Taller 4 Modelado de sistemas concurrentes

1.

```
ENCUENTRO = (hola -> conversar -> adios -> STOP).

TRABAJO = (llego -> trabajo -> parto -> TRABAJO).

MOVIMIENTO = (adelante -> GIRO),
GIRO = (izquierda -> STOP | derecha -> MOVIMIENTO).

MONEDA = (tira -> ceca -> MONEDA | tira -> cara -> MONEDA).

CUATROTICKS = (tick -> tick -> tick -> tick -> STOP).

DOBLE = (in[x:1..3] -> D[x]),
D[x:1..3] = (out[x*2] -> DOBLE).
```

4.

Con suposicion de que al pedir nivel hay tiempo para responder antes de que llegue un nuevo pulso de agua.

```
SENSOR1 = (agua[x:0..10] -> NIVEL[x]),
NIVEL[x:0..10] = (when(x<=2) nivel -> BAJO[x]
                 |when(x>=8) nivel -> ALTO[x]
                 |when(2<x & x<8) nivel -> MEDIO[x]
                 |agua[y:0..10] -> NIVEL[y]
                 ),
BAJO[x:0..2] = (bajo -> NIVEL[x]),
MEDIO[x:3..7] = (medio -> NIVEL[x]),
ALTO[x:8..10] = (alto -> NIVEL[x]).
```

Sin ninguna supocicion:

```
SENSOR2 = (agua[x:0..10] -> NIVEL[x]),
NIVEL[x:0..10] = (when(x<=2) nivel -> BAJO[x]
                 |when(x>=8) nivel -> ALTO[x]
                 |when(2<x & x<8) nivel -> MEDIO[x]
                 |agua[y:0..10] -> NIVEL[y]
                 ),
BAJO[x:0..10] = (bajo -> NIVEL[x] | agua[y:0..10] -> BAJO[y]),
```

```
MEDIO[x:0..10] = (medio -> NIVEL[x] | agua[y:0..10] -> MEDIO[y]),
ALTO[x:0..10] = (alto -> NIVEL[x] | agua[y:0..10] -> ALTO[y]).
```

5.

Modelo del comportamiento del agua:

```
AGUA = AGUA[5],
AGUA[x:0..10] = (when(x<10) agua[x+1] -> AGUA[x+1]
                |when(x>0) agua[x-1] -> AGUA[x-1]).
```

Modelo del agua compuesto con la segunda versión del sensor:

```
||SYS = (SENSOR2 || AGUA).
```

7.

```
range R = 0..7
VARIABLE = VARIABLE[3],
VARIABLE[i:R] = (
    read[i] -> VARIABLE[i] |
    write[j:R] -> VARIABLE[j] |
    write[8]-> OVERFLOW |
    write[-1] -> UNDERFLOW),
OVERFLOW = (overflow -> STOP),
UNDERFLOW = (underflow -> STOP).

SUMA1 = (read[i:R] -> write[i+1] -> SUMA1) + {write[0], write[-1]}.
RESTA1 = (read[i:R] -> write[i-1] -> RESTA1) + {write[7], write[8]}.


||SYS = ({suma,resta}::VARIABLE || suma:SUMA1 || resta:RESTA1).
```

Traza donde la variable da overflow:

```
suma.read.3
suma.write.4
suma.read.4
suma.write.5
suma.read.5
suma.write.6
```

```
  suma.read.6
  suma.write.7
  suma.read.7
  suma.write.8
  suma.overflow
 STOP
```

Traza donde la variable da underflow:

```
  resta.read.3
  resta.write.2
  resta.read.2
  resta.write.1
  resta.read.1
  resta.write.0
  resta.read.0
  resta.write.-1
  resta.underflow
 STOP
```

Traza infinita donde no ocurre ni underflow ni overflow:

```
  suma.read.3
  suma.write.4
  resta.read.4
  resta.write.3
  suma.read.3
  suma.write.4
  resta.read.4
  resta.write.3
  suma.read.3
  suma.write.4
  resta.read.4
  resta.write.3
  ...
```

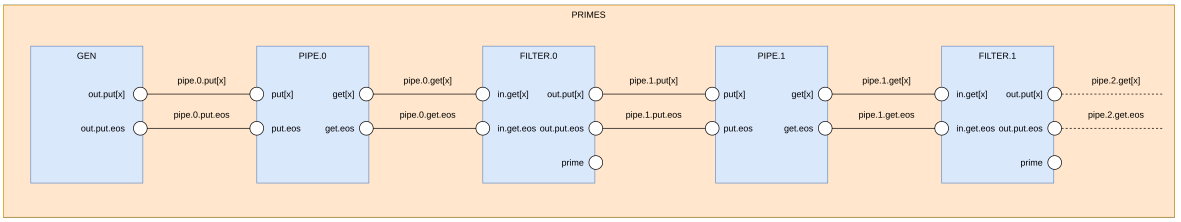
Traza donde suma y resta se interfieren:

```
  suma.read.3
  resta.read.3
  suma.write.4
  resta.write.2
  suma.read.2
```

...

10.

El diagrama de arquitectura del proceso PRIMES es:



Pipe con doble storage:

```
PIPE=(put[x:S]-> STORE[x]),
STORE[x:S] = (get[x] -> PIPE
           |put[y:S] -> get[x] -> STORE[y]).
```

Con el pipe de doble storage podemos ver que el proceso PRIMES soporta mayor concurrencia ya que en el caso de que algún filtro se tarde en procesar los mensajes, los buffers permiten que los filtros predecesores sigan procesando más mensajes. Por ejemplo, usando buffers de tamaño 2, en la siguiente traza podemos ver que se llegó a generar hasta el número 11 antes de que se procese el primo 3 por el filtro 1:

```
pipe.0.put.2
pipe.0.get.2
pipe.0.put.3
pipe.0.put.4
filter.0.prime.2
pipe.0.get.3
pipe.1.put.3
pipe.1.get.3
pipe.0.get.4
pipe.0.put.5
pipe.0.put.6
pipe.0.get.5
pipe.0.put.7
pipe.1.put.5
pipe.0.get.6
pipe.0.put.8
pipe.0.get.7
```

```
pipe.0.put.9
pipe.1.put.7
pipe.0.get.8
pipe.0.put.10
pipe.0.get.9
pipe.0.put.11
filter.1.prime.3
```

En cambio, usando buffers de tamaño 1, solo se llega a generar hasta el número 8:

```
pipe.0.put.2
pipe.0.get.2
pipe.0.put.3
filter.0.prime.2
pipe.0.get.3
pipe.0.put.4
pipe.1.put.3
pipe.0.get.4
pipe.0.put.5
pipe.1.get.3
pipe.0.get.5
pipe.0.put.6
pipe.1.put.5
pipe.0.get.6
pipe.0.put.7
pipe.0.get.7
pipe.0.put.8
filter.1.prime.3
```