MAC 0316/5754 — Conceitos de Linguagens de Programação

Primeiro Semestre de 2010

Segunda Prova — 11 de maio de 2010

Versão revisada, com pequenas correções na questão 1(a) e na questão 7

Nome do aluno: _____

Assinatura: _____

Nº USP: _____

Instruções:

1. Preencha o cabeçalho acima.
2. Não destaque as folhas deste caderno.
3. A prova tem 8 questões. Antes de começar a trabalhar verifique se o seu caderno de questões está completo.
4. A prova deve ser resolvida individualmente. Não é permitida a consulta a livros, apontamentos ou colegas.
5. Não é permitido o uso de folhas avulsas para rascunho.

Boa prova!

Questão	Valor	Nota
1	1,5	
2	1,5	
3	1,0	
4	2,0	
5	0,5	
6	1,5	
7	1,0	
8	1,0	
Total	10,0	

Questão 1 (0,5 + 0,5 + 0,5 pontos)

Vimos em classe a linguagem FWAE, cuja sintaxe (concreta) é definida da seguinte maneira, em BNF:

```
<FWAE> ::= <num>
         | {+ <FWAE> <FWAE>}
         | {with {<id> <FWAE>} FWAE}
         | <id>
         | {fun {<id>} FWAE}
         | {FWAE FWAE}
```

Em FWAE os identificadores têm escopo estático. Agora considere a linguagem "FWAE com escopo dinâmico", ou FWAE/D. Essa é uma linguagem com sintaxe idêntica à de FWAE e semântica muito parecida com a de FWAE. A única diferença é que em FWAE/D o escopo dos identificadores é dinâmico.

(a) Dê um exemplo de programa que rode sem problemas tanto sob um interpretador FWAE quanto sob um interpretador FWAE/D, mas produza um resultado numérico diferente em cada uma dessas linguagens.

(b) Dê um exemplo de programa que gere um erro de "identificador livre" quando executado por um interpretador FWAE, mas rode sem problemas e produza um resultado numérico quando executado por um interpretador FWAE/D.

(c) Dê um exemplo de programa que rode sem problemas e produza um resultado numérico quando executado por um interpretador FWAE, mas gere um erro semântico quando executado por um interpretador FWAE/D.

Questão 2 (0,5 + 0,5 + 0,5 pontos)

Vimos que a forma `with` da linguagem FWAE é "acúcar sintático" para uma aplicação de função. O mesmo acontece com a forma `let` de Scheme.

(a) Reescreva o seguinte programa FWAE, substituindo a forma `with` pela correspondente aplicação de função.

```
{with {duplica {fun {x} {+ x x}}}
  {+ 1 {duplica 10}}}
```

(b) Reescreva o seguinte programa Scheme, substituindo a forma `let` pela correspondente aplicação de função.

```
(let ([duplica (lambda (x) (+ x x))]
      [soma-um (lambda (x) (+ 1 x))])
    (soma-um (duplica 10)))
```

(c) Reescreva o seguinte programa FWAE, substituindo as formas `with` pelas correspondentes aplicações de funções.

```
{with {duplica {fun {x} {+ x x}}}
  {with {soma-um {fun {x} {+ 1 x}}}
    {soma-um {duplica 10}}}}
```

Questão 3 (1,0 pontos)

A definição e a implementação de uma linguagem comportam escolhas em múltiplas dimensões de projeto. Vimos pelo menos duas dessas dimensões:

- Dimensão 1 (regime de substituição): substituição explícita ou substituição postergada?
- Dimensão 2 (regime de avaliação): avaliação ávida ou avaliação preguiçosa (postergada)?

Explique o que varia em cada uma das duas dimensões de projeto acima. Deixe clara a diferença entre substituição postergada e avaliação preguiçosa.

Questão 4 (2,0 pontos)

(a) Explique o que é fechamento (*closure*).

(b) Qual o propósito dos fechamentos? Em outras palavras, que problema ocorreria se eles não fossem usados?

(c) Há relação entre a necessidade de fechamentos o regime de substituição (explícita ou postergada)?

(d) As estruturas `fun` e `closureV` são usadas por vários dos interpretadores do PLAI. Compare essas duas estruturas. Qual a finalidade de cada uma? Elas têm campos em comum? Quais? Que campo(s) as distingue(m)?

Questão 5 (0,5 pontos)

Dê um exemplo simples de programa Scheme que gere um erro em tempo de execução, mas cuja tradução para Haskell rode sem problemas graças à avaliação preguiçosa.

Questão 6 (1,5 pontos)

Considere a linguagem CFAE/L (*Conditionals, Functions and Arithmetic Expressions*, com semântica *Lazy*), cuja sintaxe é definida da seguinte maneira, em BNF:

```
<CFAE/L> ::= <num>
           | {+ <CFAE/L> <CFAE/L>}
           | <id>
           | {fun {<id>} CFAE/L}
           | {CFAE/L CFAE/L}
           | {if0 CFAE/L CFAE/L CFAE/L}
```

(a) Em que ocasiões um interpretador para a linguagem CFAE/L cria fechamentos de expressões (*expression closures*)?

(b) Qual o propósito dos fechamentos de expressões? Em outras palavras, que problema ocorreria se o interpretador não criasse esses fechamentos?

(c) Quais os pontos de avaliação estrita (*strictness points*) da linguagem CFAE/L? Em outras palavras, em que ocasiões um interpretador precisa forçar a conversão de fechamentos de expressões em outros tipos de valores (valores numéricos ou fechamentos "normais")?

Questão 7 (1,0 pontos)

Simule a execução do seguinte programa RCFAE:

```
{rec {fac {fun {n}
             {if0 n
                  1
                  {* n {fac {+ n -1}}}}}}
  {fac 2}}
```

Mostre o ambiente (*environment*) a cada passo (isto é, o ambiente passado como parâmetro em cada chamada recursiva ao interpretador), bem como o ambiente contido em cada fechamento criado pelo interpretador.

Questão 8 (1,0 pontos)

O material do PLAI que vimos até agora usou mutabilidade (*boxes* de Scheme) em apenas duas ocasiões. Quais foram essas ocasiões? Explique para que finalidade foi usada uma *box* em cada uma dessas ocasiões.