Exame de Linguagens de Programação II

13 Fevereiro 2006

(duração: 2h)

I [9]

O objectivo deste exercício é desenvolver um interpretador para a linguagem C_0 (não confundir com C_++), definida pela seguinte gramática

```
Expr ::= Expr + Expr | Expr - Expr |  
        Expr * Expr | Expr / Expr |  
        { Id = Expr ; ... ; Id = Expr ; Expr ; ... ; Expr } |  
        Expr := Expr |  
        *Expr |  
        malloc(Expr) |  
        Num | Id | ( Expr )
```

Num e Id representam respectivamente os literais numéricos (ex: 0, 98) e os identificadores (ex: i, xpto, chicharro). Os programas nesta linguagem são expressões geradas pela categoria sintáctica Expr. Os únicos valores manipulados pela linguagem são os números **inteiros** e as **referências**, produzidos pela avaliação das expressões. A resolução de nomes na linguagem C_0 usa a regra correcta de resolução estática.

A avaliação da expressão `malloc(Expr)` produz como resultado uma referência para o primeiro elemento de um novo vector, cujo número de elementos é indicado pelo valor da expressão inteira *Expr*. Por exemplo `malloc(2 + 3)` devolve uma referência para o primeiro elemento de um novo vector (alocado em memória fresca) com exactamente 5 elementos. Cada elemento é uma célula de memória com capacidade para um número inteiro, inicializada com 0. Dada uma expressão *E* que produz uma referência, a expressão `*E` devolve o conteúdo (um inteiro) da célula de memória referida.

Em C_0 uma referência pode indicar não apenas a base de um vector, mas também qualquer um dos seus outros elementos. Por exemplo: se *R* for uma expressão que produz uma referência e *N* for uma expressão que produz um inteiro, então `R + N` produz uma referência para o elemento que está *N* posições à frente de *R*. Por exemplo: a expressão `*(j + 2)` devolve o conteúdo da terceira posição do vector referido por *j*.

A afectação $E_1 := E_2$ tem o efeito esperado: substituir o valor inteiro guardado na célula de memória referida pela referência E_1 pelo valor inteiro produzido pela expressão E_2 . Note-se que a expressão E_1 deve sempre produzir uma referência e a expressão E_2 um inteiro.

Eis um exemplo mais completo de um programa em C_0 :

```
{  
  k = 2;  
  a = malloc(k);  
  b = malloc(*a+1);  
  a := 1;  
  a+1 := 2;  
  b := *a - *(a+1);  
  *b  
}
```

Note que as 3 primeiras linhas são declarações e as últimas 4 linhas são expressões (ver gramática acima). O valor deste programa é -1.

- [1] Explique como representar a sintaxe abstracta da linguagem C_0 usando um conjunto de classes na linguagem Java.
- [1] Explique como representar os valores (resultados) da linguagem C_0 em Java. (Sugestão: as referências C_0 não podem ser apenas simples referências Java, mas sim pares referência-inteiro).

3. [4] Especifique a semântica operacional da linguagem C° , programando em cada uma das classes que apresentou em (1) acima, os métodos de avaliação

```
? evaluate(a: Env)
```

de tal modo que, dada uma AST guardada na variável `exp`, a chamada

```
exp.evaluate(new Env())
```

o execute completamente, ou assinale um erro de interpretação.

Importante: `Env` implementa a interface “standard” usada nas aulas.

4. [1] Na linguagem C , se a referir um vector de inteiros, podemos guardar 2 na sua primeira posição usando a afectação $*a = 2$. Em C° , escrevemos $a := 2$ para obter o mesmo efeito. Qual lhe parece a forma mais simples de interpretar a operação de afectação (relacione com a noção de valor esquerdo e valor direito)?
5. [1] Que erros podem ocorrer durante a execução de um programa C° sintacticamente correcto?

II [8]

Este exercício trata de sistemas de tipos. Imagine que se pretende desenvolver um sistema de tipos para a linguagem C° , com o objectivo de garantir estaticamente a ausência de alguns erros de execução. A linguagem resultante é a poderosa $C^\circ\circ!$ A ideia é considerar os tipos

```
Type ::= int | pointer
```

onde *int* é o tipo dos inteiros, *pointer* é um tipo atribuído a expressões que produzem referências. A sintaxe da linguagem C° é exactamente a mesma da linguagem $C^\circ\circ$.

1. [4] Descreva um sistema de tipos que atribua um tipo razoável a cada expressão da linguagem C° . Preferivelmente, apresente o sistema de tipos usando um conjunto de regras. Por exemplo, o seu sistema poderá derivar a asserção de tipificação

$$x : \text{pointer}, n : \text{int} \vdash *(x + n + 3) : \text{int}$$

indicando que em qualquer contexto onde x tem tipo *pointer* e n tem tipo *int*, a expressão $*(x + n + 3)$ terá tipo *int*.

2. [1] Compare o tipo *int* da linguagem C° com o tipo *int* da linguagem C . Qual é a diferença?
3. [2] Usando o sistema de tipos anterior, indique qual o tipo esperado das expressões, apresentando as respectivas derivações:

```
(a) { a = 2; b = malloc(a); b+a }
```

```
(b) { b = malloc(1); c = malloc(1); *b + *c }
```

4. [1] Dos erros salientados na questão I.5, quais são impedidos pelo seu sistema de tipos e quais não é possível garantir estaticamente? Justifique.

III [4]

1. [2] Considere o programa CORE

```
decl
  x = 1
  y = decl z = x*2 in 3-z end + 4 end
in print(x+y) end
```

Apresente uma sequência de instruções CIL para este programa, usando variáveis locais de tipo `int32`.

2. [2] Explique em linhas gerais como poderia ser compilada a linguagem C° para CIL. Ilustre, apresentando o código gerado para o programa na alínea II.3 (b).