Lenguajes de Programación (CC41A) - Primavera 2009

Clase Auxiliar 6

Profesor: Tomás Barros
Auxiliar: Víctor Ramiro

1. Considere un lenguaje con la siguiente gramática:

```
<expr> ::= <id> | <num>
         | (+ <expr> <expr>)
         | (if <expr> <expr> <expr>)
         | (lambda (<id>) <body>)
         | (<expr> <expr>)
```

   Defina la función `free-vars` que retorna la lista de variables libres de una expresión dada.

2. Considere la función `fold` que reduce una lista de elementos basado en: una función de reducción `f`, un valor inicial, una lista de elementos de cualquier tipo. Por ejemplo:

```
(fold + 0 '(1 2 3)) ---> 6
(fold + 2 '(1 2 3)) ---> 8
```

   a) ¿Cual tiene que ser la firma de la función de reducción? De un ejemplo de uso con una función `f` *anónima* definida por el programador.

   b) Sabiendo que el primer parametro de la función de reducción tiene que ser el elemento tomado de la lista a reducir, defina `fold` en Scheme.

   c) Use `fold` para definir `reverse`, la función que toma una lista y retorna la lista inversa:

```
(reverse '(1 2 3)) ---> (3 2 1)
```

3. **Meta o no meta?** En un meta-intérprete del FAE, las funciones del lenguaje base estan representadas como funciones del lenguaje usado para definir el interprete.

   a) Si el intérprete esta definido en Scheme, ¿que puede decir del *scope* de las funciones del FAE? ¿por qué?

   b) Si el interprete esta definido en Scheme, ¿que puede decir del *régimen de evaluación* del lenguaje interpretado? ¿por qué?

   c) Si el interprete esta definido en Haskell, ¿que puede decir del *scope* de las funciones del FAE? ¿por qué?

   d) Si el interprete esta definido en Haskell, ¿que puede decir del *régimen de evaluación* del lenguaje interpretado? ¿por qué?

4. *a*) Considere el siguiente programa Scheme:

```
(define factor 3.14)
(define (f x) (* factor x))
```

```
(define (g x)
  (let ((factor 1.96))
    (f x)))
```

¿Es cierto que las funciones `f` y `g` siempre retornan el mismo resultado? ¿o siempre un resultado distinto? ¿por qué?

b) En Common Lisp, una variable global introducida por `defvar` siempre tiene scope dinámico, y se le llama *variable especial*. Por convención, el nombre de una variable especial se prefija y postfija con `*`. Por ejemplo:

```
(defvar *factor* 3.14)
(defun (f x) (* *factor* x))
(defun (g x)
  (let ((*factor* 1.96))
    (f x)))
```

¿Qué puede decir de las funciones `f` y `g` en este programa Common Lisp? ¿por qué?

5. Benancio acaba de implementar en Scheme la función `(lista-pares n)`, que retorna la lista de los primeros `n` números pares. El código es el siguiente:

```
(define lista-pares
  (let ((lista '()))
    (letrec ((pares (lambda (n)
                      (if (eq? n 0) lista
                        (begin
                          (set! lista (cons (* n 2) lista))
                          (pares (- n 1)))))))
     pares)))
```

a) ¿Es correcta esa implementación? Ilustre.

b) ¿Es necesario hacer uso de mutación para escribir `lista-pares`?