Exame de Implementação de Linguagens

Departamento de Ciência de Computadores
Faculdade de Ciências – Universidade do Porto
14 de Julho de 2014
Duração: 2 horas

Parte I

(1) A compilação dum termo da linguagem FUN da forma let $f = M$ in N para a máquina SECD resulta na seguinte sequência de código:

```
[LDF [LD 0,LD 0,LDC 0,AP,AP,RTN],LDF [LDC 3,LD 0,MUL,LDC 1,ADD,RTN],AP,HALT]
```

Re-construa os fragmentos M, N 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 têm apenas valores inteiros e funções; contudo, podemos representar valores boleanos como inteiros:

falso é representado por *zero*;

verdade é representado por *qualquer inteiro diferente de zero.*

Pretende-se neste exercício que implemente as operações binárias de conjunção "e_1 and e_2" e disjunção "e_1 or e_2" com avaliação não-estrita, isto é, que não avaliam e_2 quando o resultado da expressão pode ser determinado apenas por e_1. Alguns exemplos (onde '_' representa qualquer expressão):

$$0 \text{ and } \_ \Downarrow 0 \qquad 1 \text{ and } 1 \Downarrow 1$$
$$0 \text{ or } 1 \Downarrow 1 \qquad 1 \text{ or } \_ \Downarrow 1$$

(a) Mostre como implementar estes operadores na máquina SECD apresentada nas aulas; concretamente pretende-se que define as regras de compilação para eles. Sugestão: não é necessário acrescentar novas instruções à SECD — utilize as instruções SEL/JOIN para expressões condicionais.

(b) Considere a expressão

$$0 \text{ and } ((\lambda x.\, xx)\ (\lambda x.\, xx))$$

Use o esquema de compilação da SECD e as regras definidas na alínea anterior para justificar porque a redução desta expressão dá 0 apesar da sub-expressão $(\lambda x.\, xx)\ (\lambda x.\, xx)$ não admitir forma normal (i.e. a sua redução não termina).

(3) Considere a seguinte definição em Haskell da função *zipWith* que combina duas listas numa só usando uma função passada como primeiro argumento.

```
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith f [] ys = []
zipWith f xs [] = []
zipWith f (x:xs) (y:ys) = f x y : zipWith f xs ys
```

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) Considere o seguinte predicado Prolog:

```
p(no,[]).
p(yes,Z) :- do([],Z).
p([X|Y],Z) :- p(X,[]), do(Y,Z).
p(or(X,Y),Z) :- p(X,Z).
p(or(X,Y),Z) :- p(Y,Z).
p(X,Z) :- do(X,Z).
```

Assumindo que o código WAM de cada cláusula já se encontra compilado, escreva o código WAM de optimização que utiliza indexação no primeiro argumento. Na sua resolução referencie o código compilado de cada cláusula como *'Código WAM de C'* em que *C* é a cláusula pretendida. Por exemplo, o código da primeira cláusula deverá ser referenciado como *'Código WAM de p(no,[]).'*.

(5) Considere o seguinte programa Prolog em que *p/3* é um predicado tabelado com *mode-directed tabling*:

```
:- table p(index,index,min).
p(A,B,D) :- p(A,X,D1), e(X,B,D2), D is D1+D2.     e(a,b,1).
p(A,B,D) :- e(A,B,D).                              e(b,a,2).

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

Considere ainda a seguinte consulta:

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

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

(b) Descreva de forma sucinta o que aconteceria caso *p/3* fosse apenas declarado como um predicado tabelado sem *mode-directed tabling*.

(c) A técnica de tabulação pode ser implementada utilizando diferentes *estratégias de escalonamento*. A escolha da estratégia de escalonamento pode dar origem a diferentes ordenações da sequência de soluções e pode ter um impacto significativo no desempenho. As estratégias de escalonamento mais conhecidas são as estratégias de *batched scheduling* e de *local scheduling*. Descreva de forma sucinta em que consistem ambas as estratégias e enumere as suas principais diferenças, vantagens e desvantagens.

(6) Um dos principais desafios na exploração de paralelismo-Ou em Prolog é o problema dos *múltiplos ambientes*. Diga no que consiste esse problema, e explique quais são os dois principais modelos de execução de paralelismo-Ou que solucionam o problema dos múltiplos ambientes.