Examen – Lenguajes de Programación (CC41A)

Departamento de Ciencias de la Computación – Universidad de Chile

Profesor: Éric Tanter

14 de Julio 2008

duración: 2 horas / 1.5pt por pregunta)

sin apuntes

1. *a*) Defina en Scheme la función `compose` que toma como parámetro dos funciones unarias f y g y retorna una función que corresponde a la composición $f \circ g$. Eg:

   ```
   ((compose square add1) 7) --> 64
   ```

   b) Usando `compose`, defina la función `repeated` que toma una función f y un número $n > 0$, y retorna una función que corresponde a $f \circ f \circ \ldots \circ f$ (f aplicada n veces). Eg:

   ```
   ((repeated add1 10) 5) --> 15
   ```

2. *a*) Los lenguajes con evaluación perezosa no incluyen expresiones de mutación (como asignaciones). Explique porque es así, usando un ejemplo pequeño que ilustre su argumento.

   b) Un compañero le pide a usted que implemente la función `if0` en Scheme que recibe como tres parámetros: `e1`, `e2`, `e3`; tal que evalua `e2` si `e1` es 0, `e3` sino. La restricción que le imponen es que sea usada de la siguiente manera.

   ```
   (if0 (- 4 5)
        (write "zero")
        (write "not zero"))
   -> "not zero"
   ```

   ¿Qué le responde usted a su compañero? ¿Cambia su respuesta si le pide que sea implementada en Haskell?

3. Estamos usando un lenguaje que usa un GC tipo copy-collector, usando dos espacios y sin generaciones. Al ejectuar el programa, detectamos tiempos de detención durante el GC que son demasiado grandes para la aplicación. Un ingeniero propone agrandar el espacio de memoria dinámica con que se ejecuta el programa pero otro opina que eso hará que el GC demore más aún. Discuta esa afirmación.

   Si el lenguaje provee una opción de cambiar el GC por un Reference Count, sería una buena idea usarla?

4. Considere el siguiente lenguaje (similar al FAE visto en clases):

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

   - Exprese, en este lenguaje, un programa que nunca termina.
   - Especifique los juicios de tipos para las distintas expresiones de este lenguaje. No es necesario asumir que los programas llevan anotaciones de tipo (ie. puede suponer que tiene un inferenciador de tipos).
   - ¿Es posible asignar un tipo a su programa que nunca termina? En caso en que no, esto significa que tiene un lenguaje (FAE+tipos) que asegura que todos los programas validos en tipos siempre terminan. ¿No es eso contradictorio con el hecho de que los sistemas de tipos son sujetos al Halting Problem? Comente.