Auxiliar Cinco - CC41A
Lenguajes de Programación
Substitución y Funciones (2da Parte)

Oscar E. A. Callaú
oalvarez@dcc.uchile.cl

Santiago - Chile, 6/Sep/2008

1. Subtitución explícita (cont.)

Benancio, un alumno de cc41a algo despistado, le hace la siguiente pregunta: *Como es posible tener expresiones del tipo (A) y no del tipo (B) en nuestros interpretes?*

```
;;(A) .. suponiendo 0 = false
>(interp (parse '{sum 3})
         (list (fundef 'sum 'n
                       (parse '{if n {+ n {sum {- n 1}}} n}))))

;;(B)
>(interp (parse '{with {sum {fun {n} {if n {+ n {sum {- n 1}}} n}}}
                       {sum 3}}))
```

Que le respondería a Benancio de tal manera de que logre comprenderlo? Recuerde no le estoy pidiendo solucione el problema, solo que responda porque no es posible *(B)* pero si *(A)*.

Pauta, En el caso *(A)* el lenguaje usado presenta funciones de 1er orden, por tanto este lenguaje de por si acepta recursión, ya que las funciones siempre estan visibles en todo momento para sus sustitución con la lista de fundefs. En el caso *(B)*, un ejemplo de este interprete es el `FAE`, con las funciones de 1er orden y con la implementación de subsitución que hemos visto (la explícita como la diferidad) no es posible obtener recursión, ya que no puedo mantener todavia una referencia cíclica.

2. Funciones de 1era Clase

Encuentre las diferencias en estos dos pedazos de código:

```
{with {a 5}
 {+ a a}}
;;--------
{{fun {a} {+ a a}} 5}
```

Dadas sus diferencias como puede optimizar su interprete para que maneje estos dos casos sin redundancia?

Pauta, las diferencias solo se presentan en el nivel sintáctico y no así en el semántico, por tanto puedo optimizar la definición de mi lenguaje cambiando toda referencia del `with` por la de una aplicación de función con una λ, esto se lo realiza en el `parser` como un ázuzar sintáctico.

```
(define (parser sexp)
 ..
 [(with) (app (fun (first (second sexp)) (parse (third sexp)))
              (parse (second (second sexp))))]
 ..
)
```

3. Funciones varios

Hasta el momento hemos vistos estos tres clases de funciones que puedes existir en un lenguaje:

1. Funciones de 1er orden
2. Funciones de orden superior
3. Funciones de 1era clase

tiene por trabajo entonces agrupar las siguientes definiciones o características según donde corresponda:

2 `filter :: (a ->Bool) x list-of(a) ->list-of(a)`

3
```
{with {f {fun {x} x}}
         {f f}}
```

1 Un programa escrito en Java

1 Hasta el momento es nuestro único mecanismo de recursión.

3 Es la base del λ-cálculo

2 Se caracteriza por recibir uno o mas funciones como parámetros o generar una función con su salida.