Conceitos de Linguagens de Programação
Primeira Prova – 2008

Sugestão de Respostas

1. Explique a diferença entre laziness e substituição postergada.

R. Substituição postergada é uma forma de evitar que o parse tenha que avaliar todo o código do programa sempre que encontra um *with* (principalmente no caso de with's alinhados, como o que tem no início de capítulo 5 do livro). Para tanto, a substituição postergada usa um repositório de substituições, onde o identificador fica associado a um valor. Laziness, assim como eagerness, é só uma das técnicas de como proceder com essa associação. Com laziness o identificador fica associado à named expression, e essa última só é calculada quando necessário. Com eagerness a named expression é calculada antes de ser associada ao identificador.

2. Mostre um exemplo em que acontece erro na interpretação ávida (eager), mas não na preguiçosa (lazy).

R. {with {x {+ 4 5}}
{with {y {+ x x}}
{with {z y}
{with {x 4} z}}}

Com ávida o resultado é 18 e com preguiçosa é 8.

3. Compare o repositório de funções no F1WAE com o ambiente Environment da substituição postergada. Quais as similaridades? Quais as diferenças?

R. Similar ao environment o repositório de funções também associa identificadores a valores, só que neste caso os valores são funções. Porém para que a substituição possa ser postergada corretamente (i.e. sem escopo dinâmico) é necessário que as substituições no corpo da função sejam carregadas consigo. Esse é o conceito de closure.

4. Descreva pelo menos uma situação onde o escopo dinâmico pode ser interessante. Dica: Uma das formas está ligada à utilização do shell do Unix.

R. Um exemplo seria uma função que utiliza o valor de uma variável global. Sendo assim, a função teria um comportamento diferente a depender do valor da variável global no momento em que a função é utilizada. Um exemplo é função 'cd' que, se usada sem parâmetro algum, busca o valor da variável global $HOME. Se o valor dessa variável for um caminho inválido a função retorna uma mensagem de erro.

5. Considere a expressão abaixo:

```
1) (interp
2)   (parse `{with {a {+  3  5}}
3)                  {with {b {-  a  3}}
4)                        {+ {with {c {+  b  a}} {*   c  4}}
5)                          {with {a {+  b  a}} {-  a  23}}
6)                 }}}}))
```

Mostre a avaliação e os valores do environment (repositório) a cada passo, com e sem laziness.

:: Com laziness:
2) [a → {+ 3 5}]
3) [b → {- {+ 3 5} 3}; a → {+ 3 5};]
4) [c → {+ {- {+ 3 5} 3} {+ 3 5}}; b → {- {+ 3 5} 3}; a → {+ 3 5}]
5) [a → {+ {- {+ 3 5} 3} {+ 3 5}}; c → {+ {- {+ 3 5} 3} {+ 3 5}};
b → {- {+ 3 5} 3}; a → {+ 3 5}]

:: Sem laziness:
2 [a → 8]
3 [b → 5; a → 8]
4 [c → 13; b → 5; a → 8]
5 [a → 13; c → 13; b → 5; a → 8]

:: Avaliação (em 4 a = 8, mas em 5, a definição de a muda e fica igual a 13) :
{+ 52 (-10)} = 42

6. Responda: É verdade que o repositório sempre cresce à medida em que nos aproximamos do final da expressão? O que determina o tamanho do repositório?

R. Mentira. O tamanho do repositório depende da quantidade de escopos aninhados, isto é, o repositório aumenta à medida em que passamos de um escopo mais externo para um mais interno. Quando saímos de um escopo, os identificadores que só faziam parte dele pode ser retirados do repositório.