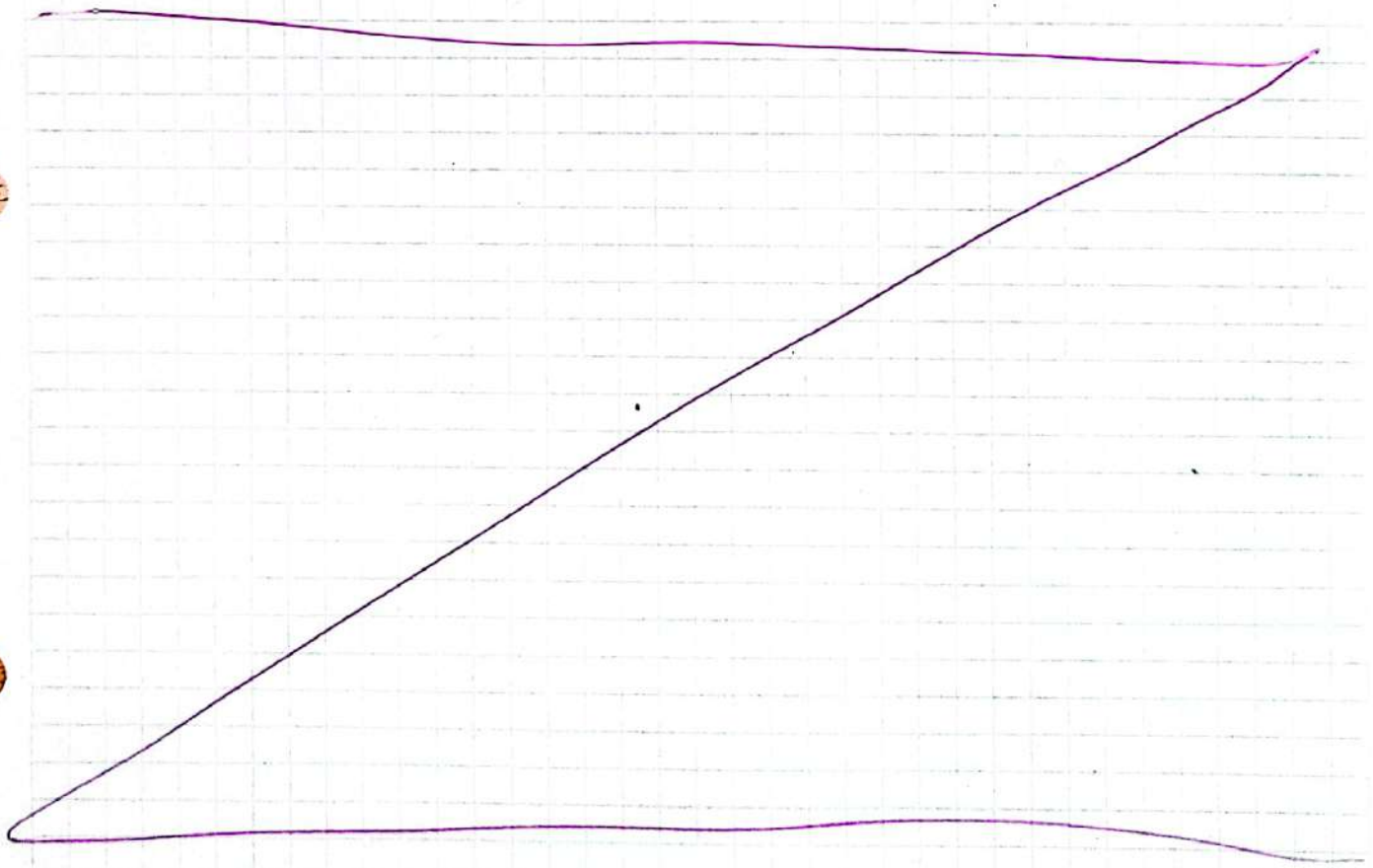
tieneValor : dicc(α, nat) × nat → bool

tieneValorAux: dicc(α, nat) d × conj(α) as × nat → bool
{as ⊆ claves(d)}

∀ d: dicc(α, nat), ∀ as: conj(α), ∀ n: nat

tieneValor (d, n) ≡ tieneValorAux (d, claves(d), n) ✓

tieneValorAux (d, as, n) ≡
  IF vacío? (as) then
    false
  else
    obtener (dameUno(as), d) =obs n
    v tieneValorAux (d, sinUno(as), n)
  Fi ✓

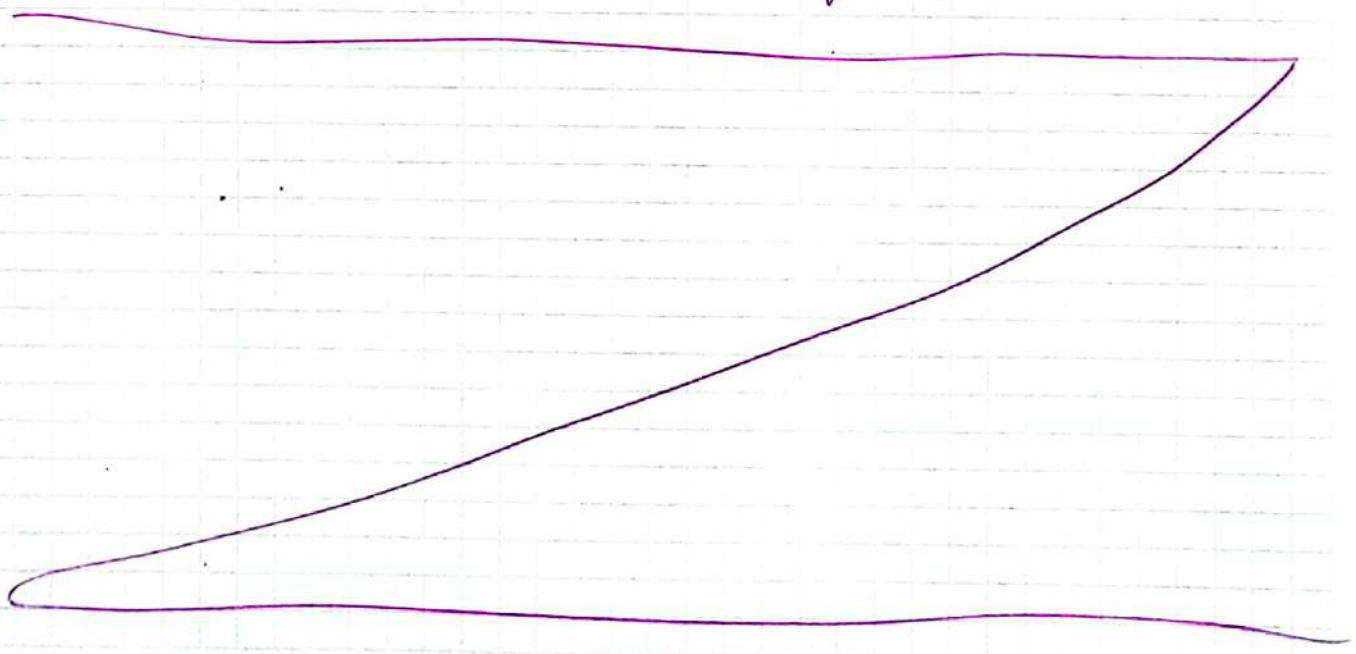


diasRestantesAlquiler (inaugurarVC (cs, ps), cz) ≡ vacío ✓

diasRestantesAlquiler (alquilar (v, c, p), cz) ≡
  if c =obs cz then (?)
    definir (p, pelis(v), diasRestantesAlquiler (v, c))
  else  "pelis(v)" es un dicc, debería haber sido "obtener(p, pelis(v))"
    diasRestantesAlquiler (v, cz)
  fi

diasRestantesAlquiler (devolver(v, c, p), cz) ≡
  if c =obs cz then
    borrar (p, diasRestantesAlquiler (v, c))
  else
    diasRestantesAlquiler (v, cz)
  fi ✓

diasRestantesAlquiler (pasarDia (v), cz) ≡
  restarUno (diasRestantesAlquiler (v, cz)) ✓



clientes (inaugurarVC (cs, ps)) ≡ cs ✓

clientes (alquilar (v, c, p)) ≡ clientes (v) ✓

clientes (devolver (v, c, p)) ≡ clientes (v) ✓

clientes (pasarDia (v)) ≡ desaFiliarClientes (clientes (v), v) ✓

desaFiliarClientes : conj(cliente) cs × VC v → VC

{ cs ⊆ clientes (v) }

desaFiliarClientes (cs, v) ≡

if vacío? (cs) then

∅

Debería ser 1, pues con 0 ya debería haber sido sacada

else

if tieneValor (diasRestantesAlquiler(v, dameUno(cs)), 0) then

// se le venció el plazo de al menos una peli

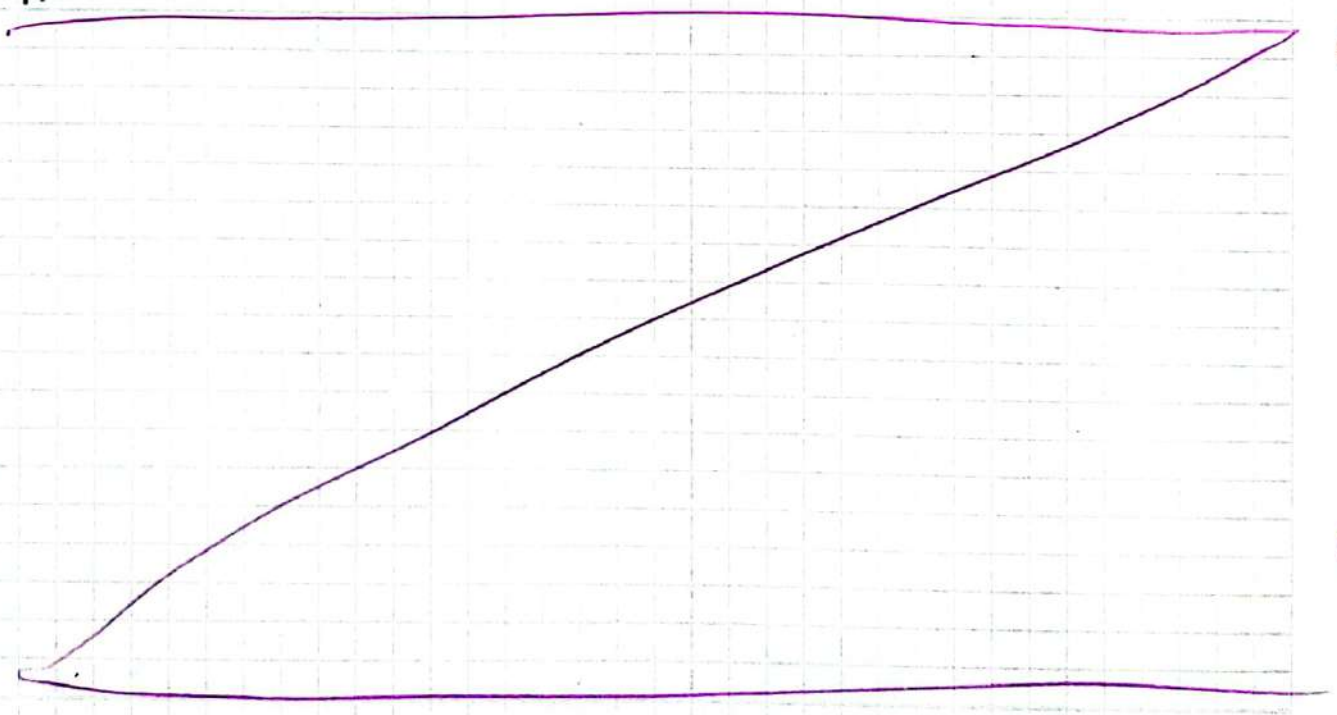
desaFiliarClientes (sinUno(cs), v)

else

Ag (dameUno(cs), desaFiliarClientes (sinUno(cs), v))

fi

Fi ✓



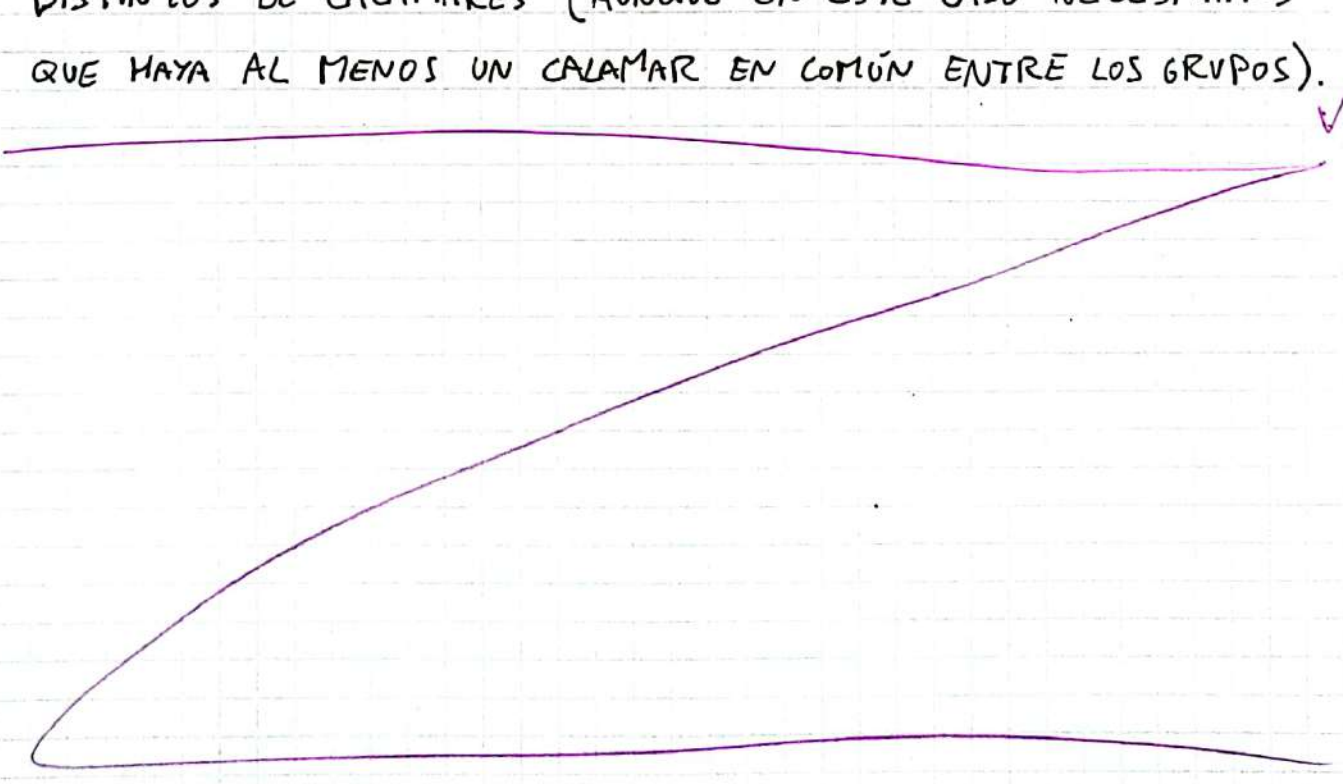
AED 2 3

EJERCICIO 3 29/4/22

PARA DIFERENCIAR DOS JUEGOS SOLO NECESITAMOS SABER QUIÉNES ESTÁN AÚN EN EL JUEGO Y QUIÉN SIGUE A QUIÉN (LOS SEGUIDORES DE CADA CALAMAR). ✓

LA CANTIDAD DE SEGUIDORES DE CADA CALAMAR SE PUEDE CALCULAR A PARTIR DE SUS SEGUIDORES, POR LO TANTO NO NECESITAMOS ESTE DATO (COMO OBSERVADOR) PARA DIFERENCIAR DOS JUEGOS. ✓

A SU VEZ, COMO A HBOBO NO LE INTERESA SABER EL HISTORIAL DE LUCHAS, NO NECESITAMOS LLEVAR REGISTRO DE QUIÉN LUCHÓ CON QUIÉN. ESTO SIGNIFICA QUE EN NUESTRO MODELO, DOS JUEGOS PUEDEN SER (OBSERVACIONALMENTE) IGUALES A PESAR DE HABERSE GENERADO DE FORMAS DISTINTAS. ✓ ES DECIR, CON DISTINTAS LUCHAS, DISTINTA SECUENCIA DE QUIÉN FUE SIGUIENDO A QUIÉN, E INCLUSO HABERSE INICIADO EL JUEGO CON GRUPOS DISTINTOS DE CALAMARES (AUNQUE EN ESTE CASO NECESITAMOS QUE HAYA AL MENOS UN CALAMAR EN COMÚN ENTRE LOS GRUPOS). ✓



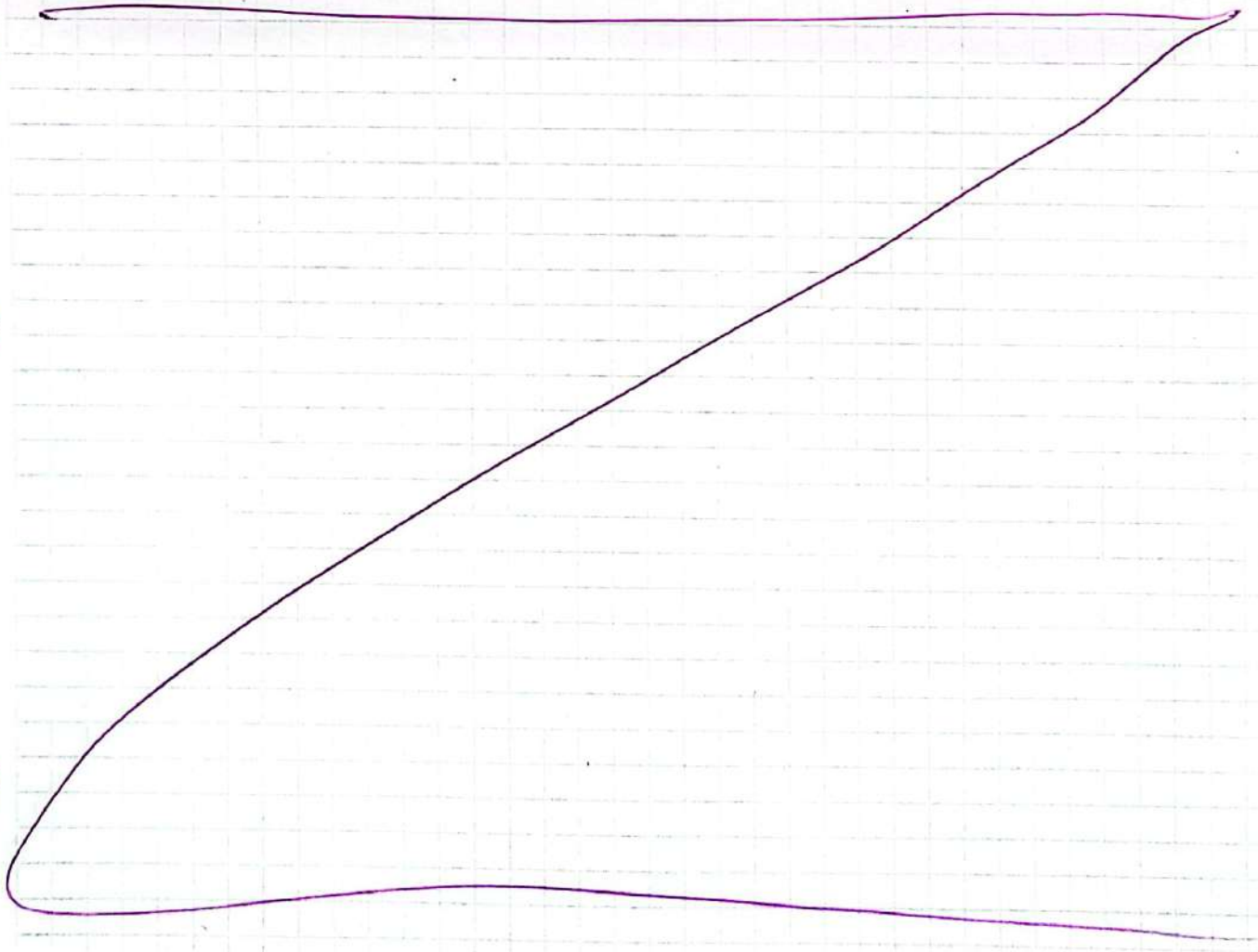
OBSERVADORES BÁSICOS

calamares : juego → conj(calamar) ✓

seguidores : calamar c × juego j → conj(calamar) ✓
{ c ∈ calamares(j) }

IGUALDAD OBSERVACIONAL

(∀ j1, j2 : juego)( j1 =obs j2 <=> (
calamares(j1) =obs calamares(j2)
∧L (∀ c: calamar)( c ∈ calamares(j1) ⇒L (
seguidores(c, j1) =obs seguidores(c, j2)
)
))



GENERADORES

comenzar : conj(calamar) cs → juego  { #(cs) > 0 } ✓

seguir : calamar c × calamar p × juego j → juego

{ c ≠obs p ∧ {c, p} ⊆ calamares(j) } ✓ ∧ c ∉ seguidores(p, j)

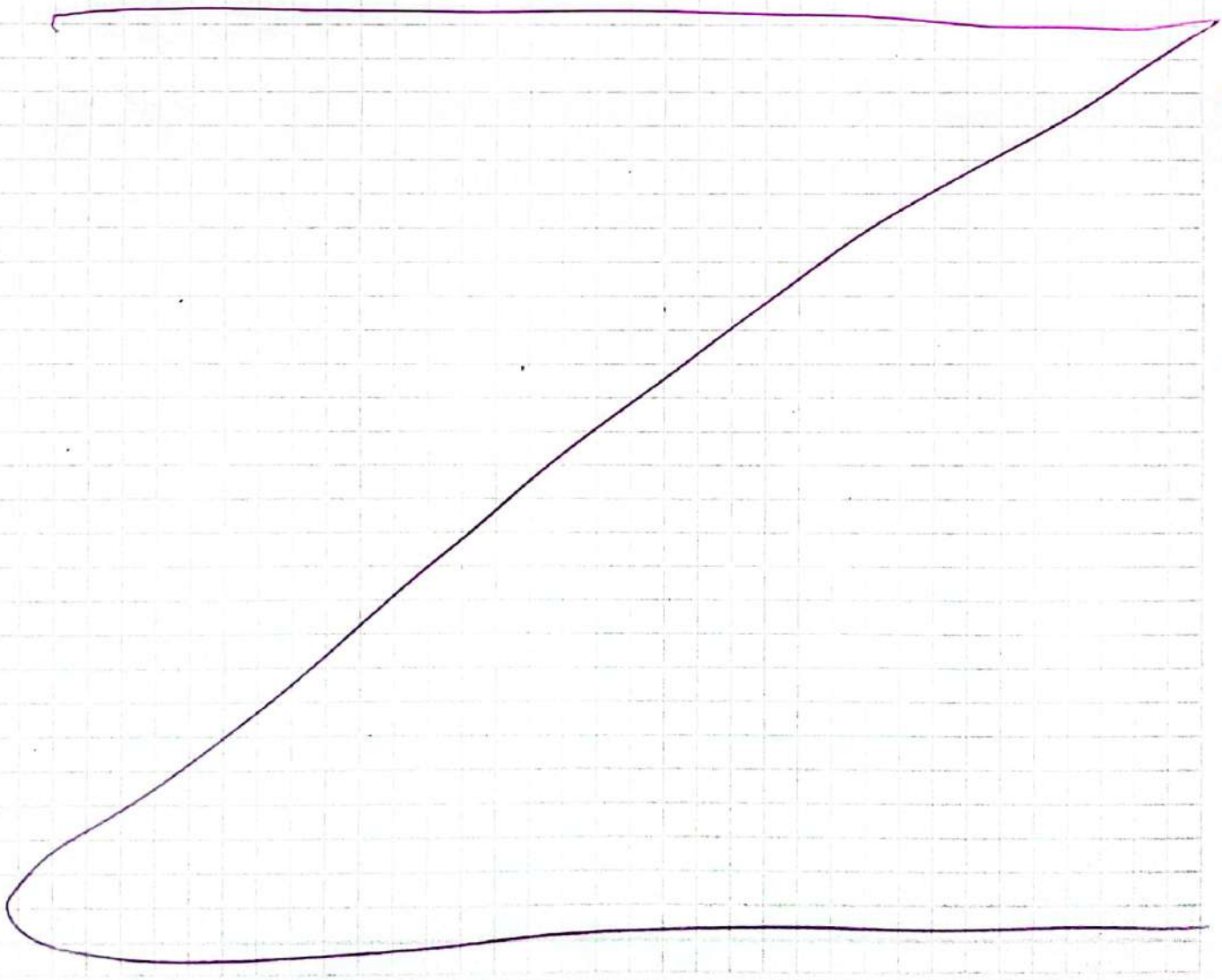
avergonzar: calamar c × calamar p × juego j → juego

{ c ≠obs p ∧ {c, p} ⊆ calamares(j) } ✓ ✓

OTRAS OPERACIONES

ganadores : juego → conj(calamar)

// No consideramos a ganadores como un observador básico
// porque se puede axiomatizar a partir de los seguidores de
// cada calamar. ✓ ✓



AXIOMAS

∀J: juego, ∀c,c2,P: calamar, ∀cs: conj(calamar)

seguidores(c, comenzar(cs)) ≡ ∅ ✓

seguidores(c, seguir(c2, P, J)) ≡
  IF c =obs P then
    Ag(c2, seguidores(c,J)) ✓
  else
    // No sabemos si c2 es seguidor de c, pero sí sabemos que
    // a partir de ahora sigue a P, así que no puede ser
    // seguidor de ningún otre calamar.
    seguidores(c,J) − {c2} ✓
  Fi

Está bien. xq el operador − no modifica el cjto si (c) no está. Sin embargo, es mejor tener cuidado en otras axiomatizaciones de otros TADs. Es mejor chequear si c2 ∈ segs(c) y recién ahí sacarlo.

seguidores(c, avergonzar(c2, P, J)) ≡
  IF c =obs c2 then
    // Como c2 avergonzó a P, ahora todos los seguidores de P
    // pasan a ser seguidores de c2 (además de los que ya tenía)
    seguidores(c,J) ∪ seguidores(P,J)
    // Si bien P ya no está más en juego, en la instancia J
    // previo al avergonzamiento sí lo está. Por eso podemos
    // preguntar por los seguidores de P en J. Exacto, había que hacerlo.
  else
    seguidores(c,J)
  Fi

Si c2 ∈ segs(P,J) ⇒ hay que sacar a c2 de la unión, xq c2 no puede seguirse a sí mismo.

Además si P ∈ segs(c2,J) ⇒ hay que sacar a P de la unión, xq si no va a haber un seguidor que quedó fuera del juego; lo cual está malo.

calamares (comenzar (cs)) ≡ cs ✓

calamares (seguir (c, p, j)) ≡ calamares (j) ✓

calamares (avergonzar (c, p, j)) ≡ calamares (j) − {p} ✓

ganadores (j) ≡ ganadoresAux (calamares (j), j) ✓

ganadoresAux : conj (calamar) cs × juego j → conj (calamar)
{cs ⊆ calamares (j)}

ganadoresAux (cs, j) ≡ No era necesario hacerlo, pero se valora positivamente haberlo hecho.

```
if vacio?(cs) then
  ∅
else
  if #(seguidores(dameUno(cs), j)) =obs maxSeguidores(calamares(j), j) then
    Ag(dameUno(cs), ganadoresAux(sinUno(cs), j))
  else
    ganadoresAux(sinUno(cs), j)
  fi
fi
```

✓

maxSeguidores : conj (calamar) cs × juego j → nat {cs ⊆ calamares(j)}

maxSeguidores (cs, j) ≡ if vacio?(cs) then 0 else

```
if #(seguidores(dameUno(cs), j)) > maxSeguidores(sinUno(cs), j) then
  #(seguidores(dameUno(cs), j))
else
  maxSeguidores(sinUno(cs), j)
fi
fi
```

✓