Control 1 – Lenguajes de Programación

Departamento de Ciencias de la Computación

Universidad de Chile

Profesor: Éric Tanter

19 de Abril del 2010

2 horas / 2 puntos por pregunta

1. Considere la siguiente función con nombre misterioso:

```
(define @@@
  (lambda (l)
    (cond
      ((null? l) 1)
      ((atom? (car l))
       (@@@ (cdr l)))
      (else
        (cond
          ((> (@@@ (cdr l))
              (add1 (@@@ (car l))))
           (@@@ (cdr l)))
          (else
            (add1 (@@@ (car l)))))))))
```

   a) ¿Que retorna esta función cuando aplicada a la lista `((1) 2 3)`? a la lista `(1 (2 ((3))))`? Describa en terminos generales lo que hace esa función y proponga un nombre adecuado para ella.

   b) Identifique las expresiones que se calculan más de una vez en la función. Use `let` para eliminar esa redundancia. Asegurese que su nueva definición es correcta.

   c) ¿Como el régimen de substitución asociado a `let` afecta la pregunta anterior? Comente.

   d) Finalmente, identifique un patrón común, interesante de abstraer en una función auxiliar. Nombre y defina esa función auxiliar, y actualize la definición de la función misteriosa (si el `let` ya no es necesario, elimínelo).

2. Considere un lenguaje con la siguiente gramática:

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

   a) Defina en Scheme la función `free-vars` que retorna la lista de variables libres de una expresión dada. No se olvide de partir escribiendo su firma y al menos un test por cada tipo de expresión.

   b) Recuerde la firma de la función `map` de Scheme y ilustre su uso.

   c) Extienda su definición para que `free-vars` funcione con `let` multiples: `(let ((<id> <expr>)...) <expr>)`. Se sugiere hacer uso de `map`.

3. Disponemos de un dispositivo liviano cuyo procesador es basado en una pila y que solamente soporta las siguientes instrucciones:

   - `push num`: pone el número `num` en la pila
   - `add`: adiciona los dos números encima de la pila
   - `sub`: subtrae los dos números encima de la pila

   `add` y `sum` consumen los números y ponen el resultado en la pila.

   a) Definan en PLAI Scheme un compilador (parser + generador de código) del lenguaje AE al procesador basado en pila. El output del compilador es la lista (plana) de instrucciones a ejecutar por el procesador. Por ejemplo:

   ```
   > (compile '(+ 3 5))
   (push 3 push 5 add)
   ```

   b) Queremos extender el lenguaje fuente para soportar variables locales, con `let`: eg. `(- 3 (let (x 10) (+ x x)))`
   Sin embargo, el procesador minimalista no soporta registros. ¿Es posible modificar el compilador para compilar ese lenguaje extendido? Si sí, entonces hágalo. Si no, explique por qué.