Práctica: Cálculo ς

Departamento de Computación, FCEyN, UBA

2018

Igualdad de objetos

Símil Ejercicio 8

Decir si los siguientes pares de objetos son considerados equivalentes:

- $$\begin{aligned} o_1 &\stackrel{\text{def}}{=} [m_1 = \varsigma(x)x.m_2,\ m_2 = \varsigma(x)x.m_1] \\ o_2 &\stackrel{\text{def}}{=} [m_2 = \varsigma(z)z.m_1,\ m_1 = \varsigma(v)v.m_2] \end{aligned}$$

- $$\begin{aligned} o_1 &\stackrel{\text{def}}{=} [m_1 = \varsigma(x)x.m_2,\ m_2 = \varsigma(x)x.m_1] \\ o_2 &\stackrel{\text{def}}{=} [m_3 = \varsigma(z)z.m_1,\ m_1 = \varsigma(v)v.m_3] \end{aligned}$$

- $$\begin{aligned} o_1 &\stackrel{\text{def}}{=} [m_1 = \varsigma(x)\lambda(y)y.l] \\ o_2 &\stackrel{\text{def}}{=} [m_1 = \varsigma(x)\lambda(z)z.l] \end{aligned}$$

Codificación de lambda cálculo $[\![\_]\!] : M \to a$

$$
\begin{aligned}
[\![x]\!] &\stackrel{\text{def}}{=} x \\
[\![MN]\!] &\stackrel{\text{def}}{=} ([\![M]\!].arg := [\![N]\!]).val \\
[\![\lambda x.M]\!] &\stackrel{\text{def}}{=} [\, val = \varsigma(y)[\![M]\!]\{x \leftarrow y.arg\}, \\
&\qquad arg = \varsigma(y)y.arg \,]
\end{aligned}
$$

Semántica operacional

Símil Ejercicio 9

Mostrar cómo reduce la siguiente expresión:

$$([val = \varsigma(x)x.arg, arg = \varsigma(x)x.arg].arg := []).val$$

Semántica operacional

Valores

$$v ::= [l_i = \varsigma(x_i)b_i{}^{i \in 1..n}]$$

Reducción *big-step* $\longrightarrow$

$$\frac{}{v \longrightarrow v}\ [\textsc{Obj}]$$

$$\frac{a \longrightarrow v' \quad v' \equiv [l_i = \varsigma(x_i)b_i{}^{i \in 1..n}] \quad b_j\{x_j \leftarrow v'\} \longrightarrow v \quad j \in 1..n}{a.l_j \longrightarrow v}\ [\textsc{Sel}]$$

$$\frac{a \longrightarrow [l_i = \varsigma(x_i)b_i{}^{i \in 1..n}] \qquad j \in 1..n}{a.l_j \Leftarrow \varsigma(x)b \longrightarrow [l_j = \varsigma(x)b,\ l_i = \varsigma(x_i)b_i{}^{i \in 1..n-\{j\}}]}\ [\textsc{Upd}]$$

`if-then-else` como método de los booleanos

Variante ejercicio 11

- Codificar los objetos `true` y `false` que proveen tres métodos `if`, `then` y `else`. El comportamiento es el siguiente: `true.if` evalúa `true.then`, mientras que `false.if` evalúa `false.else`
- Usando los objetos definidos previamente, mostrar cómo codificar el condicional `if` b `then` c `else` d.

(Stateless) Traits

- Los vamos a representar como una colección de pre-métodos:
  - pre-método: $\varsigma(\mathtt{t})\lambda(y)b$ con $\mathtt{t} \notin \mathsf{fv}(\lambda(y)b)$ (no usan el parámetro `self`).
  - Recordar que en este caso podemos omitir $\varsigma(\mathtt{t})$ y escribir $\lambda(y)b$.
  - Luego, $\mathtt{t} = [l_i = \lambda(y_i)b_i{}^{i\in 1..n}]$ es un trait.
- A partir de un trait $\mathtt{t} = [l_i = \lambda(y_i)b_i{}^{i\in 1..n}]$ podemos definir un constructor de objetos (cuando `t` es completo).

$$new \stackrel{\text{def}}{=} \lambda(z)[l_i = \varsigma(s)z.l_i(s)^{i\in 1..n}]$$

$$\begin{array}{rcl} o & \stackrel{\text{def}}{=} & new\ \mathtt{t} \\ & \longrightarrow & [l_i = \varsigma(s)\mathtt{t}.l_i(s)^{i\in 1..n}] \\ & \approx & [l_i = \varsigma(y_i)b_i{}^{i\in 1..n}] \end{array}$$

(Stateless) Traits

$$\begin{aligned}
\texttt{CompT} \quad &\stackrel{\text{def}}{=} \quad [\ eq \ = \varsigma(t)\lambda(x)\lambda(y) \\
&\qquad\qquad if\,(x.comp(y)) == 0 \ then\ \texttt{true}\ else\ \texttt{false}, \\
&\qquad\quad leq = \varsigma(t)\lambda(x)\lambda(y) \\
&\qquad\qquad if\,(x.comp(y)) < 0 \ then\ \texttt{true}\ else\ \texttt{false}]
\end{aligned}$$

$$new \quad \stackrel{\text{def}}{=} \quad \lambda(z)[l_i = \varsigma(s)z.l_i(s)^{i\in 1..n}]$$

$$\begin{aligned}
new\ \texttt{CompT} \longrightarrow \quad & [\ eq \ = \varsigma(s)\texttt{CompT}.eq(s), \\
& \ \ leq = \varsigma(s)\texttt{CompT}.leq(s),\,] \\
\approx \quad & [\ eq \ = \varsigma(s)\lambda(y) \\
&\qquad if\,(s.comp(y)) == 0 \ then\ \texttt{true}\ else\ \texttt{false}, \\
& \ \ leq = \varsigma(s)\lambda(y) \\
&\qquad if\,(s.comp(y)) < 0 \ then\ \texttt{true}\ else\ \texttt{false}\ \ ]
\end{aligned}$$

- Este objeto es inutilizable (porque `CompT` no es completo).

Clases

- Una clase es un *trait* (completo) que además provee un método new.

$$\mathsf{c} \stackrel{\text{def}}{=} \begin{array}{l} [\ new = \varsigma(z)[l_i = \varsigma(s)z.l_i(s)^{i\in 1..n}], \\ \ \ l_i \quad\;\; = \lambda(s)b_i{}^{i\in 1..n}] \end{array}$$

- Luego,

$$\begin{array}{rcl} o & \stackrel{\text{def}}{=} & \mathsf{c}.new \\ & \longrightarrow & [l_i = \varsigma(s)\mathsf{c}.l_i(s)^{i\in 1..n}] \\ & \approx & [l_i = \varsigma(s)b_i{}^{i\in 1..n}] \end{array}$$

Definiendo Clases

Contador

Definir una clase `Contador`, cuyas instancias provean dos operaciones, *inc* y *get*

$$
\begin{aligned}
Contador \stackrel{\text{def}}{=} [\; new = \varsigma(z)[\; & v = \varsigma(s)z.v(s),\\
& inc = \varsigma(s)z.inc(s),\\
& get = \varsigma(s)z.get(s)],\\
v = \lambda(s)0,&\\
inc = \lambda(s)s.v := s.v + 1,&\\
get = \lambda(s)s.v &\qquad\qquad\qquad\qquad]
\end{aligned}
$$

Representando Herencia

- Sea la clase

$$\mathsf{c} \stackrel{\mathsf{def}}{=} \begin{array}{l} [\ new = \varsigma(z)[l_i = \varsigma(s)z.l_i(s)^{i\in 1..n}], \\ \ l_i \quad\ = \lambda(s)b_i{}^{i\in 1..n}] \end{array}$$

- Se desea definir c' como subclase de c que agrega los pre-métodos $\lambda(s)b_k{}^{k\in n+1..n+m}$

$$\mathsf{c}' \stackrel{\mathsf{def}}{=} \begin{array}{l} [\ new = \varsigma(z)[l_i = \varsigma(s)z.l_i(s)^{i\in 1..n+m}], \\ \ l_j \quad\ = \mathsf{c}.l_j{}^{j\in 1..n} \\ \ l_k \quad\ = \lambda(s)b_k{}^{k\in n+1..n+m} \qquad\qquad] \end{array}$$

Representando Subclases

Contador Reseteable (símil Ejercicio 12)

- Definir una subclase de $Contador$ que provea un método que permita resetear el valor inicial del contador.
- Modificar la solución de manera tal que el valor inicial de las instancias sea 1.
- Cómo modificaría su solución para hacer que el método *get* se comporte como el de la superclase, pero que además contabilice las operaciones de *get* realizadas.

Simulando clases y subclases en JS

Contador y contador Reseteable

Mostrar cómo implementaría en JS a las clases Contador y su subclase Contador Reseteable.