optimus

ICL 30-5-2012

Exame Ep. Normal 09/10

1.

a) Object (List<Pair<String, AST>> fields, List<Triple<String, String, AST>> methods)
Integer (int num)
Add (AST l, AST r)
Id (string s)
Call (AST obj, String m, AST arg)
Assign (string s, AST, v)
Decl (string x, AST d, AST b)

type ast = ... | Call of ast * string * ast | ... (ast: expressão)

b)
E ::= [Id1 = E1, ..., Idn = En, I1(id) = E', ... Im(id) = En]
| Num | E + E | Id
| E.I(E) | Id := E | self
| decl Id = E in E

type value Int of int | Object of (string * ref value) list * (string * string * ast) list
type method of (string * string * ast)
(method)

c) let rec eval ast env = (env)
match ast with
Num n → Int n
| Id s → find env s
| Add (l, r) →
(se tiverem self) | Self → find env "self"
| Decl (x, d, b) → let v = eval env d in eval ((x,v)::env) b
|
| Call (o, m, a) → let o' = eval env o in
let a' = eval env a in
let (x, b) = get_method o' m in
eval ((x, a) :: ("self", o')::env) b

| Assign (f, v) → let v' = eval env v in
let f = get_field (find env "self") f in
f := v ; v

let get_method o m =
match o with
Object(fields, methods) →
List.assoc m methods
| _ → raise BLa

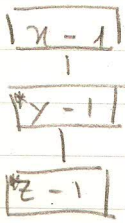
```
let get_field o m =
  match o with
    object (fields, methods) -> List.assoc m fields
  | raise
```

(d) Enumerar os erros de execução possíveis.
Quais os erros que o meu sistema de tipos terá de cobrir

- → Identificador não existente
- → Adição de valores "não inteiros"
- → Uso de "self" fora do contexto de um object
- → chamada de um método com um valor "não objecto"
- → chamada de um método não existente
- → Afectação de um identificador que não é um campo do obj. self
- → Afectação fora do contexto de um objecto

2.

(a)
```
decl x=1 in
decl f= fun y => fun z => x+y+z end end in
decl g= fun x => f(x+1) in
(g(3))(4)
```



x - (2,1)
y - (1,1)
z - (0,1)
endereços

(b) Resultado final com RES. DINÂMICA

eval((g(3))(4), [g=fun → ... , f=fun → ... , x=1])
eval((g(3))(4), ...
eval(f(x+1), [x=3, g... f... x=1]) (4) , [g, f, x])
eval(fun z => x+y+z , [y=4, x=3, ...]) (4)
eval(x+y+z, [z=4, y=4, x=3, ..., x=1]) = 11

optimus

(c) Não Contra exemplo: spaghetti stack

Não podemos usar apenas stacks, pois temos closures tb.

4.

(a)

```
type ty =
    IntT
  | object of (string * ty) list * (string * (string * ty * ty)) list
                                    label      tipo de métodos
```

ICL 31-5-2012

(b) Sistema de tipos

$$\frac{\Delta \vdash e : T_a \quad \Delta, self : T, x : T_x \vdash em : T_m}{\Delta \vdash [a = e, \ldots, m(x) : T_x = em, \ldots] : T}$$

$T = object(a : T_a, \ldots, m : T_x \to T_m, \ldots)$

$$\frac{}{\Delta \vdash Num : Int}$$

$$\frac{\Delta \vdash e : Int \quad \Delta \vdash e' : Int}{\Delta \vdash e \vdash e' : Int}$$

$$\frac{}{\Delta, x : T, \Delta' \vdash x : T}$$

$$\frac{}{\Delta, self : \ldots, \Delta' \vdash self : \ldots}$$

$$\frac{self : \alpha \notin \Delta'}{\text{primeiro}}$$

$$\frac{self : obj(x : T \ldots) \notin \Delta'}{\Delta, x : T, \Delta' \vdash x : T}$$

$$\frac{x : T \notin \Delta', self : T \notin \Delta'}{\Delta, self : obj(x : T), \Delta' \vdash x : T}$$

$$\frac{self, o \notin \Delta'}{\Delta, self : T, \Delta' \vdash self : T}$$

e : T → e tem de ser do tipo T

$$\frac{\Delta \vdash e : obj(\ldots\ m : \sigma \to \tau) \quad \Delta \vdash e' : \sigma}{\Delta \vdash e.m(e') : \tau}$$

$$\frac{\Delta \vdash self : obj(a : \tau) \quad \Delta \vdash e : \tau}{\Delta \vdash a := e : \tau}$$

$$\frac{\Delta \vdash e : \sigma \quad \Delta, x : \sigma \vdash e' : \tau}{\Delta \vdash decl\ x = e\ in\ e' : \tau}$$

(c)

object(a : Int, set : Int → Int, get : Int → Int)

```
def cell = [new(x:Int): = [a = x, set(x:Int):Int = a := x, get(x:Int):Int = a]] in
def point = [x = cell.new(0), y = cell.new(0),
        movex(d:Int):Int = x.set(x.get(0) + d),
        movey(d:Int):Int = x.set(x.get(0) + d)] in point.movex(10)
```

ccell = objet(new : Int → Cell

Cell = object(a : Int, set : Int → Int, get : Int → Int)),

D = cell:ccell, point : object(x : cell, y : cell, movex : Int → Int, movey : Int → Int)

Point

```
check "point.movex(cell.new(0).get(0))" D                    (case call)
    check point D → Object(... movex : Int → Int)            (case Id)
    check cell.new(0).get(0) D                               (case call)
        check cell.new(0)                                    (case call)
            check cell D → ccell = Objet(...new : Int → Cell) (case Id)
            check 0 D → Int                                  (case Num)
        → Cell = Object(...get : Int → Int)
        check 0 D → Int                                      (case Num)
    → Int
→ Int
```

$$Cell = Obj(a : Int,\ get : Int \to Int,\ set : Int)$$

$$\frac{\Delta \vdash 0 : Int}{\dfrac{\Delta \vdash Cell : Obj(new : Int \to Cell)}{\Delta \vdash cell.new(0) : Cell} \qquad \Delta \vdash 0 : Int}$$

$$\frac{\Delta \vdash point : Obj(move : Int \to Int) \qquad \Delta \vdash (cell.new(0)) \vdash get(0) : Int}{\Delta \vdash point.move(cell.new(0).get(0)) : Int}$$