Písemná zkouška – I014

21. června 1999

Jméno a příjmení: .....

login: .....

1

Jsou dány typové výrazy

$$a \to (b \to \mathsf{Nat}) \to b$$

$$(\mathsf{Bool}, c, b) \to c$$

s typovými proměnnými a, b, c. Jejich nejobecnější unifikátor je

(A) $[(b \to \mathsf{Nat}) \to b/c] \circ [(\mathsf{Bool}, c, b)/a]$ **(B)** $\bot$ (Typové výrazy nejsou unifikovatelné)

(C) $[(\mathsf{Bool}, c, b)/a] \circ [(b \to \mathsf{Nat}) \to b/c]$ **(D)** $[(\mathsf{Bool}, c, b)/a] \circ [\mathsf{Nat}/b] \circ [(b \to \mathsf{Nat}) \to b/c]$

(E) $[(\mathsf{Bool} \to \mathsf{Nat}) \to \mathsf{Bool}/c] \circ [\mathsf{Bool}/b] \circ [(\mathsf{Bool}, c, b)/a]$

Odpověď: ☐

2

Při převedení termu

$$\lambda x \lambda y.\mathsf{F}\,(\lambda z.\mathsf{F}\ x\ z)(\lambda z.\mathsf{F}\ z\ z\ y)(\lambda z.\mathsf{F}\ z\ z\ x)$$

do superkombinátorového termu

(A) jsou potřeba 3 superkombinátory **(B)** jsou potřeba 4 superkombinátory

(C) je potřeba aspoň 5 superkombinátorů **(D)** obejdeme se bez superkombinátorů

(E) stačí definovat 2 superkombinátory

Odpověď: ☐

3

Kombinátor Θ, který je definován δ-pravidlem

$$\Theta\ x\ y\ z \quad \rightsquigarrow \quad x\ z\ z\ y$$

je ekvivalentní kombinátorovému termu

(A) B C (B (C S I) (C I)) **(B)** C (B B S) (C I) **(C)** C(B B B) (C (S I I))

(D) C (B B B) ((B(S I))(C I)) **(E)** B C (C S I)

Odpověď: ☐

4

Typ definovaný

```
data Either a b = Left a | Right b
```

lze v impredikativním typovém systému polymorfního lambda kalkulu vyjádřit

(A) `Either` $\alpha\ \beta = \forall\tau.(\alpha \to \beta \to \tau) \to (\beta \to \alpha \to \tau) \to \tau$ **(B)** `Either` $= \forall\alpha\forall\beta\forall\tau.(\alpha \to \tau) \to (\beta \to \tau) \to \tau$

(C) `Either` $= \forall\alpha\forall\beta\forall\tau.\alpha \to \beta \to \tau$ **(D)** `Either` $\alpha\ \beta = \forall\tau.(\alpha \to \tau) \to (\beta \to \tau) \to \tau$

(E) `Either` $\alpha\ \beta = \forall\tau.\alpha \to \beta \to \tau$

Odpověď:

5

Máme typ `[Int]` seznamů celých čísel (s konstruktory `[]` a `(:)`) a je dán typ binárních stromů s uzly ohodnocenými seznamy celých čísel:

```
data Tree = Empty | Node [Int] Tree Tree
```

Dále máme definován nekonečný strom t, který má v uzlech konečné seznamy nul a jedniček:

```
t = f [] where f s = Node s (f (0:s)) (f (1:s))
```

Strom t lze zapsat pomocí kombinátoru pevného bodu Y takto:

(A) $t = \mathsf{Y}\big((\lambda s.\ \texttt{Node}\ s\ (0{:}s)\ (1{:}s))\ []\big)$ **(B)** $t = \mathsf{Y}\big((\lambda f\lambda s.\ \texttt{Node}\ s\ (f(0{:}s))\ (f(1{:}s)))\ []\big)$

(C) $t = \mathsf{Y}\big(\lambda f\lambda s.\ \texttt{Node}\ s\ (f(0{:}s))\ (f(1{:}s))\big)$ **(D)** $t = \mathsf{Y}\big(\lambda s.\ \texttt{Node}\ s\ (0{:}s)\ (1{:}s)\big)$

(E) $t = \mathsf{Y}\big(\lambda f\lambda s.\ \texttt{Node}\ s\ (f(0{:}s))\ (f(1{:}s))\big)\ []$

Odpověď:

6

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

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

s využitím typového kontextu

$$\Delta = \{f :: \forall\alpha\forall\beta.\alpha \to (\mathsf{Nat} \to \alpha \to \beta) \to \beta\}$$

použijeme na dvou místech odvození vždy dvojici pravidel (SPEC). V těchto dvou dvojicích pravidel jsou použity substituce

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

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

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

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

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

Odpověď:

7

Původní modul `Frequency` (viz modul) z programu realizujícího Huffmanovo kódování byl změněn takto:

```
module Frequency ( frequency ) where


frequency :: [Char] -> [ (Char,Int) ]
frequency = quickSort sndLess . mrg . quickSort fstLess . map start
             where start ch = (ch,1)
                   fstLess u v = fst u < fst v
                   sndLess u v = snd u < snd v


quickSort :: Ord a => (a->a->Bool) -> [a] -> [a]
quickSort _ [] = []
quickSort less (p:s) = quickSort less [ x | x<-s, x `less` p ]
                       ++ [p]
                       ++ quickSort less [ x | x<-s, not (x `less` p) ]
```

Doplňte definici funkce `mrg`.

```
------------------------------------------------------------------------
--      Frequency.hs                                                  --
------------------------------------------------------------------------

module Frequency ( frequency ) where

-- Četnosti znaků v textu
frequency :: [Char] -> [ (Char,Int) ]
frequency
  = mergeSort freqMerge . mergeSort alphaMerge . map start
    where
    start ch = (ch,1)


-- Třídění slučováním parametrizované operací merge

mergeSort :: ([a]->[a]->[a]) -> [a] -> [a]
mergeSort merge x = if length x<2
                       then x
                       else merge (mergeSort merge first)
                                  (mergeSort merge second)
                    where first = take half x
                          second = drop half x
                          half = (length x) `div` 2



-- slučování dvojic podle znaků s případným spojením dvou dvojic
-- do jedné při stejných znacích v první složce

alphaMerge x [] = x
alphaMerge [] y = y
alphaMerge ((a,n):x) ((b,m):y)
  | (a==b)      = (a,n+m) : alphaMerge x y
  | (a<b)       = (a,n) : alphaMerge x ((b,m):y)
  | otherwise   = (b,m) : alphaMerge ((a,n):x) y


-- slučování dvojic podle lexikografického pořadí, kde větší váhu
-- mají četnosti, menší váhu mají znaky

freqMerge x [] = x
freqMerge [] y = y
freqMerge ((a,n):x) ((b,m):y)
  | (n<m || (n==m && a<b))
    = (a,n) : freqMerge x ((b,m):y)
  | otherwise
    = (b,m) : freqMerge ((a,n):x) y
```