Interpretação e Compilação de Linguagens de Programação

Exame de Época Normal 2009-2010

6 de Janeiro de 2010

Notas: O exame pode ser realizado com consulta de materiais em suporte papel trazidos pelo aluno. O exame tem a duração de 3h.

1. Considere a linguagem *VEC* cuja sintaxe concreta é definida pela gramática a seguir.

$$\begin{aligned} E ::= & \text{Num} \mid E+E \mid \text{Id} \mid \text{decl } \text{Id} = E \text{ in } E \\ & \mid \text{fun}(\text{Id}) \{ E \} \mid E(E) \mid \text{if } E \text{ then } E \text{ else } E \\ & \mid \text{array}(E) \mid E[E] \mid E := E \mid !E \\ & \mid E \oplus E \mid \text{ifend } E \text{ then } E \text{ else } E \end{aligned}$$

As construções básicas da linguagem *VEC* são os números inteiros (*Num*) e as operações habituais sobre os números inteiros (aqui simbolizadas pela operação de adição), os identificadores (*Id*), e a declaração recursiva de identificadores ($\text{decl } x = E_1 \text{ in } E_2$). A definição de funções, a chamada de funções e a expressão condicional. A expressão condicional está definida para condições sobre os números inteiros; tem o comportamento esperado com o valor 0 (zero) a ser interpretado como falso e qualquer outro valor inteiro como verdadeiro.

A linguagem *VEC* é imperativa, no entanto a memória é sempre alocada na forma de *arrays* produzidos pela expressão $\text{array}(E)$ onde o tamanho do *array* é dado pela expressão *E*. Para manipular o estado existe também a expressão de indexação ($E[E]$) que denota uma referência para uma célula de memória pertencente a um *array*, a afectação ($E_1 := E_2$) cujo valor é o valor denotado pela expressão E_2 , e a desreferenciação ($!E$). O valor inicial de uma célula de memória (de um *array*) é sempre 0.

Existem ainda duas expressões especiais sobre referências. A primeira é a expressão de adição sobre apontadores ($E_1 \oplus E_2$) que denota a referência para a célula de memória que se encontra *n* posições à frente da célula denotada pela (referência da) expressão E_1 no *array* respectivo, sendo *n* o valor denotado pela expressão pela expressão E_2 . Note-se que a indexação ou uma operação de adição que exceda o tamanho do *array* produz uma referência inválida. A desreferenciação de uma referência inválida causa um erro de execução. A segunda expressão sobre referências é a expressão $\text{ifend } E_1 \text{ then } E_2 \text{ else } E_3$ que funciona como uma expressão condicional onde é escolhido o ramo E_2 se a referência denotada pela expressão E_1 for inválida e o ramo E_3 no caso contrário.

A linguagem *VEC* tem passagem de parâmetros por valor (*call-by-value*) e resolução estática de nomes.

Considere o seguinte exemplo da linguagem *VEC*.

```
decl a = array(10) in
decl init = fun(x){ ifend x then 0 else x := init(x ⊕ 1) + 1 } in
decl y = init(a[0]) in
decl f = fun(x){ ifend x then 0 else !x + f(x ⊕ 1) } in
f(a[0])
```

- (a) **Descreva** a sintaxe abstracta da linguagem *VEC* usando um conjunto de classes. Apresente apenas a assinatura de um constructor de cada classe com o nome da classe e um parâmetro para cada variável de instância.
- (b) **Enumere** os valores da linguagem *VEC* e represente-os numa hierarquia de classes.
- (c) **Defina** a semântica das seguintes construções da linguagem *VEC*:

$$\text{array}(E) \quad E[E] \quad E := E \quad !E \quad E \oplus E \quad \text{ifend } E \text{ then } E \text{ else } E$$

através da implementação do método **evaluate** de cada uma das classes correspondentes.

Explique sucintamente quais os parâmetros e quais os possíveis valores de retorno do método.

- (d) **Enumere** quais os erros de execução que podem ocorrer num programa da linguagem *VEC*.

2. Considere o programa escrito na linguagem MicroML apresentada nas aulas da disciplina.

```

decl x = 1 in
decl f = fun y → x + 1 in
decl g = fun h → fun z → f(h(z)) in
decl f = fun y → f(y) in
g(g)(f)

```

- Indique justificando** qual o ambiente da primeira avaliação da expressão $f(h(z))$ no corpo da função g se se considerar resolução estática de nomes.
 - Indique justificando** qual o ambiente da primeira avaliação da expressão $f(h(z))$ no corpo da função g se se considerar resolução dinâmica de nomes.
 - Indique justificando** o resultado do programa (se existir) quando interpretado com passagem de parâmetros por necessidade (*lazy*) e resolução dinâmica de nomes.
 - Indique justificando** o resultado do programa (se existir) quando interpretado com passagem de parâmetros por necessidade (*lazy*) e resolução estática de nomes.
 - Indique** o resultado do programa (se existir) quando interpretado com passagem de parâmetros por valor (*call-by-value*) e resolução estática de nomes.
 - Considere o sistema de tipos para a linguagem MicroML apresentado nas aulas da disciplina. **Indique** se o exemplo acima pode ser tipificado e nesse caso **proponha** uma versão com anotações de tipos nas declarações e nos parâmetros das funções.
3. Considere a linguagem VET, uma extensão tipificada da linguagem VEC com anotações de tipos na declaração dos parâmetros das funções, nas declarações recursivas e na criação de arrays.

$$\begin{aligned}
 E ::= & \dots \mid \text{decl } Id : T = E \text{ in } E \mid \dots \\
 & \dots \mid \text{fun } (Id : T) \{ E \} \mid \dots \\
 & \dots \mid \text{array}(T)(E) \mid \dots
 \end{aligned}$$

- Indique** quais os tipos necessários para tipificar a linguagem VET.
- Defina** o sistema de tipos da linguagem VET através de um conjunto de regras de tipos. **Nota:** indique apenas os casos das construções enumeradas na alínea 1(c).
- Considere o seguinte exemplo na linguagem VEC

```

decl a = array(10) in
decl x = a[0] in
decl f = fun (z) { z ⊕ 5 } in
f(x) := 2

```

Apresente a versão tipificada do exemplo e a respectiva **derivação**.

4. **Apresente** um exemplo da utilização de *closures* na linguagem Java. **Enumere e justifique** quais as restrições colocadas à utilização de variáveis não locais nestas situações. **Explique** sucintamente como são traduzidas as *closures* na linguagem Java para a linguagem intermédia da máquina virtual.