Exame de Implementação de Linguagens

Departamento de Ciência de Computadores
Faculdade de Ciências – Universidade do Porto
25 de Junho de 2014
Duração: 2 horas

Parte I

(1) A compilação dum termo da linguagem FUN da forma fix $\lambda f.\, \lambda x.\, M$ para a máquina SECD resulta na seguinte sequência de código:

```
[LDRF [LD 0,SEL [LDC 0,JOIN] [LD 0,LD 0,MUL,LD 1,LD 0,LDC 1,SUB,AP,ADD,JOIN],RTN]]
```

Re-construa o fragmento M em falta no termo original, justificando a sua resposta usando o esquema de compilação da FUN para SECD.

(2) A linguagem FUN apresentada nas aulas tem apenas uma expressão condicional para testar se uma expressão e_0 é zero ou não cuja sintaxe abstrata em Haskell é "`IfZero` e_0 e_1 e_2".

Pretende-se acrescentar à linguagem uma nova expressão condicional "`IfSign` e_0 e_1 e_2 e_3" para discriminar o *sinal* duma expressão e_0 em três casos:

positivo: e nesse caso o resultado será e_1;

zero: o resultado será e_2;

negativo: o resultado será e_3.

Mostre como implementar a compilação destas expressões acrescentando para tal uma nova instrução de seleção à máquina SECD.

Sugestão: defina o esquema de compilação de expressões `IfSign` e a execução da nova instrução acrescentando casos às funções `compile` e `execute`, respectivamente.

(3) Considere a seguinte definição em Haskell duma função recursiva que duplica elementos numa lista; o caso base é a lista com apenas um elemento que é transformada na lista vazia.

```
f [x] = []
f (x:xs) = x : x : f xs
```

Traduza esta definição para a linguagem da máquina STG. Justifique a sua resposta, em particular a escolha de *flags* de *update* nas expressões *lambda-form*.

Parte II

(4) No contexto da máquina $\mathcal{M}_3$ da WAM, considere o código WAM que se segue:

```
p/2: try_me_else L2
     allocate 0
     get_structure entry/1, A1
     unify_variable X3
     get_structure entry/1, A2
     unify_variable X4
     put_value X3, A1
     put_value X4, A2
     call q/2
     deallocate
```

```
L2: trust_me
    allocate 2
    get_structure entry/1, A1
    unify_variable X3
    get_structure entry/1, A2
    unify_variable Y1
    put_structure entry/1, A1
    set_value X3
    put_structure entry/1, A2
    set_variable Y2
    call p/2
    put_value Y2, A1
    put_value Y1, A2
    call q/2
    deallocate
```

(a) Escreva o programa Prolog correspondente ao código WAM apresentado.

(b) Considere as seguintes otimizações à WAM: (i) tratamento de constantes; (ii) tratamento de variáveis anónimas; (iii) melhor alocação de registos; (iv) *last call optimization*. Reescreva o código anterior de forma a tirar partido dessas otimizações (indique de forma clara onde cada otimização se encontra no novo código).

(5) Considere o seguinte programa Prolog em que *p/2* é um predicado tabelado:

```
:- table p/2.
p(A,B) :- e(A,X), p(X,B).
p(A,B) :- e(A,B).

e(a,b).
e(b,c).
e(c,b).

output([A]) :- !, write(A), nl.
output([H|T]) :- write(H), write('-'), output(T).
```

Considere ainda a seguinte consulta:

```
?- p(X,Y), output([X,Y]), fail.
```

(a) Indique quais as linhas de *output* (independente da ordem) obtidas pela execução da consulta.

(b) Justifique qual seria a primeira linha de *output* obtida pela execução da consulta caso *p/2* não fosse declarado como um predicado tabelado.

(c) Em tabulação, um conjunto de objetivos tabelados pode ser mutuamente dependente (*strongly connected component (SCC)*) e nesse caso só pode ser completo em simultâneo. Justifique qual a sequência de SCCs que a execução da consulta dá origem (por ordem temporal) e para cada SCC identifique o respetivo líder.

(6) Uma das principais características do Prolog é o seu potencial para a exploração *implícita* de paralelismo. Diga o que entende por paralelismo implícito, e explique quais são as duas principais formas de paralelismo implícito existentes em Prolog.