Control 1 – Lenguajes de Programación (CC41A)

Departamento de Ciencias de la Computación – Universidad de Chile

Profesor: Éric Tanter

9 de Abril 2007

Evaluación: 1 punto por pregunta

1. Implemente en Scheme la función `revrel` que invierte una relación dada (una relación es una lista de pares):

```
(revrel '((1 2)))             --> ((2 1))
(revrel '((1 2) (3 4) (5 6))) --> ((2 1) (4 3) (6 5))
```

2. Considere la siguiente implementación de la función de substitución `subst` del lenguaje WAE visto en clases:

```
;; subst: WAE symbol WAE -> WAE
(define (subst expr sub-id val)
  (type-case WAE expr
    (num (n) expr)
    (add (l r) (add (subst l sub-id val)
                    (subst r sub-id val)))
    (sub (l r) (sub (subst l sub-id val)
                    (subst r sub-id val)))
    (with (bound-id named-expr bound-body)
          (if (symbol=? bound-id sub-id)
              expr
              (with bound-id
                    named-expr
                    (subst bound-body sub-id val))))
    (id (v) (if (symbol=? v sub-id) val expr))))
```

   Con esta función de substitución, el calculador se cae en las dos expresiones siguientes:

   a) `{with {x 5} {with {y x} y}}`

   b) `{with {x 5} {with {x x} x}}`

   Para cada caso explique porqué se cae el calculador, y luego de una versión corregida de `subst`.

3. ¿Que régimen de substitución implementa el siguiente calculador? ¿Que habría que cambiar si uno quisiera pasar al otro régimen?

```
;; calc : WAE -> number
(define (calc expr)
  (type-case WAE expr
     (num (n) n)
     (add (l r) (+ (calc l) (calc r)))
     (sub (l r) (- (calc l) (calc r)))
     (with (bound-id named-expr bound-body)
           (calc (subst bound-body
                        bound-id
                        named-expr)))
     (id (v) (error 'calc "free identifier"))))
```

4. ¿Puede probar que el régimen de substitución en un lenguaje como WAE no afecta el resultado de la evaluación de un programa? ¿Para que lenguajes podría tener un efecto, tal que cambiar de régimen de substitución cambie el valor de una expresión? De un ejemplo.

5. Considere un lenguaje donde la ejecución del siguiente programa retorna el valor 5:

```
(define (f p) n)
(local ((define n 5))
  (f 10))              --> retorna 5
```

   ¿Como caracterizaría a tal lenguaje? ¿En que difiere de lo que esperaría un programador Java? ¿Que modificación haría a su interprete para volver a la semantica esperada?

6. En un intento de implementar el interprete del lenguaje F1WAE, tenemos errores con ciertas expresiones con `with`. Aquí va su interprete:

```
;; interp : F1WAE listof(fundef) DefrdSub -> number
(define (interp expr fun-defs ds)
  (type-case F1WAE expr
    (num ...) (add ...)
    (with (bound-id named-expr bound-body)
          (interp bound-body fun-defs
                  (aSub bound-id
                        (interp named-expr fun-defs ds)
                        (mtSub))))
    (id ...) (app ...)))
```

   De *a)* un ejemplo de expresión con `with` que sí funciona, *b)* un ejemplo de expresión con `with` que no funciona, *c)* la implementación corregida de la interpretación del `with`.

Pauta control 1

-- P1 ----------

```
(define (revrel list)
  (cond
    ((null? list) '())
    (else (cons (swap (car list)) (revrel (cdr list))))
    ))

(define (swap pair)
  (list (cadr pair) (car pair)))
```

-- P2 ----------

(a) Se cae porque (en el else del if(symbol=?)) se crea un with de la forma {with {y x} ...}, es decir, no se substituye el valor de "x" en la "named-expr".

(b) Se cae porque (en el if(symbol=?)) se devuelve la expresión with original, sin subtituir en la "named-expr".

Versión corregida:

```
if(...)
  (with bound-id (subst named-expr sub-id val) bound-body)
else
  (with bound-id (subst named-expr sub-id val) subst(bound-body sub-id val)
```

-- P3 ----------

Lazy, pues la "named-expr" NO se calcula antes de ser substituir.
Para pasar a eager, habría que cambiar (en la linea 10):

named-expr --> (num (calc named-expr))

-- P4 ----------

(a) La unica razon es el hecho de que las expresiones de WAE siempre dan el mismo resultado que uno la evalua 1 o 1000 veces.

(b) Lenguajes donde una misma expresion no da siempre el mismo resultado cuando evaluada varias veces. Es decir, lenguajes con estado compartido variable,

o lenguajes con funciones como "random(seed)" que a cada evaluacion retornan lo mismo.

```
> (c) Java:
> int i = 1;
> int j = i + 1;
> i = 0;
> int k = i + 1;
> print(j + " - " + k);
```

ok, otro con random, otro con variable global, etc.

-- P5 ----------

(a) Lenguaje con scope dinámico.

(b) En que Java tiene scope estático y por lo tanto la variable "n" depende solamente del scope donde está inserta y este scope está dado por la sintaxis,

no por la semántica. (o sea, difiere en el sentido en que un programador Java esperaria que el programa ni siquiera compilara (undefined identifier: n))

(c) Habría que evitar pasar el repositorio de substituciones deferridas actual cuando se hace la interpretación. Es decir, empezar uno nuevo (vacio) cada

ver que se interpreta una función al momento de extenderlo con la
substitucion del parametro formal de la funcion con el parametro actual

-- P6 ----------

(a) {with {x 1} x}

(b) {with {x 1} {with {y 1} x}}

(c) Hay que cambiar la linea 9:

(mtStub) --> (ds)