Pauta Examen – Lenguajes de Programación (CC41A)

Departamento de Ciencias de la Computación – Universidad de Chile

Profesor: Éric Tanter

26 de Noviembre 2008

duración: 2h30 / 1.5pt por pregunta

sin apuntes

1. Definiciones de funciones, en Scheme o Haskell.

   a) (0,4 pts/1,5) Defina la función `filter-in-all`, que toma como parametro un filtro y una lista, y elimina de la lista *todos* los elementos que no "pasan" el filtro, incluyendo los contenidos en sublistas.:

   ```
   ;; filter-in-all :: (a -> bool) * list-of(a) -> list-of(a)
   (define (filter-in-all pred? ls)
     (cond
      ((empty? ls) empty)
      ((list? (car ls)) (cons (filter-in-all pred? (car ls)) (filter-in-all pred? (cdr ls))))
      ((pred? (car ls)) (cons (car ls) (filter-in-all pred? (cdr ls))))
      (else (filter-in-all pred? (cdr ls)))))

   (filter-in-all odd? '((4 5) 2 (3 5 (8 7)))) --> ((5) (3 5 (7)))
   ```

   b) (0,4 pts/1,5) Defina la función `add-all`, que toma como parametro una lista y retorna la suma de todos los elementos, incluyendo sublistas. Por ej:

   ```
   ;; add-all:: list-of(number) -> number
   (define (add-all ls)
     (cond
      ((empty? ls) 0)
      ((number? (car ls)) (+ (car ls) (add-all (cdr ls))))
      ((list? (car ls)) (+ (add-all (car ls)) (add-all (cdr ls))))
      (else (add-all (cdr ls)))))

   (add-all '((4 5) 2 (3 5 (8 7)))) --> 34
   ```

   c) (0,7 pts/1,5) Ahora, defina una función `deep-recur` que capture el patrón común a

`filter-in-all` y `add-all`: la recursión profunda sobre listas. Luego redefina ambas funciones usando `deep-recur` (0,1 pts cada redefinición).

```
;; deep-recur:: (a * b -> b) * b * list-of(a) -> b
(define (deep-recur f ini ls)
  (cond
   ((empty? ls) ini)
   ((list? (car ls)) (f (deep-recur f ini (car ls)) (deep-recur f ini (cdr ls))))
   (else (f (car ls) (deep-recur f ini (cdr ls))))))

;; my-add-all:: list-of(number) -> number
(define (my-add-all ls)
  (deep-recur + 0 ls))

;; my-filter-in-all :: (a -> bool) * list-of(a) -> list-of(a)
(define (my-filter-in-all pred? ls)
  (deep-recur (lambda (x xs) (cond
                               ((list? x) (cons x xs))
                               ((pred? x) (cons x xs))
                               (else xs)))
              empty
              ls))
```

Para mayor detalle ver el archivo `examen.scm`

2. Sobre evaluación perezosa, explique:

   a) (2 pts/6) ¿Por qué, cuando es introducido en forma ingenua, la evaluación perezosa lleva a scope dinámico?

   R. Si decidimos implementar laziness simplemente guardando la expresión (sin su ambiente actual), degenerariamos en un scope dinámico ya que esta expresión puede contener variables que luego de guardar la expresión lazy cambien de valor.

   b) (2 pts/6) ¿Como se preserva el scope estático en un lenguaje con evaluación perezosa?

   R. Para mantener el scope estático es necesario guardar el ambiente actual cuando se crea la expresión, esto se puede ver claramente en datatype `CFAE/L-Value` en su constructor `(exprV (expr CFAE/L?) (env Env?))` donde `(env Env?)` es el ambiente actual referenciado a la expresión `expr`.

   c) (2 pts/6) ¿Cual es el costo de la evaluación perezosa, en comparación con la evaluación temprana?

   R. En el siguiente ejemplo:

```
{with {doble {fun {x} {+ x x}}}
      {double large_expr}}
```

Se aprecia que `large_expr` es ligado a `x` y luego usado dos veces en el cuerpo de `doble`, si usamos evaluación perezosa sin caching tendriamos que evaluar `large_expr` dos veces y usando caching (una solución más inteligente) aún tendriamos que poner un marcador y verificar lo cada vez para no volver a evaluar de nuevo la expresión.

¿En que casos es útil?

R. Laziness es útil cuando ciertas expresiones que se definen nunca llegan a ser evaluadas, por ej. ciertos parámetros en una función que no se usan por que existe una condición en la cual no son necesarios para ejecutar la respuesta de la función. Otro caso muy importante es que con laziness nosotros podemos construir expresiones sin fin aparente como por ejemplo lista infinitas, ejemplo de ello es la función `fibs` que se definio en clase.

3. Sobre el lenguaje C. En cada caso, explique su respuesta e ilustre con ejemplos si necesario.

   a) (2 pts/6) ¿Se puede decir que las funciones son de primera clase? ¿de orden superior?

   R. Las funciones en C son un caso particular, ya que estas apesar de ser de primer orden pueden ser referenciadas en memoria, por tanto esto permite que se las pueda recibir como argumento en una función (la función recibe un `int` que es el puntero a la función recibida) o que pueden ser retornadas como tal mediante punteros. Este compportamiento se debe a que la memoria en C es de primera clase y degenera en este tipo de comportamientos.

   b) (2 pts/6) ¿Es cierto que C es seguro en tipo?

   R. Esto es totalmente falso, ya que para que un lenguaje sea seguro en tipos debe asegurar que si sus expresiones son declaras de un tipo, por ej. `int` que representa un entero, mantengan ese tipo durante todo su ciclo de vida. Lo cual no ocurre en C, ya que un valor `int` puede ser un entero o puede ser un puntero a una estructura o a una función, que claramente no son del tipo entero.

   c) (2 pts/6) ¿Es posible implementar un recolector de basura automático para C?

   R. No es posible, debido al uso especial de memoria que ejemplificamos en los incisos anteriores. Más precisamente el GC no podría diferenciar en memoria si algo es realmente información o basura. Además también existe el problema de no saber que espacio de memoria realmente se usa, ya que en algunos casos por ejemplo se usan menos bits de los que se necesitan y en otros más de lo que se especifico (arreglos por ej.)

4. La empresa LoHagoPorVo propone servicios para manejar documentos tributarios electrónicos, haciendose cargo de los tramites con el SII por parte de sus clientes. Expone los siguientes servicios:

   - $s1 :: DocTributario \rightarrow Timbre$
   - $s2 :: Factura \rightarrow ItemSII$

   a) (0,3 pts/1,5) Recuerde la regla de tipo que caracteriza que una función $A \rightarrow B$ sea subtipo de otra función $A' \rightarrow B'$.

$$\frac{\Gamma \vdash A' <: A \qquad \Gamma \vdash B <: B'}{\Gamma \vdash \quad A \to B <: A' \to B'}$$

b) Considerando las siguientes relaciones de tipos:

- $Boleta <: DocTributario$, $Factura <: DocTributario$
- $Timbre <: ItemSII$, $Certificado <: ItemSII$

explique si la empresa puede actualizar, sin causar problemas en sus clientes existentes, su servicio $s1$ con uno de los siguientes servicios:

1) $s1a :: Boleta \to Timbre$

   R. (0,3 pts/1,5) No es posible, por que $DocTributario$ no es subtipo de $Boleta$.

2) $s1b :: DocTributario \to ItemSII$

   R. (0,3 pts/1,5) No es posible, por que la relación $ItemSII <: Timbre$ no existe.

y su servicio $s2$ con uno de los siguientes servicios:

1) $s2a :: DocTributario \to ItemSII$

   R. (0,3 pts/1,5) Si es posible, dado que $Factura <: DocTributario$ y $ItemSII <: ItemSII$ por reflexividad.

2) $s2b :: DocTributario \to Certificado$

   R. (0,3 pts/1,5) Si es posible, dado que $Factura$ es subtipo de $DocTributario$ y $Certificado$ es subtipo de $ItemSII$.