

# Lenguajes de Programación 2020-1

## Facultad de Ciencias UNAM

### Tarea Examen Parcial 4

Sandra del Mar Soto Corderi   Edgar Quiroz Castañeda

Fecha de entrega: 2 de diciembre de 2019

Esta tarea vale 8 puntos sobre el parcial 4, la calificación se completa con una pregunta presencial el 29 de noviembre de 2019

1. (1pt) Se desea implementar una función `ct` que reciba un árbol heterogéneo de naturales o booleanos y devuelva la conjunción de sus elementos siempre y cuando todos sean booleanos y en otro caso devuelva el valor `n+1`, donde `n` es el primer natural encontrado en el árbol. Para este propósito defina una función `ctaux` y una expresión `e` tal que la función `ct` quede implementada como

```
ct t = handle ctaux t with x => e
```

Bosqueje la evaluación de la expresión

```
Node (iszero 9) (Node False Void Void) (Node 5 Void Void)
```

en la máquina  $\mathcal{K}$ .

Puede omitir varios pasos pero no los que involucren el manejo de excepciones.

*Sugerencia:* Es más fácil si define `ctaux` a partir de una función binaria `vand` que realice la conjunción si sus argumentos son booleanos y en caso contrario lance una excepción adecuada de forma que sea manejada por `e`. Puede suponer que existe una función unaria `isbool` que verifica si su argumento es o no un booleano.

#### Solución:

Definiendo `vand`.

```
-- Función auxiliar para mandar excepciones
bool_panic p = if isbool then p else raise(p)

-- Realiza la conjunción si sus argumentos son booleanos o lanza una
-- excepción con el primer argumento que no lo sea.
vand p q = and (bool_panic p) (bool_panic q)
```

Ahora, hay que usar esta función para definir `ctaux`.

```
ctaux Void = True
ctaux (Node r i d) = vand r (vand (ctaux i) (ctaux d))
```

Y usando esto, se define `ct` como

```
ct t = handle ctaux t with x => x + 1
```

Ahora, para evaluar la expresión, es útil definir lo siguiente

```
a = Node (iszero 9) (Node False Void Void) (Node 5 Void Void)
r = iszero 9
i = Node False Void Void
d = Node 5 Void Void
```

Ahora, para evaluar

```

□ > ct a
→β □ > handle (ctaux a) with x => x + 1
→κ handle(-, x.x+1) > ctaux a
→β handle(-, x.x+1) > and (bool-panic r) (vand (ctaux i) (ctaux d))
→κ and(False, -), handle(-, x.x+1) > vand (ctaux i) (ctaux d)
→κ and(False, -), and(False, -), handle(-, x.x+1) > bool-panic(ctaux d)
→κ isbool(-), if(-, ctaux d, raise(ctaux d)) > ctaux d

```

2. (1pt) Considere el siguiente programa, donde suponemos que el lenguaje contiene un operador primitivo `not` para la negación booleana.

```

not letcc k2 in
  iszero(2 + letcc k1 in
    3 + if x = pred 8 then
      4 * pred (continue k2 6)
    else 5 * suc (continue k1 false)
  end
end
end

```

- a) ¿Cuáles son los tipos de `k1` y `k2`?

`k1 : Cont(Integer) k2 : Cont(Bool)`

- b) ¿A qué continuaciones se ligán las variables `k1` y `k2`?

`k1 -->(continue k1 6) k2 -->(continue k2 False)`

- c) ¿A qué se evalúa el programa para `x = 7` y para `x ≠ 7`?

`e[x := 7] →κ* True, e[x /= 7] →κ* True`

3. (1.5pt) Considere la siguiente función `N` que depende de ciertas funciones dadas `f`, `g`, `h`.

```

N 0 = 17
N 1 = f (1 + 13)
N 2 = 22 + (2-3) + 2
N 3 = 22 + (f 3) + 37
N 4 = g 22 (f 4)
N 4 = 22 + (f 4) + 33 + (g y)
N x = h (f x) (44 - y) (g y)

```

Defina la versión `cps` de `N`, denotada por `cpsN`. Para esto, puede suponer definidas las versiones `cps` de `f`, `g`, `h`, denotadas por `cpsf`, `cpsg`, `cpsh`.

*Atención:* Las operaciones aritméticas deben permanecer sin cambios. Es decir, no se piden las versiones `cps` de `+`, `-`.

4. (1.5pt) Considere las siguientes cuatro características para un lenguaje de programación.

- a) Existe un tipo de cadena `String` y se cumple  $\text{Int} \leq \text{String}$  mediante una conversión implícita que transforma a un entero en una cadena.  
Por ejemplo, `123` a `"123"`.
- b) Existe un tipo cadena `String` que cumple que  $\text{String} \leq \text{Int}$  mediante una conversión implícita que transforma cadenas a enteros ignorando los caracteres que no sean dígitos, excepto por el carácter inicial `-`.  
Por ejemplo, `"-0aw23r4"` corresponde a `-234`.
- c) Existe un operador binario `+` que denota a ambas la suma de enteros y la concatenación de cadenas.
- d) Existe un operador binario `=` que denota a ambas la igualdad de enteros y de cadenas.

Para cada par de estas características, discuta si se violan o no los principios fundamentales del subtipado. En caso afirmativo escriba un ejemplo de un programa simple que cause un comportamiento ambiguo o contraintuitivo.

5. (1pt) Usando la definición de números naturales en `Java` `Peso` `Pluma` vista en clase.

- a) Agregue un método `pot` y `leq` para las operaciones de potencia y el orden  $\leq$ .

**Solución:**

```
// Clase de naturales definida en las notas
class Nat extends Object {
    Object p;

    Nat (Object n) { super(); this.p = n;}

    Nat suc() { return new Nat(this); }

    Nat pred() { return (Nat) this.p; }

    Nat suma(Nat n) { return this.pred().sum(n.suc());}

    Nat multi(Nat n) { return n.suma(this.pred().multi(n));}

    // método 'pot'
    Nat pot(Nat n) { return n.flipPot(this) }

    // método auxiliar para calcular 'pot'
    Nat flipPot(Nat n) {return n.multi(this.pred().flipPot(n))}

    // método 'leq'
    Boolean leq(Nat n) {return n.lt(this).not();}

    // método auxiliar para calcular 'leq'
    Boolean lt(Nat n) {return this.pred().lt(n.pred());}

    // método auxiliar para calcular 'not' en Boolean
    Nat inv() {return new Cero(this);}
}

// Clase del Cero definida en las notas
class Cero extends Nat {
    Cero (Object n) {super(n);}

    Nat pred() {return this;}

    Nat suma(Nat n) {return n;}

    Nat multi(Nat n) {return this;}

    // calcular el caso base del método auxiliar 'flipPot'
```

```

    Nat flipPot(Nat n) {return this.suc();}

    // calcular el caso base del método auxiliar 'lt'
    Boolean lt(Nat n) {return new Boolean(n);}

    // método auxiliar para calcular 'not' en Boolean
    Nat inv() {return this.suc();}
}

```

Donde `not` está definido como

```

class Boolean extends Object {
    ...
    Boolean not() {return new Boolean(this.m.inv());}
}

```

b) Modele la clase `Boolean` para el manejo de valores booleanos

- La clase debe extender de `Object`
- El constructor recibirá un objeto de la clase `Nat` para definir su valor. 0 representa `true` y 1 representa `true`.
- Debe tener métodos `true` y `false` que regresen una instancia de `Boolean` según el caso.

**Solución:**

```

class Boolean extends Object {
    Nat m;

    Boolean (Nat n) { super(); this.m = n;}

    Boolean true() { return new Boolean (new Cero(this).suc());}

    Boolean false() { return new Boolean (new Cero(this));}

    Boolean not() { return new Boolean(this.m.inv());}
}

```

6. (2pt) Definina el siguiente lenguaje `MinEAB`

`e ::= n | true | false | e + e | e < e`

en Java `Peso Pluma`.

Hay que seguir los siguientes pasos

a) Defina una clase `Expr` que incluya los siguientes métodos

- `isAtom` que devuelve `true` si la expresión no tiene subexpresiones propias.
- `lsub` que devuelve la subexpresión izquierda de una expresión no atómica.
- `rsub` que devuelve la subexpresión derecha de una expresión no atómica.
- `eval` que devuelve el valor de la expresión.

Esta clase debe ser abstracta en el sentido de que no tiene atributos y por lo tanto sus métodos no tiene cuerpo. Es una interfaz.

**Solución:**

```

class Expr extends Object {

    Boolean isAtom() {return error;}

    Expr lsub() {return error;}

    Expr rsub() {return error;}

    Expr eval() {return error;}
}

```

b) Defina las siguientes clases que implemente a `Expr`

- `NumExpr` que implemente los métodos para manejar números.

**Solución:**

```
class NumExpr extends Expr {
    Nat v;

    NumExpr(Nat w) {super(); this.v = w;}

    Boolean bool() {return new Boolean(new Cero(this));}

    Boolean true() {return this.bool().true();}

    Boolean false() {return this.bool().false();}

    Boolean isAtom() {return this.true();}

    Expr eval() {return this.v;}
}
```

- `BoolExpr` que implemente los métodos para manejar booleanos.

**Solución:**

```
class BoolExpr extends Expr {
    Boolean p;

    BoolExpr(Boolean q) {super(); this.p = q;}

    Boolean bool() {return new Boolean(new Cero(this));}

    Boolean true() {return this.bool().true();}

    Boolean false() {return this.bool().false();}

    Boolean isAtom() {return this.true();}

    Expr eval() {return this.p;}
}
```

- `SumExpr` que implemente los métodos para manejar sumas.

**Solución:**

```
class SumExpr extends Expr {
    Expr i;
    Expr d;

    SumExpr(Expr e1, Expr e2) {super(); this.i = e1; this.d = e2;}

    Boolean bool() {return new Boolean(new Cero(this));}

    Boolean true() {return this.bool().true();}

    Boolean false() {return this.bool().false();}

    Boolean isAtom() {return this.false();}

    Expr eval() {return this.i.eval().suma(this.d.eval());}
}
```

- `LTEExpr` que implemente los métodos para manejar comparaciones de orden.

**Solución:**

```
class LTEExpr extends Expr {
    Expr i;
```

```

Expr d;

LTExpr(Expr e1, Expr e2) {super(); this.i = e1; this.d = e2}

Boolean bool() {return new Boolean(new Cero(this));}

Boolean true() {return this.bool().true();}

Boolean false() {return this.bool().false();}

Boolean isAtom() {return this.false();}

Expr eval() {return this.i.eval().lt(this.d.eval());}
}

```

c) Dé ejemplos de instancias de cada una de las clases anteriores.

d) Extienda `MinEAB` con las expresiones `-e`, `iszero e`.

Con esta nueva definición, cree las siguientes clases

- `NegExpr`
- `IsZero`

**Solución:**

e) Dé ejemplos de instancias de estas dos clases.

f) ¿Cómo se modifican las subclases de `Expr` definidas en puntos anteriores?

Puede suponer definida la clase `Value` (esencialmente `Nat + Bool`) cuyas instancias sean los valores del lenguaje.

Además de otras clases primitivas con los métodos que requiera.

También se puede usar la constante de error en cualquier método.

## 7. Extra (hasta 2pt): Privacidad en Java Peso Pluma

Java proporciona mecanismos para controlar el acceso. Un método o atributo de una clase pueden declararse como *público*, *protegido* o *privado*. En `Java Peso Pluma` es posible agregar este mecanismo como sigue

```

C ::= class C extends C { $\vec{p}$   $\vec{C}$   $\vec{f}$ ;  $K$   $\vec{M}$ } // declaración de clases
M ::= p C m ( $\vec{C}$   $\vec{x}$ ) {return e;} // declaración de métodos
p ::= public | protected | private // modificador de privacidad

```

EL significado intuitivo de los modificadores de privacidad es el siguiente:

- Si un método o atributo de una clase `C` se declara `public`, entonces se permite el acceso desde cualquier lugar.
- Si un método o atributo de una clase `C` se declara `protected`, entonces se permite el acceso únicamente desde métodos de `C` y subclases de `C`.
- Si un método o atributo de una clase `C` se declara `private`, entonces se permite el acceso únicamente desde métodos de `C`.

Extienda la semántica estática para filtrar programas que contengan violaciones de privacidad de acuerdo a las reglas dadas arriba. Especifique claramente cuales reglas de tipado originales se eliminan o se sustituyen por nuevas y cuáles se mantienen.

**Sugerencia:** Cuando se verifique si un método está bien formado en una clase `C`, se debe verificar si la expresión en el cuerpo del método no se refiere a métodos o atributos en otra clase `D` que sean privados o protegidos si  $C \not\leq D$ . Podría necesitarse el paso de algo más en el contexto  $\Gamma$  en el juicio de tipado para expresiones.