Segunda Prova de 2013.1 — Linguagems de Programação

Fabio Mascarenhas

05 de Agosto de 2013

A prova é individual e sem consulta. Responda as questões na folha de respostas, a lápis ou a caneta. Se tiver qualquer dúvida consulte o professor.

Nome: _____

DRE: _____

Questão:	1	2	3	Total
Pontos:	5	3	2	10
Nota:				

1. O trecho de Scala abaixo é de um interpretador para *microc*, escrito usando manipulação direta da memória ao invés da abstração de *ações*:

```
case Let(nome, exp, corpo) => (sp, mem) => {
  val (ve, sp1, mem1) = exp.eval(funs)(env)(sp, mem)
  val (vc, sp2, mem2) = corpo.eval(funs)(env + (nome -> sp))(sp + 1, mem + (sp -> ve))
  (vc, sp2 - 1, mem2 - (sp2 - 1))
}
```

   (a) (1 ponto) Dê a assinatura da função `eval` desse interpretador, considerando que os valores e endereços são de tipo `Int`.

   (b) (2 pontos) O trecho apresenta bugs de linearidade no uso da memória e do *stack pointer*; escreva um programa *microc* que mostra o problema, dizendo qual o resultado do programa no interpretador com os bugs e qual o resultado correto (assuma que o restante do interpretador está correto).

   (c) (2 pontos) Reescreva o trecho para eliminar os bugs de linearidade.

2. (3 pontos) O trecho de Scala abaixo é de um interpretador para *fun* com referências de primeira classe, escrito com manipulação direta da memória ao invés da abstração de *ações* e sem bugs de linearidade:

```
case Soma(e1, e2) => mem => {
  val (v1, mem1) = e1.eval(funs)(env)(mem)
  val (v2, mem2) = e2.eval(funs)(env)(mem1)
  (v1, v2) match {
    case (NumV(n1), NumV(n2)) => (NumV(n1 + n2), mem2)
    case _ => sys.error("soma precisa de dois números")
  }
}
```

Reescreva o trecho para usar *continuações* ao invés do estilo direto acima; a assinatura da função `eval` do interpretador de continuações é dado abaixo, onde `Valor => Mem => (Valor, Mem)` é o tipo das continuações:

```
def eval(funs: Env[Fun1])(env: Env[Valor]):
        (Valor => Mem => (Valor, Mem)) => (Mem => (Valor, Mem))
```

3. (2 pontos) Continuações não são úteis apenas na semântica de linguagens; o estilo de programação em que uma função, ao invés de retornar seus resultados, passa esses resultados para uma continuação é chamado de *estilo de passagem de continuações* (*continuation passing style*, ou *CPS*).

   Um grande exemplo de CPS são as funções assíncronas usadas em JavaScript, como no código abaixo, em que a função `post` da biblioteca *jQuery* recebe uma continuação para a qual passa o resultado da requisição:

```
$.post('ajax/test.html', function (data) {
  $('.result').html(data);
});
```

   Reescreva o código *fun* abaixo para que todas as funções (`dois`, `dobra`, `soma` e `func` estejam no estilo de passagem de continuações:

```
fun dois()                      -- fun dois(k) ... end
  2
end

fun dobra(x)                    -- fun dobra(x, k) ... end
  x * 2
end

fun soma(a, b)                  -- fun soma(a, b, k) ... end
  a + b
end

fun func()                      -- fun func(k) ... end
  let a = dois() in
    let b = dobra(a) in
      soma(a, b)
    end
  end
end
```

   Os comentários indicam qual a assinatura a função deve ter quando reescrita em CPS.

BOA SORTE!