než druhý, tak se zbytek zahodí.

$$\begin{aligned} zip &:: ([\alpha],[\beta]) \to [(\alpha,\beta)] \\ zip\ ([\,], s) &= [\,] && \text{— zbytek zahodíme} \\ zip\ (s, [\,]) &= [\,] && \text{— zbytek zahodíme} \\ zip\ (x:s, y:t) &= (x,y) : zip\ (s,t) \end{aligned}$$

  Funkce *zip* se dá karifikovat:

$$zip\ (['\check{x}','f\check{n}','b\check{z}'],[4,14,815]) = curry\ zip\ ['\check{x}','f\check{n}','b\check{z}']\ [4,14,815] =$$
$$= [('\check{x}',4),('f\check{n}',14),('b\check{z}',815)]$$

- Funkce *map* přemapuje seznam argumentů jisté funkce do seznamu funkčních hodnot této funkce.

$$\begin{aligned} map &:: (\alpha \to \beta) \to [\alpha] \to [\beta] \\ map\ f\ [\,] &= [\,] \\ map\ f\ (x:s) &= (f\ x) : (map\ f\ s) \end{aligned}$$

- Složení funkcí — *kompozice*

$$\begin{aligned} (.) &:: (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma \\ (f.g)\ x &= f(g\ x) \end{aligned}$$

- Logické funkce *and* a *or* a funkce *any* a *all*

$$\begin{aligned} or, and &:: [Bool] \to Bool \\ and\ [\,] &= true & or\ [\,] &= false \\ and\ (x:s) &= x\ \&\&\ (and\ s) & or\ (x:s) &= x\ \|\ (or\ s) \end{aligned}$$

$$\begin{aligned} any, all &:: (\alpha \to Bool) \to [\alpha] \to Bool \\ any\ p\ [\,] &= false & all\ p\ [\,] &= true \\ any\ p\ (x:s) &= p\ x\ \|\ any\ p\ s & all\ p\ (x:s) &= p\ x\ \&\&\ all\ p\ s \end{aligned}$$

  Vztah mezi funkcemi *any*,*or* a *and*,*all*:

$$\begin{aligned} any\ p\ s &= or\ (map\ p\ s) & any\ p &= or.map\ p \\ all\ p\ s &= and\ (map\ p\ s) & all\ p &= and.map\ p \end{aligned}$$

- Možnost práce s datovými strukturami

$$\begin{aligned} ones &= 1 : ones \\ ones &= [1,1,1,1,1,1,1,1,1,1,1,1,1 \ldots] \end{aligned}$$

  vyhodnocení *ones* nikdy neskončí

Funkce $take$:

$$\begin{array}{rcl} take & :: & Int \to [\alpha] \to [\alpha] \\ take\ 0\ s & = & [\,] \\ take\ n\ [\,] & = & [\,] \\ take\ n\ (x:s) & = & x : take\ (n-1)\ s \end{array}$$

$take$ vezme prvních několik ze seznamu

Díky línému vyhodnocování lze vyhodnotit $take$ 14 $ones$

Funkce $filter$:

$$\begin{array}{rcl} filter & :: & (\alpha \to Bool) \to [\alpha] \to [\alpha] \\ filter\ f\ [\,] & = & [\,] \\ filter\ f\ (x:s) & = & \begin{cases} x : filter\ f\ s, & f\ x \\ filter\ f\ s, & jinak \end{cases} \end{array}$$

Operátor ...:

$$\begin{array}{rcl} [1\ldots] & \equiv & [1,2,3,4,5,6,7,8,9,10,11,12,13,14,atd.] \\ [8,12\ldots] & \equiv & [8,12,13,14,atd.] \end{array}$$

Operátor !! bere n–tý prvek seznamu

$$\begin{array}{rcll} (!!) & :: & [\alpha] \to Int \to \alpha & \\ (x:s)\ !!\ 0 & = & x & \text{první je nultý} \\ (x:s)\ !!\ n & = & s\ !!\ (n-1) & \end{array}$$

- Příklad — seznam všech prvočísel

```
primes = si [2...] where si[] = []
    si (a:s) = a : filter (λn→n mod a /= 0) (si s)
```

- ϱ termy jsou vnitřně reprezentovány jako graf a redukují se jako redukce grafu

7 Polymorfní λ–kalkul (λ–kalkul 2.řádu)

7.1 Curry—Howardův isomorfismus

Typy Máme spočtenu množinu typových proměnných $\alpha_1, \alpha_2, \ldots$

(i). Každá typová proměnná je typ

(ii). Jsou-li σ, τ typy $\Longrightarrow \sigma \to \tau$ je typ (funkce)

(iii). Je-li σ typ, α typová proměnná $\Longrightarrow \forall\alpha.\sigma$ je typ $(\Pi\alpha.\beta)$
... třeba $if : \forall\alpha.Bool \to \alpha \to \alpha \to \alpha$ (čili funkce, která pro typ vrátí typ)

Termy Máme spočtenu množinu proměnných, a to $\forall$ typ σ $x_1, x_2, \ldots$

(i). Pro každou proměnnou typu σ, každý term M typu σ ... značíme $M : \sigma$

(ii). $x : \sigma$ je proměnná, $M : \tau$ je term $\Longrightarrow \lambda x : \sigma.M$ je term typu $\sigma \to \tau$

(iii). $M : \sigma \to \tau$, $N : \sigma$ jsou termy $\Longrightarrow MN : \tau$ je term

(iv). $M : \sigma$ je term, α je typová proměnná $\Longrightarrow \Lambda\alpha.M : \forall\alpha.\sigma$ je term

(v). $M : \forall\alpha.\sigma$ je term, τ je typ $\Longrightarrow M\tau : \sigma[\tau/\alpha]$ je term

Redukce

(i). $\lambda x:\sigma.M \to^{\alpha} \lambda y:\sigma.M[y/x]$ ... α–redukce (přejmenování proměnných)

(ii). $(\lambda x:\sigma.M)N \to^{\beta} M[N/x]$ ... β–redukce (nejdůležitější)

(iii). $\lambda x:\sigma.Mx \to^{\eta} M$

(iv). $\Lambda\alpha:M \to^{A} \Lambda\beta.M[\beta/\alpha], \quad M:\sigma, \quad \beta \notin FV(\sigma)$

(v). $(\Lambda\alpha.M)\tau \to^{B} M[\tau/\alpha], \quad M:\sigma, \quad M[\tau/\alpha]:\sigma[\tau/\alpha]$

(vi). $\Lambda\alpha.M\alpha \to^{H} M$

(vii). $\hookrightarrow$ je kompatibilní uzávěr relací $\to^{\alpha}, \to^{\beta}, \to^{\eta}, \to^{A}, \to^{B}, \to^{H}$. Uzávěr kompatibilní na tvoření podtermů... je-li $M \hookrightarrow N$, pak i pro kontexty $C[M] \hookrightarrow C[N]$

(viii). $\hookrightarrow^{*}$ je reflexivní a tranzitivní uzávěr $\hookrightarrow$

Věta Relace $\hookrightarrow$ je CR i SN
(Všechny termy polymorfního λ–kalkulu jsou silně normalizovatelné a normalizovatelné formy jsou určeny, tzn...)

7.2 Reprezentace typů

Lemma Jediné uzavřené normalizovatelné formy typu $\sigma \to \tau$, resp. $\forall\alpha.\sigma$ mají tvar $\lambda x:\sigma N$, resp. $\Lambda\alpha.N$.

7.2.1 Typ Boolean

$Bool = \forall\alpha.\alpha \to \alpha \to \alpha$

Důsledek Typ *Bool* má právě dvě uzavřené normalizovatelné formy, ty potom nazveme *true* a *false*.

$$\left.\begin{array}{r}true = \Lambda\alpha.\lambda x:\alpha.\lambda y:\alpha.x\\ false = \Lambda\alpha.\lambda x:\alpha.\lambda y:\alpha.y\end{array}\right\}\text{ jediné dvě norm. formy typu } Bool$$

$if = \Lambda\alpha.\lambda x:Bool.\lambda y:\alpha.\lambda z:\alpha.x\;\alpha\;y\;z$
$M:\sigma, \quad N:\sigma$

if aplikuju na $\sigma \to if[\sigma/\alpha]$

$$\begin{aligned}if\sigma\;true\;M\;N &\hookrightarrow^{\beta} (\lambda x:Bool.\lambda y:\sigma.\lambda z:\sigma.x\,\sigma\,y\,z)\;true\;M\;N \hookrightarrow^{\beta}\\ &\hookrightarrow^{\beta} (\lambda y:\sigma.\lambda z:\sigma.true\,\sigma\,y\,z)\;M\;N \hookrightarrow^{\beta}\\ &\hookrightarrow^{\beta} (\lambda z:\sigma.true\,\sigma\,M\,z)\;N \hookrightarrow^{\beta}\\ &\hookrightarrow^{\beta} true\;\sigma\;M\;N \hookrightarrow^{\beta}\\ &\hookrightarrow^{\beta} (\lambda x:\sigma.\lambda y:\sigma.x)\;M\;N \hookrightarrow^{\beta}\\ &\hookrightarrow^{\beta} (\lambda y:\sigma.M)\;N \hookrightarrow^{\beta} M\end{aligned}$$

7.2.2 Typ Natural

$Nat = \forall\alpha.\alpha \to (\alpha \to \alpha) \to \alpha$
termy: $\overline{n} = \Lambda\alpha.\lambda x:\alpha.\lambda y:\alpha \to \alpha.\underbrace{y(y(\ldots(y\,x)\ldots))}_{n-krát}$

Důsledek Jediné uzavřené nf typu *nat* jsou tvaru $\overline{n}$ pro $n \in \mathcal{N}$

$\Lambda\alpha.\lambda x:\alpha.\lambda y:\alpha \to \alpha\;.x = \overline{0}$
$.yx = \overline{1}$ atd.

$succ = \lambda z:Nat.\Lambda\alpha.\lambda x:\alpha.\lambda y:\alpha \to \alpha.y(z\,\alpha\,x\,y)$
$succ\;\overline{0} \hookrightarrow^{*} \overline{1}$

$succ\;\overline{2} \hookrightarrow \Lambda\alpha.\lambda x:\alpha.\lambda y:\alpha \to \alpha.y((\lambda x:\alpha.\lambda y:\alpha \to \alpha.y(y\,x))x\,y)$
$\ldots \hookrightarrow \Lambda\alpha.\lambda x:\alpha.\lambda y:\alpha \to \alpha.y(y(y\,x)) \;=\; \overline{3}$

7.2.3 Kartézský součin

$\sigma \ \times \ \tau \ = \ \forall\alpha.(\sigma \to \tau \to \alpha) \to \alpha$

Definice (term typu $\sigma \times \tau$ - dvojice)
Jsou-li $M : \sigma$, $N : \tau$ termy, pak definujeme:

$$\begin{aligned} (M, N) \ &= \ \Lambda\alpha.\lambda x : \sigma \to \tau \to \alpha.x \ M \ N \\ fst \ &= \ \Lambda\sigma.\Lambda\tau.\lambda p : \sigma \times \tau.p \ \sigma \ (\lambda x : \sigma.\lambda y : \tau.x) \\ snd \ &= \ \Lambda\sigma.\Lambda\tau.\lambda p : \sigma \times \tau.p \ \tau \ (\lambda x : \sigma.\lambda y : \tau.y) \end{aligned}$$

$fst \ \sigma \ \tau(M, N) \ \hookrightarrow (M, N) \ \sigma \ (\lambda x : \sigma.\lambda y : \tau.x) \ \hookrightarrow$
$\hookrightarrow \ (\lambda x : \sigma \to \tau \to \sigma.x \ M \ N) \ (\lambda x : \sigma.\lambda y : \tau.x) \ \hookrightarrow$
$\hookrightarrow \ (\lambda x : \sigma.\lambda y : \tau \ x) \ M \ N \ \hookrightarrow \ (\lambda y : \tau.M) \ N \ \hookrightarrow \ M$

$snd \ \sigma \ \tau(M, N) \ \hookrightarrow$
... protože snd je polymorfní, musí se to aplikovat na ty typy
$\hookrightarrow \ (\lambda p : \sigma \times \tau.p \ \tau \ (\lambda x : \sigma.\lambda y : \tau.y)) \ (M, N) \ \hookrightarrow$
... toto je snd aplikováno na σ a τ; teď aplikovat na (M, N)
$\hookrightarrow \ (M, N) \ \tau \ (\lambda x : \sigma.\lambda y : \tau.y) \ \hookrightarrow \ (\lambda x : \sigma \to \tau \to \tau.x \ M \ N) \ (\lambda x : \sigma.\lambda y : \tau.y) \ \hookrightarrow$
... $(M, N) \ = \ \Lambda\alpha$: atd. aplikováno na τ
$\hookrightarrow \ (\lambda x : \sigma.\lambda y : \tau.y) \ M \ N \ \hookrightarrow \ (\lambda y : \tau.y) \ N \ \hookrightarrow \ N$

Tedy:

$$\begin{aligned} fst \ \sigma \ \tau \ (M, N) \ &\hookrightarrow^* \ M \\ snd \ \sigma \ \tau \ (M, N) \ &\hookrightarrow^* \ N \end{aligned}$$

... a je dokázána správnost definice fst a snd

Chceme ještě, aby $(fst \ \sigma \ \tau \ P, \ snd \ \sigma \ \tau \ P) \ \hookrightarrow^* \ P$, ale to nejde ani pomocí η– a H–redukce. Totiž ono se to redukuje na (M, N) a to je to stejné co P.

7.2.4 Seznamy

K typu σ definujeme typ $List \ \sigma$. Termy jsou posloupnosti prvků (ne nutně konečné) typu σ.

Např.: $[1, 2, 3, 4] : \ List \ Nat, \quad [false] : \ List \ Bool$

Definice $List \ \sigma \ = \ \forall\alpha.\alpha \to (\sigma \to \alpha \to \alpha) \to \alpha$

Prázdný seznam: $nil \ = \ \Lambda\sigma.\Lambda\alpha.\lambda x : \alpha.\lambda y : \sigma \to \alpha \to \alpha.x$
... píše se $[\,]$; σ je polymorfní
$\alpha \ \simeq$ typ seznamu
$\sigma \ \simeq$ typ prvků
$x \ \simeq$ term seznamu
$y \ \simeq$ funkce, která seznam vytváří

$Cons$ ... píše se $\mu : t$ (head:tail)
$Cons \ = \ \Lambda\sigma.\lambda\mu : \sigma.\lambda t : List \ \sigma.\Lambda\alpha.\lambda x : \alpha.\lambda y : \sigma \to \alpha \to \alpha.y \ \mu \ (t \ \alpha \ x \ y)$
Typ nil : $\quad nil : \forall\sigma.List \ \sigma \quad$ (polymorfní)
Typ $cons$: $\quad cons : \forall\sigma.\sigma \to List \ \sigma \to List \ \sigma$
$[\mu_1, \mu_2, \ldots, \mu_n] \ = \ cons \ \sigma \ \mu_1(cons \ \sigma \ \mu_2(\ldots(cons \ \sigma \ \mu_n(nil \ \sigma))\ldots))$
$(\mu_1, \mu_2, \ldots, \mu_n : \sigma) \ = \ (\hookrightarrow^*) \ \Lambda\alpha.\lambda x : \alpha.\lambda y : \sigma \to \alpha \to \alpha.y \ \mu_1(y \ \mu_2(\ldots(y \ \mu_n \ x)\ldots))$
... a to je typu: $\forall\alpha.\alpha \to (\sigma \to \alpha \to \alpha) \to \alpha \ = \ List \ \sigma$

7.3 Kombinátor pevného bodu

$Y : \forall\sigma.(\sigma \to \sigma) \to \sigma$
$M : \sigma \to \sigma$, pak $Y\ \sigma\ M \ \hookrightarrow^{\delta}\ M\ (Y\ \sigma\ M)$
... tím přestává platit silná normalizovanost, ale máme rekurzi

data $\alpha_1, \ldots, \alpha_n\ =\ constr_1 \mid \ldots \mid constr_n$
data $List\ \alpha\ =\ Nil \mid Cons\ \alpha\ (List\ \alpha) \ldots (cons\ \mu_1 \mid cons\ \mu_2\ Nil) :\ List\ \alpha$

A Otázky ke zkoušce

1. Netypovaný λ–kalkul
2. Redukce, konfluence (CR), silná normalizovatelnost (SN)
3. Typy, typovaný λ–kalkul, redukce
4. Kombinátor pevného bodu, rekurze
5. Pořadí vyhodnocování, striktní a líná redukce
6. Sémantika: operační a denotační
7. Adekvátnost operační a denotační sémantiky
8. Plná abstrakce, paralelní $\underline{or}$
9. Implementace funkcionálních jazyků v λ–kalkulu (definice včetně rekurzivních, let, $where$)
10. Polymorfní λ–kalkul
11. Vyjádření základních typů ($Bool$, Nat, $List$, $\times$) a základních operací v polymorfním λ–kalkulu
12. Datové typy v polymorfních jazycích

B Jazyk `GOFER`

Na počítači `adelard.dcs.muni.cz` je k dispozici jazyk `GOFER`, což je nemodulární podmnožina Haskellu.

- **adresář** `/usr/local/bin` nebo `/usr/local`
- **.profile:** přidat řádek `export GOFER=/usr/local/lib/prelude`
- **adelard.p14$** `gofer`
- pak se napíše **výraz** za ?

`(&&)`	`::`	`Bool→Bool→Bool`
`False && x`	`=`	`False`
`True && x`	`=`	`x`
`A where x = B`		A s dosazením za volnou proměnnou
	`(x+4)*(x+14) where x=815`	

- **funkce** se dají zapisovat infixově i prefixově:

f	$:: \alpha \to \beta \to \gamma$		... zápis typu funkce
f	A	B	... prefix
A	$'f'$	B	... infix
$(+)$	A	B	... prefix pro operátory

- **dokumentace:** `/ftp/pub/incoming/gofer.zip`

— Konec —