Control 1 – Lenguajes de Programación (CC41A)

Departamento de Ciencias de la Computación – Universidad de Chile

Profesor: Éric Tanter

14 de Abril 2008

Evaluación: 1.5 puntos por pregunta

1. **Filtros y árboles**

   - Implemente en Scheme la función `filter-<-` que retorna, en una lista plana, todos los elementos menores que un cierto valor:

     ```
     (filter-<- 5 '(1 3 8 4 6)) --> (1 3 4)
     ```

   - Extienda ahora esta función para que funcione con un árbol cualquiera:

     ```
     (filter-<- 5 '(1 3 (8 4 ((6))))) --> (1 3 (4 ((()))))
     ```

   - Generalize la función anterior, definiendo ahora una función `filter` parametrizada por la condición de filtro. Luego redefine `filter-<-` y define `filter->-` usando la función genérica `filter`.

2. **Substitución.** Un compañero (`A`) suyo propone el siguiente intérprete del lenguaje `WAE` que utiliza el método de substitución visto en clases

   ```
   (define (interp exp)
     (type-case WAE exp
       [(num (n) ...]
       [(add (l r) ...]
       [(with (bound-id named-expr bound-body) ...]
       [(id (v) (error 'interp "free identifier")])))
   ```

   Otro alumno (`B`) del curso sugiere que esta función es incorrecta, dado que nunca se podrá utilizar un identificador porque entregará un error. ¿Quién(es) está(n) equivocado(s)?
   Explique a este(os) alumno(s) con sus palabras por qué está(n) equivocado(s), de tal manera que pueda(n) entender su(s) error(es).

3. Dada la siguiente gramática concreta:

   ```
   (define-type ALE
     [id (v VAR?)]
     [add (l ALE?) (r ALE?)]
     [sub (l ALE?) (r ALE?)])
   ```

```
(define-type VAR
   [var (coef number?) (x symbol?)])
```

Implemente la función `simp :: ALE ->ALE` que simplifica la expresión algebraica:

```
(simp (parse '{+ {3 a} {- {2 b} {- {1 a} {3 b}}}}))
 -> {+ {2 a} {5 b}}
```

Note que:

```
{3 a} = 3*a
{- a b} = a - b
```

4. **Representando números con funciones.** Esta pregunta tiene como objetivo que entienda como se definen los número naturales en el λ-cálculo[1]. Al respecto:

   - Los número son símbolos para contar cosas. Dada la siguiente base de numeros

     ```
     zero = (lambda(f)(lambda(x) x))
     one = (lambda(f)(lambda(x)(f x)))
     two = (lambda(f)(lambda(x)(f (f x))))
     ```

     Deduzca la forma para un número n cualquiera.
     Obs: Para que se convezca que esta representación funciona piense que `f` es la función `add1` y `x` como 0.

   - Escriba la función `succ` que entrega el siguiente número al dado (en su notación funcional!)

   - Escriba las funciones `add` y `mult`. Para ello note que:

$$m + n = \overbrace{1 + 1 + ... + 1}^{n-veces} + m$$

$$m * n = \overbrace{m + m + ... + m}^{n-veces} + 0$$

[1] El λ-cálculo sólo cuenta con tres tipos de expresiones: variable, definición de función, y aplicación de función.

Pauta

1. **Filtros y árboles**
2. **Substitución.**
   - **(1.5 ptos)** Dado que se utiliza el método de substitución visto en clases, cada vez que esta función se encuentre con un identificador lo reemplazará por el correspondiente valor. Si la función de interpretación se encuentra con un id, significa que éste nunca fue substituido, luego entregar el error es la acción correcto. Por tanto el compañero B es quien está equivocado.
3. **Representando números con funciones.**
   - **(0.2 ptos)** Forma para n cualquiera

$$n = (lambda(f)(lambda(x)\overbrace{(f(f...(fx)))}^{n-veces}))$$

   - **(0.3 ptos)** Sucesor

```
(define (succ n)
   (lambda (f)
      (lambda (x) (f ((n f) x)))))
```

   - **(0.5 ptos)** Adición

```
(define (add n)
   (lambda (m)
      ((n succ) m)))
```

   - **(0.5 ptos)** Producto

```
(define (mult n)
   (lambda (m)
      ((n (sum m)) zero)))
```