Universidad Nacional Autónoma de México
Lenguajes de Programación
Tercer Examen Parcial
Semestre 2023-2
30 de mayo de 2023

Nombre: _____ Calificación: _____ Puntos extra: _____

1. (1 pt.) Del siguiente código en Racket:

```
(define (filter-neg l)
   (cond
     ((nempty? l) empty)
     (else
      (if (< (nfirst l) 0)
          (cons (nfirst l) (filter-neg (nrest l)))
           (filter-neg (nrest l))))))
```

   a) Convierte el código anterior a CPS.

   b) ¿Qué regresa la función que convertiste a CPS cuando recibe la lista `'(0 1 -1 0 -4 1 -2)`?

2. (1 pt.) Da la expresión asociada a la continuación y el resultado de dicha expresión, para cada uno de los siguientes códigos:

```
I.    (- (+ 5 (call/cc
                  (lambda (k)
                      (begin
                          (set! *k* k)
                           (k (* 6 3))))))
        12)

II.   (*k* (* 3 4))
```

3. (1 pt.) Explica al menos dos ventajas de cada uno de los tipos de polimorfismo vistos en clase y da ejemplos de cada uno de ellos.

4. (1 pt.) Para los términos siguientes en el lenguaje usado en clase da la prueba formal de la preservación de sus tipos:

   a) Expresión condicional *if* (usando el comportamiento del if de Haskell).

   b) Expresión *aplicación de función* (fa).

5. (1 pt.) Da la demostración del Teorema de Progreso para los términos siguientes en el lenguaje usado en clase:

   a) Expresión suma.

6. (1 pt.) Da el juicio de tipo para la siguiente expresión.

```
{with {x   2}
   {{fun {y} {if true 0 {+ y x}}} 3}}
```

7. (1 pt.) Realiza la inferencia de tipos de la siguiente expresión, mencionando al término de la inferencia, los tipos de cada una de las variables de la función.

```
(define foo
   (lambda (lst item)
       (cond
         ((nempty? lst) nempty)
         ((nequal? item (nfirst lst)) (nrest lst))
         (else (ncons (nfirst lst) (foo (nrest lst) item))))))
```

8. (1 pt.) Utiliza el algoritmo de unificación visto en clase en la expresión:

```
((lambda (x) (+ x x)) 2)
```

9. (1 pt.) Da las sentencias de variables de tipo para las siguientes funciones de Racket:

   a) empty? (recibe una lista y si es la lista vacía regresa el valor booleano de $true$ y $false$ en otro caso).

   b) filter-neg (recibe una lista de números enteros y regresa la lista solo con los enteros negativos).

10. (1 pt.) Modifica la siguiente expresión utilizando el Combinador de Punto Fijo Y:

```
(let ([f (lambda (n)
             (if (zero? (modulo n 2))
                 #t
                (f (- n 1))))])
(f 3))
```

   a) ¿Por qué utilizando el Combinador de punto fijo Y para implementar recursión en Racket no funciona como debería?

   b) ¿Existe algún otro Combinador de punto fijo que solvente este problema?

Algoritmo de Unificación

1. Si X e Y son constante idénticas, no se hace nada.
2. Si X e Y son identificadores idénticos, no se hace nada.
3. Si X es un identificador, reemplaza todas las ocurrencias de X por Y tanto en el stack como en la sustitución, y añade $X \mapsto Y$ en la sustitución.
4. Si Y es un identificador, reemplaza todas las ocurrencias de Y por X tanto en el stack como en la sustitución, y añade $Y \mapsto X$ en la sustitución.
5. Si X es de la forma $C(X_1, X_2, ..., X_n)$ para algún constructor C, e Y es de la forma $C(Y_1, Y_2, ..., Y_n)$ (i.e. tienen el mismo constructor), entonces agrega $X_i = Y_i$ para toda $1 \leq i \leq n$ en el stack.
6. En cualquier otro caso, X e Y no se unifican y se reporta un error.

¡Éxito!