Exame de Implementação de Linguagens

Departamento de Ciência de Computadores
Faculdade de Ciências – Universidade do Porto
5 de Setembro de 2011
Duração: 2 horas

Parte I

(1) Considere o programa Prolog que se segue:

```
p(step(stop,Y),_).
p(step(X,Y),context(Z)) :- do(X,Z,W), p(step(Y,W),context(Z)).
```

(a) Escreva o código compilado do programa no contexto da máquina $\mathcal{M}_3$ da WAM.

(b) Considere as seguintes optimizações à WAM: (i) tratamento de constantes; (ii) tratamento de variáveis anónimas; (iii) melhor alocação de registos; (iv) *last call optimization*. Escolha duas destas optimizações, diga sucintamente no que cada uma consiste e reescreva o código anterior de forma a incluir essas optimizações.

(2) A técnica de *tabulação* pode ser implementada utilizando diferentes *modelos de execução*. Os dois modelos de execução mais conhecidos são a *tabulação por suspensão da computação* e a *tabulação linear*. Descreva de forma sucinta em que consistem ambos os modelos e enumere as suas principais diferenças, vantagens e/ou desvantagens.

Parte II

(3) Considere o seguinte termo da linguagem FUN:

$$\text{let } f = \lambda x.\, 3 * x + 1 \text{ in } f\ 5$$

(a) Baseando-se no esquema de compilação em anexo, traduza este termo para código da máquina SECD.

(b) Simule a execução passo-a-passo do código SECD, indicando quais os valores na pilhas de execução.

(4) Considere a definição em Haskell da função que calcula o comprimento duma lista:

```
length :: [a] -> Int
length [] = 0
length (x:xs) = 1 + length xs
```

Escolha uma codificação para listas usando *pares* e *variantes* e traduza esta definição num termo na linguagem FUN com estas extensões. Justifique a sua resposta.

(5) Considere a definição do prelúdio *standard* de Haskell da função *iterate* que, dada uma função f e um ponto x, constroi a lista infinita dos iterandos x, $f\,x$, $f\,(f\,x)$, ...

```
iterate          :: (a -> a) -> a -> [a]
iterate f x      =  x : iterate f (f x)
```

Traduza esta definição para a linguagem da máquina STG. Justifique a sua resposta.

Anexo

Esquema de compilação para SECD

```
compile :: Term -> [Var] -> Code
compile (Var x) sym
    = case elemIndex x sym of
        Nothing -> error ("unbound identifier: "++show x)
        Just k -> [LD k]
compile (Lambda x e) sym = [LDF (compile e (x:sym) ++ [RTN])]
compile (App e1 e2) sym = compile e1 sym ++ compile e2 sym ++ [AP]
compile (Const n) sym = [LDC n]
compile (e1 :+ e2) sym = compile e1 sym ++ compile e2 sym ++ [ADD]
compile (e1 :- e2) sym = compile e1 sym ++ compile e2 sym ++ [SUB]
compile (e1 :* e2) sym = compile e1 sym ++ compile e2 sym ++ [MUL]
compile (e1 :== e2) sym = compile e1 sym ++ compile e2 sym ++ [EQU]
compile (e1 :<= e2) sym = compile e1 sym ++ compile e2 sym ++ [LTE]
compile (IfThenElse e1 e2 e3) sym
    = compile e1 sym ++ [SEL (compile e2 sym ++ [JOIN])
                             (compile e3 sym ++ [JOIN])]
compile (Let x e1 e2) sym
    = compile (App (Lambda x e2) e1) sym
compile (Fix (Lambda f (Lambda x e1))) sym
    = [LDRF (compile e1 (x:f:sym) ++ [RTN])]
```