Ejercicio 1

Para cada uno de los siguientes protocolos, defina el tipo sesión correspondiente.

a)

El cliente envía un polinomio lineal al servidor, que responde con su raíz. Se puede asumir que existe el tipo `poly` de los polinomios lineales.

```
Client = !poly.?float.end
Server = ?poly.!float.end
```

b)

Modifique la definición anterior de manera que el cliente envíe los coeficientes del polinomio en lugar del polinomio en sí.

```
Client = !float.!float.?float.end
Server = ?float.?float.!float.end
```

c)

Modifique la definición del protocolo anterior para el caso en el que el cliente envía un polinomio cuadrático. Tenga en cuenta que, en este caso, el servidor puede responder con `0`, `1` o `2` raíces.

```
Client = !float.!float.!float.&[
    zero: end,
    one: ?float.end,
    two: ?float.?float.end
]

Server = ?float.?float.?float.⊕[
    zero: end,
    one: !float.end,
    two: !float.!float.end
]
```

d)

Redefina el protocolo de modo que el cliente no envíe un polinomio directamente, sino que inicie una sesión a través de la cual el servidor recibirá los coeficientes del polinomio cuadrático.

```
DelegatedClient = ⊕[start: ?Client, stop: end]
DelegatedServer = &[start: !Client, stop: end]
```

Los tipos `Client` y `Server` son los del inciso c).

Ejercicio 2

Dar el LTS (Labelled Transition System) correspondiente a cada uno de los siguientes tipos sesión.

a)

```
t1 = !int.!int.?int.end
{—} → !int.?int.end
```

```
{-} → ?int.end
{-} → end
```

b)

```
t2 = !(?int.?int.end).?int.end
{-} → ?int.end
{-} → end
```

c)

```
t3 = ⊕[Pr: !int.?bool.end, Co: !int.!int.?bool.end]
{Pr} → !int.?bool.end
{-}  → ?bool.end
{-}  → end
```

d)

```
t4 = ?mode.&[read: ⊕[eof: end, val: !string.end], close: end]
{-}    → &[read: ⊕[eof: end, val: !string.end], close: end]
{read} → ⊕[eof: end, val: !string.end]
{val}  → !string.end
{-}    → end
```

Ejercicio 3

Considere las siguientes definiciones:

```
Tester1 = ?int.!bool.end
Tester2 = ?int.!int.end
Pserver = x+?(y:int).x+!true.0
Qserver = x+?(y:int).x+!false.0
Pclient = x-!1.x-?(z:bool).0
Qclient = x-!2.x-?(z:bool).0
```

Dar el LTS correspondiente a cada uno de los siguientes procesos.

a)

```
Pserver = x+?(y:int).x+!true.0
```

La expresión está en forma normal, no se puede reducir.

b)

```
(νx:Tester1) Pserver
```

La expresión está en forma normal, no se puede reducir.

c)

```
Pserver | Pclient
≡ Pclient | Pserver
≡ x−!1.x−?(z:bool).0 | x+?(y:int).x+!true.0
{x,−} → x−?(z:bool).0 | (x+!true.0){1/y}
≡ x−?(z:bool).0 | x+!true.0
≡ x+!true.0 | x−?(z:bool).0
{x,−} → 0 | 0{true/z}
≡ 0 | 0
≡ 0
```

d)

```
(νx:Tester1)(Pserver | Pclient)
≡ (νx:Tester1)(Pclient | Pserver)
≡ (νx:?int.!bool.end)(x−!1.x−?(z:bool).0 | x+?(y:int).x+!true.0)
{τ,−} → (νx:!bool.end)(x−?(z:bool).0 | (x+!true.0){1/y})
≡ (νx:!bool.end)(x+!true.0 | x−?(z:bool).0)
{τ,−} → (νx:end)(0 | 0{true/z})
≡ (νx:end)(0 | 0)
≡ (νx:end)(0)
≡ 0
```

e)

```
(νx:Tester2)(Pserver | Pclient)
≡ (νx:Tester2)(Pclient | Pserver)
≡ (νx:?int.!int.end)(x−!1.x−?(z:bool).0 | x+?(y:int).x+!true.0)
{τ,−} → (νx:!int.end)(x−?(z:bool).0 | (x+!true.0){1/y})
≡ (νx:!int.end)(x+!true.0 | x−?(z:bool).0)

# La reducción no respeta el session type
{τ,−} → (νx:end)(0 | 0{true/z})
≡ (νx:end)(0 | 0)
≡ (νx:end)(0)
≡ 0
```

f)

```
(νy:Tester1)(Pserver | Pclient)
≡ (νy:Tester1)(Pclient | Pserver)
≡ (νy:Tester1)(x−!1.x−?(z:bool).0 | x+?(y:int).x+!true.0)
{x,−} → (νy:Tester1)(x−?(z:bool).0 | (x+!true.0){1/y})
≡ (νy:Tester1)(x−?(z:bool).0 | x+!true.0)
≡ (νy:Tester1)(x+!true.0 | x−?(z:bool).0)
{x,−} → (νy:Tester1)(0 | 0{true/z})
≡ (νy:Tester1)(0 | 0)
≡ (νy:Tester1)(0)
```

g)

```
(νy:Tester2)(Pserver | Pclient)
```

Análogo a f) porque `Pserver` y `Pclient` se comunican sobre el canal `x` ≠ `y`.

h)

```
(νx:Tester1)Pserver | Pclient
≡ (νx:?int.!bool.end)(x+?(y:int).x+!true.0) | x−!1.x−?(z:bool).0
```

Se traba porque $x^p \in \text{fn}(\text{Pclient})$, entonces no se puede aplicar la equivalencia estructural.

```
(νx:Tester1)Pserver | Pclient ≠ (νx:Tester1)(Pserver | Pclient)
```

i)

El problema patológico con esta expresión es que los session types modelan una comunicación binaria, es decir entre únicamente 2 procesos (delegar canales es otra cosa). `PServer` puede comunicarse tanto con `Pclient` como con `Qclient` indistintamente. Incluso podrían comunicarse `Pclient` con `Qclient` después de que uno de ellos mande su `int`.

Hay muchas reducciones posibles, este es un caso donde `PServer` se comunica con `Pclient` como en el inciso c), y `Qclient` no reduce.

```
Pserver | Pclient | Qclient
≡ 0 | Qclient   # Por inciso c
≡ Qclient
```

j)

Si bien ahora podemos reducir al proceso `0`, esto depende exclusivamente de qué procesos elegimos comunicar. Si asociamos un `PServer` con `Pclient` y el otro `PServer` con `Qclient` entonces llegamos a `0`.

Pero hay otras posibles reducciones que quedan trabadas, por ejemplo:

```
Pserver | Pserver | Pclient | Qclient
≡ Pclient | Pserver | Pserver | Qclient
≡ (x−!1.x−?(z:bool).0) | (x+?(y:int).x+!true.0) | Pserver | Qclient
{x,−} → (x−?(z:bool).0) | (x+!true.0){1/y} | Pserver | Qclient
≡ (x−?(z:bool).0) | (x+!true.0) | Pserver | Qclient
≡ (x−?(z:bool).0) | (x+!true.0) | (x+?(y:int).x+!true.0) | Qclient
{x,0} → (x−?(z:bool).0) | 0 | (x+!true.0){true/y} | Qclient
≡ (x−?(z:bool).0) | (x+!true.0) | Qclient
≡ (x+!true.0) | (x−?(z:bool).0) | Qclient
{x,−} → 0 | 0{true/z} | Qclient
≡ 0 | 0 | Qclient
≡ 0 | Qclient
≡ Qclient
```

k)

```
(νx:Tester1)(Pserver | Qserver | Pclient | Qclient)
```

Ahora la comunicación entre estos procesos sobre el canal `x` se realiza de forma "privada" debido a la restricción `νx`. Pero seguimos en el caso patológico, los session types básicos solo modelan comunicaciones entre 2 procesos.

El canal `x` tiene asociado el siguiente protocolo, o mejor dicho session type: `Tester1 = ?int.!bool.end`. En español esto dice que por el canal `x` se recibe un `int`, luego de envía un `bool` y termina la comunicación. Cuando algún par de procesos intenten comunicarse sobre el canal `x` van a ir "consumiendo" el session type (como si fuese una lógica lineal).

Por lo tanto, por más que comuniquemos a los procesos de la forma "conveniente" para que todos terminen, por ejemplo primero reducimos `Pserver` y `Pclient`, cuando intentemos reducir `Qserver` y `Qclient` no vamos a poder porque el canal `x` ya tiene tipo `end`. No se puede aplicar la regla `R–NewS`.

l)

Este caso funciona bien porque hay 2 canales, y cada par de procesos tiene su propio canal para comunicarse. Y ya vimos en el inciso d) que estas expresiones reducen a `0`.

```
(νx:Tester1)(Pserver | Pclient) | (νx:Tester1)(Qserver | Qclient)
*↣ 0 | 0
≡ 0
```

Ejercicio 4

Dar el LTS correspondiente a los siguientes procesos.

a)

```
(νx:?int.end)(x–!1.0 | (νy:!int.end)(x+?(z:int).y+!z.0 | y–?(w:int).0))

# Ponemos el proceso x–!1.0 bajo la restricción de y.
≡ (νx:?int.end)((νy:!int.end)(x–!1.0 | x+?(z:int).y+!z.0 | y–?(w:int).0))

# Comunicamos sobre el canal x.
{τ,–} → (νx:end)((νy:!int.end)(0 | (y+!z.0){1/z} | y–?(w:int).0))
≡ (νx:end)((νy:!int.end)(0 | y+!1.0 | y–?(w:int).0))
≡ (νx:end)((νy:!int.end)(y+!1.0 | y–?(w:int).0))

# Comunicamos sobre el canal y.
{τ,–} → (νx:end)((νy:end)(0 | 0{1/w}))
≡ (νx:end)((νy:end)(0 | 0))
≡ (νx:end)((νy:end)(0))
≡ (νx:end)(0)
≡ 0
```

b)

```
(νx:?int.!int.?int.!int.end)(
    x+?(z:int).x+!(z+1).0 |
    x+?(z:int).x+!(z+1).0 |
    x–!1.x–?(z:int).Q1 |
    x–!2.x–?(z:int).Q2
)

≡ (νx:?int.!int.?int.!int.end)(
    x–!1.x–?(z:int).Q1 |
    x+?(z:int).x+!(z+1).0 |
    x–!2.x–?(z:int).Q2 |
    x+?(z:int).x+!(z+1).0
)

# Mandamos el 1.
{τ,–} → (νx:!int.?int.!int.end)(
```

```
    x−?(z:int).Q1 |
    x+!(z+1).0{1/z} |
    x−!2.x−?(z:int).Q2 |
    x+?(z:int).x+!(z+1).0
)
≡ (νx:!int.?int.!int.end)(
    x+!2.0 |
    x−?(z:int).Q1 |
    x−!2.x−?(z:int).Q2 |
    x+?(z:int).x+!(z+1).0
)

# Mandamos el 2.
{τ,−} → (νx:?int.!int.end)(
    0 |
    Q1{2/z} |
    x−!2.x−?(z:int).Q2 |
    x+?(z:int).x+!(z+1).0
)
≡ (νx:?int.!int.end)(
    Q1{2/z} |
    x−!2.x−?(z:int).Q2 |
    x+?(z:int).x+!(z+1).0
)

# Análogamente repetimos la misma comunicación con los otros 2 procesos.
{τ,−} → (νx:!int.end)(
    Q1{2/z} |
    x−?(z:int).Q2 |
    x+!(z+1).0{2/z}
)
≡ (νx:!int.end)(
    Q1{2/z} |
    x+!2.0 |
    x−?(z:int).Q2
)
{τ,−} → (νx:end)(
    Q1{2/z} |
    0 |
    Q2{2/z}
)

# Aún no podemos quitar la restricción sobre x.
# No sabemos si Q1/Q2 son 0.
≡ (νx:end)(Q1{2/z} | Q2{2/z})
```