Exame de Implementação de Linguagens

Departamento de Ciência de Computadores
Faculdade de Ciências – Universidade do Porto
29 de Junho de 2011
Duração: 2 horas

Parte I

(1) Considere o programa Prolog que se segue:

```
path(node(X),node(Z)):- node_edge(X,Z).
path(node(X),node(Z)):- path(node(X),node(Y)), node_edge(Y,Z).
```

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

(b) A estratégia de resolução SLD, na qual o Prolog se baseia, é limitadora do potencial inerente ao paradigma da Programação em Lógica. Identifique no programa acima qual o problema que a resolução SLD não consegue tratar devidamente e indique que tipo de técnica poderia ser utilizada para solucionar esse problema. Descreva de forma sucinta no que consiste essa técnica e indique quais são as principais estensões de suporte à sua implementação numa máquina WAM.

Parte II

(2) Considere o seguinte termo da linguagem FUN:

$$\text{let } f = (\text{let } y = 3 \text{ in } \lambda x.\, x + y) \text{ in } f\ 5$$

(a) Usando o 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.

(3) Pretende-se acrescentar à linguagem FUN dois operadores boleanos de conjunção e disjunção (`&&`, `||`) com semântica não-estrita, isto é, em que o segundo argumento não é avaliado se o resultado for determinado apenas pelo primeiro argumento.

(a) Descreva as modificações à sintaxe abstrata da linguagem e à semantica operacional natural (defina apenas as regras de avaliação para os novos operadores).

(b) Não é necessário acrescentar instruções à maquina SECD apresentada nas aulas para implementar estes dois novos operadores. Justifique a afirmação anterior descrevendo as modificações ao esquema de tradução para máquina SECD para os dois operadores.

(4) Considere a definição do prelúdio-padrão de Haskell da função *filter*:

```
filter :: (a->Bool) -> [a] -> [a]
filter f []    = []
filter f (x:xs) | f x       = x : filter f xs
                | otherwise = filter f xs
```

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])]
```