Primeira Prova de 2015.2 — Linguagens de Programação

Fabio Mascarenhas

16 de Dezembro de 2015

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:	2	6	2	10
Nota:				

1. (2 pontos) Dê os passos de avaliação *small-step* para o programa *fun* abaixo:

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

2. As declarações abaixo estendem o tipo algébrico que representa expressões de *fun* para acrescentar operações booleanas. Os operadores `And` e `Or` têm curto-circuito (no caso do `And`, se o `e1` for falsa o resultado todo é falso sem avaliar `e2`, e no caso de `Or` se `e1` for verdadeiro todo o resultado é verdadeiro sem avaliar `e2`). Fazer uma operação lógica com um operando que não é booleano é uma operação indefinida.

```
case class And(e1: Exp, e2: Exp) extends Exp
case class Or(e1: Exp, e2: Exp) extends Exp
case class Not(e: Exp) extends Exp
```

   (a) (3 pontos) Dê os casos da função `eval` de avaliação *big-step* para os novos operadores. Lembre-se que o valor booleano verdadeiro é `TrueV()` e o falso é `FalseV()`.

   (b) (3 pontos) Dê os casos da função `step` de avaliação *small-step* para os novos operadores. Lembre-se que o termo para o booleano verdadeiro é `True()` e o falso é `False()`.

3. (2 pontos) 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

fun ()                 fun ($)
  ...              =>    ...
end                    end
```

Funções sem parâmetros podem ser açúcar sintático para uma função com um parâmetro chamado $, já que $ não é válido como nome de um identificador. A nova definição para funções anônimas no tipo algébrico de expressões de *fun* é essa:

```
case class Fun(param: String, corpo: Exp) extends Exp
```

O código abaixo é o fragmento do parser de *fun* que reconhece uma função anônima com zero ou mais parâmetros:

```
for {
  _ <- kw("fun")
  _ <- op("(")
  params <- (for {
               (p, _) <- id
               ps <- many(for {
                            _ <- op(",")
                            (p, _) <- id
                          } yield p)
             } yield p :: ps) +: empty(List())
  _ <- op(")")
  corpo <- ExpFun
  _ <- kw("end")
} yield Fun(params, corpo) // params é List[String], corpo é Exp
```

Escreva a expressão que deve ir no lugar do `Fun(params, corpo)` para implementar o açúcar sintático descrito acima.

BOA SORTE!