Primeira Prova de 2013.1 — Linguagems de Programação

Fabio Mascarenhas

03 de Junho 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	4	Total
Pontos:	1	2	4	3	10
Nota:					

1. (1 ponto) Na semântica de ambientes, uma variável está livre se ela não existe no ambiente atual. Na semântica de substituição, o que determina que uma variável está livre?

2. O programa *fun* abaixo avalia para 14 em um interpretador com escopo léxico:

```
let x = 4 in
  let f = fun (y) x + y end in
    let x = 5 in
      (f)(10)
    end
  end
end
```

   (a) (1 ponto) Dê os passos da avaliação desse programa em um interpretador de substituição.

   (b) (1 ponto) Para qual valor ele avalia com escopo dinâmico? Por quê?

3. O tipo algébrico abaixo dá a sintaxe abstrata de uma linguagem que tem variáveis com escopo estático e variáveis com escopo dinâmico, onde variáveis introduzidas por `Let` e parâmetros têm escopo estático e variáveis introduzias por `LetDyn` têm escopo dinâmico. Os valores da linguagem são números inteiros e closures, dados por um tipo abstrato `Valor`.

```
trait Exp
case class Num(n: Int) extends Exp
case class Soma(e1: Exp, e2: Exp) extends Exp
case class If0(cond: Exp, ethen: Exp, eelse: Exp) extends Exp
case class Fun(param: String, corpo: Exp) extends Exp
case class Var(nome: String) extends Exp
case class Ap(fun: Exp, arg: Exp) extends Exp
case class Let(nome: String, exp: Exp, corpo: Exp) extends Exp
case class LetDyn(nome: String, exp: Exp, corpo: Exp) extends Exp
```

(a) (1 ponto) Dê a definição do tipo abstrato `Valor`.

(b) (1 ponto) Dê a assinatura da função *eval* para expressões.

(c) (1 ponto) Dê a implementação do caso `Ap` de eval (você é livre para usar funções auxiliares, mas deve fornecer a implementação dessas funções também).

(d) (1 ponto) Dê a implementação do caso `Var` de eval, dado que uma varíavel primeiro deve ser verificada se é uma variável estática, depois dinâmica.

4. Uma linguagem com funções de primeira classe não precisa de funções com múltiplos parâmetros como uma primitiva da linguagem; elas podem ser açúcar sintático para funções de um parâmetro:

```
fun (a, b, c)          fun (a)
  ...              =>    fun (b)
end                        fun (c)
                             ...
                           end
                         end
                       end

(foo)(a, b, c)     => (((foo)(a))(b))(c)
```

Funções sem parâmetros podem ser açúcar para uma função com um parâmetro $, onde $ é um nome que não pode aparecer como nome de variável.

As definições abaixo dão o fragmento para o tipo algébrico que dá a sintaxe abstrata das funções e chamadas com um e com múltiplos parâmetros.

```
case class Fun(param: String, corpo: Exp) extends Exp
case class Ap(fun: Exp, arg: Exp) extends Exp
case class FunM(params: List[String], corpo: Exp) extends Exp
case class ApM(fun: Exp, args: List[Exp]) extends Exp
```

(a) (1½ pontos) Dê o caso da função `desugar` para `FunM`, que converte a definição de função de múltiplos para um parâmetro.

(b) (1½ pontos) Dê o caso da função `desugar` para `ApM`, que converte a chamada de função de múltiplos para um parâmetro.

BOA SORTE!