①

a) ContadorGente $\stackrel{def}{=}$ [new = $\varsigma(z)$ [cantidad = $\varsigma(i)_z$.cantidad(i),
entrar = $\varsigma(i)_z$.entrar(i),
salir = $\varsigma(i)_z$.salir(i)],
cantidad = $\lambda(i)$ 0,
entrar = $\lambda(i)$ i.cantidad $:=$ i.cantidad + 1,
salir = $\lambda(i)$ i.cantidad $:=$ i.cantidad - 1
]

b) c $\stackrel{def}{=}$ ContadorGente.new

(*) Nota al final

surge del reemplazo de "z" en new

(OBJ) ContadorGente → ContadorGente [cantidad = $\varsigma(i)$ ContadorGente.cantidad(i),
entrar = $\varsigma(i)$ ContadorGente.entrar(i),
salir = $\varsigma(i)$ ContadorGente.salir(i)]
→ "mismo objeto" (OBJ)

(SEL)

c → [cantidad = $\varsigma(i)$ ContadorGente.cantidad(i),
entrar = $\varsigma(i)$ ContadorGente.entrar(i),
salir = $\varsigma(i)$ ContadorGente.salir(i)]

(*)1

ContadorGente.entrar(c) → [cantidad = c.cant + 1,
entrar = $\varsigma(i)$.ContGente.entrar(i),
salir = $\varsigma(i)$.ContGente.salir(i)
]

(SEL)

c.entrar → [cantidad = c.cant + 1,
entrar = $\varsigma(i)$ ContGente.entrar(i),
salir = $\varsigma(i)$.ContGente.salir(i)]

$C.entrar \longrightarrow \ldots\ C.cant+1 \longrightarrow L$ ⑫ ✓

$(C.entrar).cantidad \longrightarrow 1$ ✓ (SEL)

① (OBJ)

$ContadorGente \longrightarrow ContadorGente \quad \lambda(i)\, i.cant \leftarrow i.cant+1 \longrightarrow$

$\lambda(i)\, i.cant \leftarrow i.cant+1$ (OBJ)

(SEL)

$ContadorGente.entrar \longrightarrow \lambda(i)\, i.cant \leftarrow i.cant+1$

(OBJ)

$C \longrightarrow C$

$c.cant \leftarrow c.cant+1$ (UPD)

$\longrightarrow [cant = c.cant+1,$
$entrar = \lambda(i) ContGente.entrar(i),$
$salir = \lambda(i).ContGente.salir(i),$
$]$

$ContadorGente.entrar(c) \longrightarrow [cant = c.cant+1$ (APP)
$entrar = \lambda(i)\ ContGente.entrar(i),$
$salir = \lambda(i).\ ContGente.salir(i)$
$]$

② (OBJ) (OBJ)

$ContGente \longrightarrow ContGente \quad \lambda(i)\, 0 \longrightarrow \lambda(i)\, 0$

(SEL) (OBJ)

$ContGente.cantidad \longrightarrow \lambda(i)\, 0 \qquad 0 \longrightarrow 0$ (APP)

Hay que definir esta regla.

$C \longrightarrow C \qquad ContGente.cantidad(c) \longrightarrow 0$

$c.cant \longrightarrow 0$ (ADD) (Pa aporeja doda) ✓

$c.cant+1 \longrightarrow 1$

c)

ContadorGenteMax $\stackrel{def}{=}$ [new = $\varsigma(z)$ [max = $\varsigma(i)z.max(i)$,
cant = $\varsigma(i)z.cant(i)$,
entrar = $\varsigma(i)z.entrar(i)$,
salir = $\varsigma(i)z.salir(i)$
],
cant = ContadorGente.cant,
salir = ContadorGente.salir,
max = $\lambda(i)$ 0,
entrar = $\lambda(i)$ (i.cante ← i.cant + 1).
max ← MÁX(i.max, i.cant + 1)
]

Supongo que existe la función MAX(a,b) que hace lo que se espera. Se podría hacer con los objetos booleanos de la práctica.

(*) Supongo dada la "extensión" de funciones λ vista en la teórica, con sus reglas de semántica.

(2)

a) % camino (+R, -C)

- camino((N, [I]), [N]).
- camino ((N, RR), [N|NN]) :-
  member(Hijo, RR), camino (Hijo, NN).

b) % CaminoDeMayorValor (+R, -C)

- CaminoDeMayorValor (R,C) :- camino(R,C), not(hayMayor(C,R)).

% hayMayor (+C, +R)

- hayMayor (C, R) :- camino(R,C2), sumlist(C,V), sumlist(C2,V2), V2 > V.

c) Sí, C podría venir instanciada. Si fuera vacía, no unificaría con nada y fallaría. Si tiene la forma [N|NN], [redacted] unificará solamente in la raíz del nosetrel > "N"; de ser así, o bien se muestra una solución (caso base), o se instancia un nosetrel hijo para luego evaluar "camino" con el hijo y la cola de la lista, ambos instanciados.

③

a) • member(x, [x|-]).
$\forall x \forall L$ member(x, [x|L]).

• member(x, [-|xs]) :- member(x, xs).
$\forall x \forall L \forall y$ member(x, L) $\supset$ member(x, [y|L]).

• sacar(x, [x|xs], xs).
$\forall x \forall L$ sacar(x, [x|L], L).

• sacar(x, [y|ys], [y|xs]) :- sacar(x, ys, xs).
$\forall x \forall y \forall ys \forall xs$ sacar(x, ys, xs) $\supset$ sacar(x, [y|ys], [y|xs]).

b) Primero paso todo a sus FNC:

① { member(x, [x|xs]) }
② { ¬member(x, xs), member(x, [y|xs]) }
③ { sacar(x, [x|xs], xs) }
④ { ¬sacar(x, [y|ys], [y|xs]), ¬sacar(x, ys, xs) }

Ahora obtengo la negación de lo que quiero probar:
• $\neg \exists x \exists xs \exists L$ (sacar(x, xs, L) $\wedge$ member(x, L))
$\equiv \forall x \forall xs \forall L$ ¬sacar(x, xs, L) $\vee$ ¬member(x, L)
⑤ { ¬sacar(x, xs, L), ¬member(x, L) } ← GOAL

⑤ y ① : $L \leftarrow [x|xs]$ ✓

⑥ $\{\neg sacar(x, xs, [x|xs])\}$ ✓

⑥ y ④ : $xs_4 \leftarrow [x|xs]$ $ys_4 \leftarrow xs$ ✗

⑦ $\{\neg sacar(x, [y|ys_4], [y|[x|xs]])\}$

⑦ y ③ : FALTA LA SUSTITUCIÓN! ✗

⑧ : □ ✓

Esto significa que ⑤ (el goal) es insatisfacible, luego hemos lo que queríamos ✓.

Quod erat demonstrandum.

c) Sí. ✓

- Todas cláusulas de Horn ✓
- Un único goal inicial ✓
- Se comenzó por el goal ✓
- Fue lineal (5 y 1, 6 y 4, 7 y 3) ✓
- Resolución binaria. ✓