REFERENCIAS Y ESTADOS EN UN LENGUAJE APLICATIVO EAGER

SEMÁNTICA DENOTACIONAL DIRECTA

El conjunto de valores se extiende de la siguiente manera:

V ≈ Vint + Vbool + Vfun + Vref + {unit}

D = ({error, typeerror} + Σ x V)_⊥

Lnorm ahora acepta un par de Σ x V:

Lnorm<s,z> ∈ D

Como antes:

Env = <var> -> V
Vint = Z
Vbool = B

La funciones ahora deben reflejar la posibilidad de cambio del estado, es decir, una función no sólo computa un valor sino además eventualmente modifica el estado. Definimos ahora:

Vref = Rf
Vfun = Σ x V -> D

Como antes disponemos de:

Lint ∈ Vint -> V
Lbool ∈ Vbool -> V
Lref ∈ Vref -> V
Lfun ∈ Vfun -> V
<> ∈ V (<> representa a unit dentro de V
err, tyerr ∈ D

Las funciones que asisten la transferencia de control se adaptan a la nueva funcionalidad:

f ∈ Σ x V -> D
f* ∈ D -> D

f* Lnorm<s,z> = f <s,z>
f* err = err
f* tyerr = tyerr
f* ⊥ = ⊥

f ∈ Σ x Vint -> D
f ∈ Σ x V -> D

fint <s,Lint i> = f <s,i>
fint <s,Lx z> = tyerr (x =/= int)

De manera similar se definen los operadores "sub bool" y "sub fun".

La función semántica se define de la siguiente manera:

[[_]] ∈ <exp> -> Env -> Σ -> D

La semántica de los construcciones típicamente imperativas es la siguiente:

[[skip]]ηs = Lnorm <s,<>>

[[val e]]ηs = (λ<s',r>. if r ∈ dom s' then Lnorm <s',s' r> else err)ref* ([[e]]ηs)

[[ref e]]ηs = (λ<s',z>. Lnorm <[s'|new s':z],Lref (new s')>)* ([[e]]ηs) (new(s) = newref(dom s))

[[e:=e']]ηs = (λ<s',r>. (λ<s'',z>. Lnorm <[s''|r:z],<>>)* ([[e']]ηs'))ref* ([[e]]ηs)

[[e =ref e']]ηs = (λ<s',r>. (λ<s'',r'>. Lnorm <s'',Lbool[r=r']>)ref* ([[e']]ηs'))ref* ([[e]]ηs)

La semántica de 0 o true, es trivial, salvo que hay que promover el resultado para que sea un D (no sólo un Vint o Vbool), el estado por supuesto no se modifica:

[[0]]ηs = Lnorm <s,Lint 0>
[[true]]ηs = Lnorm <s,Lbool V>

Para evaluar -e se evalúa e y se chequea que dé entero (en caso contrario, el subíndice int se encargará de disparar un error de tipos) y que no se haya producido ya algún error (en cuyo caso, el subíndice * se encargará de propagarlo). Si todo anda bien se devuelve el entero correspondiente promoviendolo para que sea un D.

[[-e]]η = (λ<s',i>. Lnorm <s',Lint -i>)int* ([[e]]ηs)

Lo mismo ocurre con la negación lógica:

[[¬e]]η = (λ<s',b>. Lnorm <s',Lbool ¬b>)bool* ([[e]]ηs)

Y también con los operadores binarios:

[[e0+e1]]η = (λ<s',i>.(λ<s'',j>. Lnorm <s'',Lint i+j>)int* ([[e1]]ηs'))int* ([[e0]]ηs)

$$[[\text{if } e \text{ then } e0 \text{ else } e1]]\eta = (\lambda<s,b>.\begin{cases}[[e0]]\eta s & \text{si } b\\ [[e0]]\eta s & \text{c.c.}\end{cases})\text{bool}^*([[e]]\eta s)$$

Los casos del cálculo lambda se adaptan como sigue

[[v]]ηs = Lnorm <s,η v>
[[e0 e1]]η = (λ<s',f>. f* ([[e1]]ηs'))fun* ([[e0]]ηs)
[[λx.e]]η = Lnorm <s,Lfun (λ<s',z>. [[e]][η|v:z]s')>

Finalmente, las ecuaciones triviales

[[error]]ηs = err
[[typeerror]]ηs = tyerr

Finalmente, la semántica del letrec:

[[letrec v ≡ λu.e in e']]ηs = [[e']]η's

donde

η' = [η|v:Lfun f]
f = λ<s',z>. [[e]][η|v:Lfun f|u:z]s'

PROPIEDADES:
---

1. [[e;e']]ηs = (λ<s',z>.[[e']]ηs')*([[e]]ηs)

2. [[xe]]ηs = g*([[e]]ηs) si ηx = Lfun g

3. Si [[e]]ηs = Lnorm<s',z> entonces

        [[newvar v:=e in e']]ηs = [[e']][η|v:Lref r][s'|r:z]   donde r = newref (dom s')

Dicho de otra manera:

[[newvar v:=e in e']]ηs =

(λ<s',z>.[[e']][η|v:Lref (new s')][s'|new s':z])* ([[e]]ηs) (new s = newref (dom s))

Prueba de 3:

Si [[e]]ηs = Lnorm<s',z> y r = newref (dom s') entonces

[[newvar v:=e in e']]ηs =
[[(λv.e')(ref e)]]ηs =
(λ<s',f>. f* ([[ref e]]ηs'))fun* (Lnorm<s,Lfun(λ<s'',z>. [[e']][η|v:z]s'')>) =
(λ<s'',z>. [[e']][η|v:z]s'')* ([[ref e]]ηs) =
(λ<s'',z>. [[e']][η|v:z]s'')* ((λ<s',z>. Lnorm <[s'|new(s'):z],Lref (new s')>)* ([[e]]ηs)) =
(λ<s'',z>. [[e']][η|v:z]s'')* ((λ<s',z>. Lnorm <[s'|new(s'):z],Lref (new s')>)* (Lnorm<s',z>) =
(λ<s'',z>. [[e']][η|v:z]s'')* (Lnorm <[s'|r:z],Lref r>) =
[[e']][η|v:Lref r][s'|r:z]

EXTENSIONES DE LOS LENGUAJES APLICATIVOS PUROS

Volvemos ahora a trabajar con los lenguajes aplicativos para estudiar su extensión con diversas formas de construcciones presentes en los lenguajes usuales. Aunque estas son independientes del manejo de estados (y pueden ser hechas sobre ISWIM), vamos a trabajar con los lenguajes aplicativos puros para poder estudiar las variantes normal e eager.

TUPLAS

Se agregan expresiones para tuplas:

<exp> ::= <<exp>,...,<exp>> | <exp>.<tag>
<tag> ::= <natconst>

Evaluación

<cfm> ::= <intcfm> | <boolcfm> | <funcfm> | <tuplecfm>

Evaluación Eager

<tuplecfm> ::= <<cfm>,...,<cfm>>

 e1 => z1 ... en => zn

<e1,..,en> => <z1,..,zn>

e => <z1,..,zn>
--- k < n, [k] es la notación para k en el lenguaje
  e.[k] => zk

Evaluación Normal

<tuplecfm> ::= <<exp>,...,<exp>>

La regla

```
------------------------
<e1,..,en> => <e1,..,en>
```

no se agrega ya que es un caso particular de la regla z => z para formas canónicas.

```
e => <e1,..,en>     ek => z
--------------------------- k < n, [k] es la notación para k en el lenguaje
        e.[k] => z
```

Semántica Denotacional

Ahora tendremos

V ≈ Vint + Vbool + Vfun + Vtuple

Además se tiene Ltuple ∈ Vtuple -> V y para cualquier función f ∈ Vtuple -> D, ftuple ∈ V -> F.

Semántica Denotacional Eager

Vtuple = V*

```
[[<e1,..,en>]]η = (λz1∈V. .. (λzn∈V. Lnorm (Ltuple <z1,..,zn>))* ([[en]]η) ..)* ([[e1]]η)
                          ⎧Lnorm tk   si k < #t
[[e.[k]]]η = (λt∈Vtuple. ⎨                    )tuple* ([[e]]η)
                          ⎩tyerr      c.c.
```

Semántica Denotacional Normal

Vtuple = D*

```
[[<e1,..,en>]]η = Lnorm (Ltuple <[[e1]]η,..,[[e1]]η>)
                          ⎧tk         si k < #t
[[e.[k]]]η = (λt∈Vtuple. ⎨                    )tuple* ([[e]]η)
                          ⎩tyerr      c.c.
```

DEFINICIONES LOCALES Y PATRONES

Agregamos al lenguaje la posibilidad de definir localmente y notación para patrones:

```
<exp> ::= let <pat> ≡ <exp>, ..., <pat> ≡ <exp> in <exp>
      | λ<pat>.<exp>
<pat> ::= <var> | <<pat>,..,<pat>>
```

Así podemos escribir λ<u,<v,w>>.u v w en vez de λt. t.0 t.1.0 t.1.1

Para no tener que definir evaluación y semántica denotacional eager y normal para esta extensión, nos conformamos con ver que estas nuevas expresiones se pueden definir en términos de las que ya existían. Es sólo azúcar sintáctico:

λ<p1,..,pn>.e es lo mismo que λv.let p1 ≡ v.0, .., pn ≡ v.[n-1] in e

donde v es una variable nueva (no ocurre libre en e ni en ninguno de los patrones)

let p1 ≡ e1, .., pn ≡ en in e es lo mismo que (λp1..λpn.e) e1 .. en

Aplicando repetidamente estas dos transformaciones podemos eliminar los patrones que no sean variables y las definiciones locales (let) obteniendo una expresión cuya semántica ya está definida.

Tener en cuenta que cuando n = 0, let p1 ≡ e1, .., pn ≡ en in e quedaría let in e, esto en realidad es

directamente la expresión e (el let in sin nada al medio es como si no existiera).

ALTERNATIVAS

Se agregan expresiones para tuplas:

<exp> ::= @ <tag> <exp> | sumcase <exp> of (<exp>,..,<exp>)

Evaluación

<cfm> ::= <intcfm> | <boolcfm> | <funcfm> | <tuplecfm> | <altcfm>

Evaluación Eager

<altcfm> ::= @ <tag> <cfm>

$$\frac{e \Rightarrow z}{@\,[k]\ e \Rightarrow @\,[k]\ z}$$

$$\frac{e \Rightarrow @\,[k]\ z \qquad ek\ z \Rightarrow z'}{\text{sumcase } e \text{ of } (e0,..,en\text{-}1) \Rightarrow z'} \quad k < n$$

[k] es la notación para k en el lenguaje

Evaluación Normal

<altcfm> ::= @ <tag> <exp>

La regla

$$\frac{}{@\,[k]\ e \Rightarrow @\,[k]\ e}$$

no se agrega ya que es un caso particular de la regla z => z para formas canónicas.

$$\frac{e \Rightarrow @\,[k]\ e' \qquad ek\ e' \Rightarrow z}{\text{sumcase } e \text{ of } (e0,..,en\text{-}1) \Rightarrow z} \quad k < n$$

[k] es la notación para k en el lenguaje

Semántica Denotacional

Ahora tendremos

V ≈ Vint + Vbool + Vfun + Vtuple + Valt

Además se tiene Lalt ∈ Valt -> V y para cualquier función f ∈ Valt -> D, falt ∈ V -> F.

Semántica Denotacional Eager

Valt = N x V

$$[[@\,[k]\ e]]\eta = (\lambda z \in V.\ Lnorm\ (Lalt\ k\ z))^*\ ([[e]]\eta)$$

$$[[\text{sumcase } e \text{ of } (e0,..,en\text{-}1)]]\eta = (\lambda \langle k,z \rangle \in Valt.\ \begin{cases} (\lambda f \in Vfun.\ f\ z)^{fun^*}\ ([[ek]]\eta) & \text{si } k < n \\ tyerr & \text{c.c.} \end{cases})^{alt^*}\ ([[e]]\eta)$$

Semántica Denotacional Normal

Valt = N x D

```
[[@ [k] e]]η = Lnorm (Lalt k ([[e]]η))

[[sumcase e of (e0,..,en-1)]]η = (λ<k,d>∈Valt. ⎧(λf∈Vfun.f d)fun* ([[ek]]η)   si k < n
                                               ⎩tyerr                          c.c.
                                                                                )alt* ([[e]]η)
```