Zkouška I014

15. června 2000

Jméno a příjmení: ....

UČO: .... *login:* ....

1

Typ definovaný

```
data Bitlist = Nil | ConsOn Bitlist | ConsOff Bitlist
```

lze v impredikativním typovém systému polymorfního lambda kalkulu vyjádřit

(A) $\texttt{Bitlist} = \forall\tau.\tau \to (\tau \to \tau) \to \tau$ **(B)** $\texttt{Bitlist} = \forall\tau.\tau \to (\tau \to \tau) \to (\tau \to \tau) \to \tau$

(C) $\texttt{Bitlist}\ \alpha = \forall\tau.\tau \to (\alpha \to \tau) \to (\alpha \to \tau) \to \tau$ **(D)** $\texttt{Bitlist}\ \alpha = \forall\tau.(\tau \to \tau) \to (\alpha \to \tau) \to \tau$

(E) $\texttt{Bitlist} = \forall\tau.\tau \to (\tau \to \tau) \to (\tau \to \tau)$

Odpověď: ☐

2

Máme dány konstanty $0 :: \mathsf{Nat}$, $\mathsf{succ} :: \mathsf{Nat} \to \mathsf{Nat}$. Při odvozování typu výrazu

$$\mathsf{let}\ \ f = \lambda x \lambda y.x(x\ y)\ \ \mathsf{in}\ \ f\ f\ \mathsf{succ}\ 0$$

s využitím typového kontextu $\Delta = \{f :: \forall\alpha.(\alpha \to \alpha) \to \alpha \to \alpha\}$ použijeme na dvou místech pravidlo (SPEC). V těchto dvou místech jsou použity substituce

(A) $[\mathsf{Nat}/\alpha]$, $[(\mathsf{Nat} \to \mathsf{Nat}) \to \mathsf{Nat}/\alpha]$

(B) $[((\mathsf{Nat} \to \mathsf{Nat}) \to \mathsf{Nat} \to \mathsf{Nat}) \to \mathsf{Nat} \to \mathsf{Nat}/\alpha]$, $[\mathsf{Nat} \to \mathsf{Nat}/\alpha]$

(C) $[\mathsf{Nat}/\alpha]$, $[\mathsf{Nat} \to \mathsf{Nat}/\alpha]$

(D) $[\mathsf{Nat} \to \mathsf{Nat}/\alpha]$, $[(\mathsf{Nat} \to \mathsf{Nat}) \to \mathsf{Nat} \to \mathsf{Nat}/\alpha]$

(E) $[\mathsf{Nat}/\alpha]$, $[(\mathsf{Nat} \to \mathsf{Nat}) \to \mathsf{Nat} \to \mathsf{Nat}/\alpha]$

Odpověď: ☐

3

V Haskellu máme definovanou funkci

```
from :: Int → [Int]
from n = n : from (succ n)
```

Tuto funkci lze pomocí kombinátoru pevného bodu Y vyjádřit

(A) $\texttt{from} = \mathsf{Y}(\lambda f \lambda n.\ f(n\ :\ \texttt{succ}\ n))$ **(B)** $\texttt{from} = \mathsf{Y}(\lambda f \lambda n.\ f\ n\ :\ f(\texttt{succ}\ n))$

(C) $\texttt{from} = \mathsf{Y}(\lambda f \lambda n.\ n\ :\ f \circ \texttt{succ})$ **(D)** $\texttt{from} = \mathsf{Y}(\lambda f \lambda n.\ n\ :\ f \circ \texttt{succ})\ n$

(E) $\texttt{from} = \mathsf{Y}(\lambda f \lambda n.\ n\ :\ (f \circ \texttt{succ})\ n)$

Odpověď: ☐

4

Máme term

$$\lambda x \lambda y \lambda z.\mathsf{f}\ y\ (\lambda t.t\ \mathsf{g}\ x)\ (\lambda u.u\ (\mathsf{g}\ y))$$

v němž f, g jsou konstanty. Tento term převedeme do superkombinátorového termu tak, že kromě nových superkombinátorů dovolíme použít i kombinátory S a K. Při převodu smíme provádět libovolné ekvivalentní úpravy. Nejmenší počet nových superkombinátorů, které musíme zavést, je

(A) 1 **(B)** 2 **(C)** 4 **(D)** 3 **(E)** 0

Odpověď: ☐

5

Mějme term $\lambda x \lambda y.\mathsf{F}(G\ x\ x)(\mathsf{F}\ y)$ s konstantami F, G. V něm maximální volný term abstrakce λy je

(A) $\mathsf{F}\ (\mathsf{G}\ x\ x)$ **(B)** $\mathsf{F}\ (\mathsf{G}\ x\ x)\ \mathsf{F}$ **(C)** F **(D)** $\mathsf{G}\ x\ x$ **(E)** $\mathsf{G}\ x\ x\ (\mathsf{F}\ y)$

Odpověď: ☐

6

Kombinátor Φ, který je definován δ-pravidlem

$$\Phi\ x\ y\ z\ u \ \rightsquigarrow\ x(y\ z\ u)$$

je ekvivalentní kombinátorovému termu

(A) $\mathsf{B\ B\ B}$ **(B)** $\mathsf{B\ B}$ **(C)** B **(D)** $\mathsf{B\ (C\ B\ B)\ (B\ B\ B)}$ **(E)** $\mathsf{C\ (B\ B(B\ B\ B))\ B)}$

Odpověď: ☐

7

V následujícím prográmku na syntaktickou analýzu lambda-termů se za „atomické" považují ty termy (`atomy`), které jsou proměnné, abstrakce nebo jsou uzavřeny v závorkách. Obecný lambda-term (`term`) však může být i aplikací několika atomických termů zapsaných za sebou, například `(\x.x y) y (\y.z)`. Napište definici parseru `term`, která je v ukázce vynechaná.

```
data Term = Var Char | App Term Term | Lam Char Term

term :: Parser Term
term = ...

atom :: Parser Term
atom = var +++ lam +++ paren

var :: Parser Term
var = variable >>= return . Var

lam :: Parser Term
lam = symbol "\\" >> variable >>= \x -> symbol "." >> term >>= return . Lam x

paren :: Parser Term
paren = bracket (symbol "(") term (symbol ")")

variable :: Parser Char
variable = token ident >>= return . head
```