1 Introducción

Esta práctica tiene como objetivo, de una parte la familiarización del formato `RIF`, y de otra seguir con el entorno Lustre implementando dos sistemas reactivos sencillos: una especificación de un semáforo con paso de peatones y una especificación más elaborada de dos semáforos diferentes.

2 Más cosas sobre el lenguaje programación síncrono Lustre

2.1 El formato Reactive Input (RIF)

RIF (Reactive Input Format) es un formato de archivo diseñado para la comunicación y manejo de datos entre diferentes herramientas dentro del ecosistema de simulación y verificación de programas escritos en lenguajes reactivos como Lustre. Este formato es utilizado principalmente para representar entradas y salidas de simulaciones, de manera que estas puedan ser interpretadas por las herramientas correspondientes.

El formato RIF sirve como intermediario para leer entradas y escribir salidas. Esto asegura compatibilidad y permite analizar el comportamiento de programas bajo diferentes condiciones de entrada. Es decir, permite realizar baterías de pruebas que sirvan para probar el sistema. Durante la simulación de programas escritos en Lustre, se pueden generar archivos `.rif` que contienen información detallada de las entradas proporcionadas y las respuestas generadas por el sistema.

Un archivo RIF típicamente incluye:

- Entradas: Variables reactivas que se alimentan al sistema durante la simulación. Estas representan eventos o datos que el sistema debe procesar.
- Salidas: Respuestas generadas por el sistema reactivo con base en las entradas proporcionadas.
- Tiempos: Información temporal que indica en qué momento ocurren los eventos o cambios en las variables.

Este formato está diseñado para ser estandarizado y legible por las herramientas que trabajan dentro del ecosistema reactivo. De esta forma, los archivos `.rif` pueden ser visualizados utilizando `sim2chro`.

En resumen, el formato `RIF` es esencial para la simulación, análisis y depuración de programas reactivos, ya que actúa como un puente y permite realizar baterías de pruebas y asegurar la consistencia en la representación de los datos en el sistema.

2.2 El formato

Un fichero `.rif` tiene el siguiente formato:

```
# This file was generated by lv6 version v6.111.2.
#  lv6 edge.lus -n EDGE -exec -o edge.rif
# on alumno the 23/01/2025 at 11:42:36
#inputs "X":bool
#outputs "Y":bool
#step 1
true #outs f
#step 2
false #outs t
#step 3
true #outs t
#step 4
true #outs f
```

Básicamente contiene:

- comentarios: es todo lo que aparece entre un '#' y un final de línea.
- pragmas: son tipos particulares de comentarios ('#inputs', '#outputs', '#outs', '#step', etc.)
- datos: que pueden ser `int`, `bool`, `real`.

Vamos a verlo de forma detallada.

2.2.1 Comentarios

Los comentarios de una sola línea se introducen con # y terminan con un salto de línea. Los comentarios multilínea se introducen con # y se terminan con los dos caracteres #@.

2.2.2 Pragmas

Los pragmas son tipos especiales de comentarios, que pueden (o no) ser tenidos en cuenta por las herramientas que leen datos `RIF`. Los pragmas de una línea tienen la forma #pragma_ident, y los pragmas de varias líneas la forma #pragma_ident ... #@.

Los pragmas más comunes utilizados son (siguiendo formato de expresión regular):

- para declarar la lista de nombres y tipos de variables de entrada:

```
#@inputs (<nombre var> : <tipo var>)+ #@ o
#inputs (<nombre var> : <tipo var>)+
```

- para declarar la lista de nombres y tipos de variables de salida:

```
#@outputs (<var nombre> : <var tipo>)+ #@ o
#outputs (<nombre var> : <tipo var>)+
```

- para declarar la lista de nombres y tipos de variables locales:

```
#@locals (<var nombre> : <var tipo>)+ #@ o
#locs
```

- para indicar que los siguientes datos corresponden a variables de salida:

```
#outs
```

- para indicar que se está iniciando un nuevo paso, y que los siguientes datos corresponden a variables de entrada:

```
#step <int>
```

Nótese que estos pragmas son necesarios para un visor de ficheros `RIF` como `sim2chro` funcione correctamente.

2.2.3 Datos

Un fichero `RIF` es una secuencia de valores de datos separados por espacios, nuevas líneas, tabulaciones, retornos de carro y saltos de línea. Un tipo de dato puede ser un entero, un real o un booleano (t, T o 1 significa verdadero; f, F o 0 significa falso).

2.3 Creación del fichero RIF

Para crear un fichero `.rif` tenemos dos alternativas. La primera es crear el fichero manualmente copiando la salida proporcionada por la ejecución del fichero ejecutable del sistema reactivo. Retomando el ejemplo de la primera práctica sería:

```
$lv6 edge.lus -node EDGE -exec
```

Esto genera en tiempo real una salida en el terminal que posteriormente se copia manualmente a un fichero `.rif`.

La segunda opción es mediante:

```
$rm -f edge.rif
$lv6 edge.lus -node EDGE -exec -o edge.rif
```

En este caso se crea el fichero `edge.rif` automáticamente. El usuario solo tendrá que introducir las entradas del sistema reactivo y en el fichero `edge.rif` se guardará tanto las entradas proporcionadas como las salidas generadas por el sistema. Es importante destacar que para evitar incongurencias en el formato, se elimine previamente todo fichero `.rif` previamente creado.

2.4 Visualización gráfica del fichero RIF

Finalmente, también se puede visualizar gráficamente tanto las entradas como las salidas de un fichero `.rif`. Esto es mediante la herramienta `sim2chro`.

```
$sim2chro -ecran -in edge.rif > /dev/null
```

El resultado será que se abre el contenido del fichero, en este caso, `edge.rif`, tal y como se ve en la Figura 1.

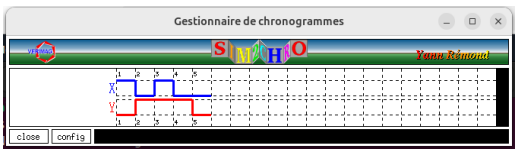

Figure 1: Salida `sim2chro` a partir del fichero `.rif`

3 Ejercicios de Lustre

3.1 Especificación de un SEMAFORO Y PASO DE PEATONES

Simulación de un sencillo sistema de semáforo con un carril (bidireccional) para coches y un paso de peatones, tal como se ve en la Figura 2.

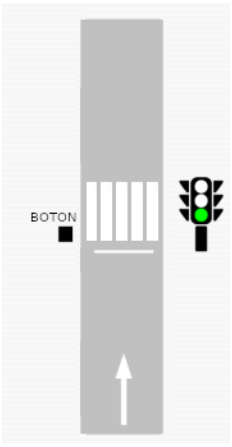

Figure 2: Especificación de un semáforo para coches y peatones

Se trata de un sistema que regula tanto el tráfico de vehículos como el paso de peatones. Funciona en base a un ciclo dividido en cinco fases que controla tanto las señales de tráfico para vehículos (rojo, amarillo, verde) como para peatones (caminar y no caminar). A continuación, se describe el comportamiento en detalle.

El nodo `TrafficLight` tiene por entrada `BOTON` que representa un botón que los peatones presionan para cruzar la calle. El sistema está diseñado para garantizar una transición fluida y segura entre las diferentes señales del semáforo. Con respecto a las salidas, tendrá las siguientes:

- `RED`: Indica si el semáforo para vehículos está en rojo.
- `YELLOW`: Indica si el semáforo para vehículos está en amarillo.
- `GREEN`: Indica si el semáforo para vehículos está en verde.
- `WALK`: Indica si los peatones pueden cruzar.
- `DONTWALK`: Indica si los peatones no pueden cruzar.

Además, el nodo `TrafficLight` posee las siguientes variables internas:

- `Fase`: Representa el estado actual del semáforo. Puede tomar valores de 0 a 4:
  - 0: Semáforo en verde (vehículos).
  - 1: Semáforo en amarillo (vehículos).
  - 2: Semáforo en rojo (transición para peatones).
  - 3: Semáforo en rojo y peatones pueden caminar.
  - 4: Semáforo en rojo, peatones deben prepararse para detenerse.
- `preFase`: Almacena el valor de la fase en el ciclo anterior para manejar la transición a la siguiente fase.

La lógica del nodo es la siguiente. Empecemos explicando la transición por fases:

- Si el botón de peatones (`BOTON`) es presionado y el semáforo estaba en la fase 0 (`GREEN`), el sistema avanza a la fase 1 (`YELLOW`).
- Si el semáforo está en una fase intermedia (1 a 4), avanza a la siguiente fase.
- Si el semáforo alcanza la fase final (4), regresa a la fase inicial (0).

En cuanto a las salidas, estas se establecen en base a:

- `GREEN`: El semáforo está en verde únicamente en la fase 0.
- `YELLOW`: El semáforo está en amarillo únicamente en la fase 1.
- `RED`: El semáforo está en rojo para vehículos en las fases 2, 3 y 4.
- `WALK`: Los peatones pueden cruzar únicamente en las fases 3 y 4.
- `DONTWALK`: Complemento de la señal Walk, los peatones no pueden cruzar en las fases 0, 1 y 2.

A continuación, se detalla el funcionamiento de las fases:

- Fase 0: El semáforo para vehículos está en verde. Los peatones no pueden cruzar. En caso de pulsar `BOTON`, se avanza a la Fase 1.
- Fase 1: El semáforo para vehículos está en amarillo. Los peatones no pueden cruzar. Esta fase es una transición hacia la fase 2, en la que el semáforo se pone en rojo y dura un ciclo.
- Fase 2: El semáforo para vehículos está en rojo. Los peatones no pueden cruzar aún, pero se prepara el cruce para peatones.
- Fase 3: El semáforo para vehículos está en rojo. Los peatones ya pueden cruzar.
- Fase 4: El semáforo para vehículos sigue en rojo. Los peatones aún pueden cruzar, pero deben prepararse para detenerse.

Una vez completada la fase 4, el semáforo regresa a la fase inicial (verde para vehículos y no caminar para peatones).

Notas:

- La implementación del nodo `TrafficLight` dispone de cinco salidas: tres para las luces (`GREEN`, `YELLOW` y `RED`) y dos para los peatones `WALK` y `DONTWALK`; y una única entrada `BOTON`.
- El botón solo tiene efecto si el semáforo está en la fase de verde para evitar interrupciones bruscas en otras fases.

3.1.1 Ejercicio 1

A continuación, se proporciona la siguiente implementación del nodo `TrafficLight`. Vamos a ver cuales son los problemas que encontramos en este código.

```
node TrafficLight(BOTON:bool) returns (GREEN, YELLOW, RED, WALK,
                                       DONTWALK: bool);
  var Fase, preFase: int;
let
  preFase = 0 -> pre(Fase);
  Fase = if BOTON then 1
         else if 0 < preFase and preFase < 4 then preFase + 1
              else 0;

  GREEN = Fase = 0;
  YELLOW = Fase = 1;
  RED = Fase > 1;

  WALK = Fase > 2 and Fase < 4;
  DONTWALK = not WALK;
tel
```

Para ello, lo primero es crearlo, luego compilarlo al tiempo que permite su ejecución, y guardar las entradas y salidas de la ejecución en un `.rif`.

```
$nano TrafficLight.lus
$lv6 TrafficLight.lus -n TrafficLight -exec -o traffic.rif
```

A partir de ahí, introducir en el ciclo 1 y 7 las entradas a `true` mientras que las demás deberían estar a `false`. Debería de obtenerse algo tal que así, o lo que es lo mismo, la Figura 4:

```
# This file was generated by lv6 version v6.111.2.
#  lv6 TrafficLight.lus -n TrafficLight -exec -o traffic.rif
# on alumno the 23/01/2025 at 16:59:27
#inputs "BOTON":bool
#outputs "GREEN":bool "YELLOW":bool "RED":bool "WALK":bool "DONTWALK":bool
#step 1
t #outs f t f f t
#step 2
f #outs f f t f t
#step 3
f #outs f f t t f
#step 4
f #outs f f t f t
#step 5
f #outs t f f f t
#step 6
f #outs t f f f t
#step 7
t #outs f t f f t
#step 8
f #outs f f t f t
#step 9
f #outs f f t t f
#step 10
f #outs f f t f t
```

Node: TrafficLight

Time	0	1	2	3	4	5	6	7	8
BOTON (bool)	☑	☐	☐	☐	☐	☐	☑	☐	☐
GREEN (bool)	☐	☐	☐	☐	☑	☑	☐	☐	☐
YELLOW (bool)	☑	☐	☐	☐	☐	☐	☑	☐	☐
RED (bool)	☐	☑	☑	☑	☐	☐	☐	☑	☑
WALK (bool)	☐	☐	☑	☐	☐	☐	☐	☐	☑
DONTWALK (bool)	☑	☑	☐	☑	☑	☑	☑	☑	☐
Fase (int)	1	2	3	4	0	0	1	2	3
preFase (int)	0	1	2	3	4	0	0	1	2

Figure 3: TrafficLight ciclo a ciclo

En esa figura se observa aparentemente un correcto funcionamiento.

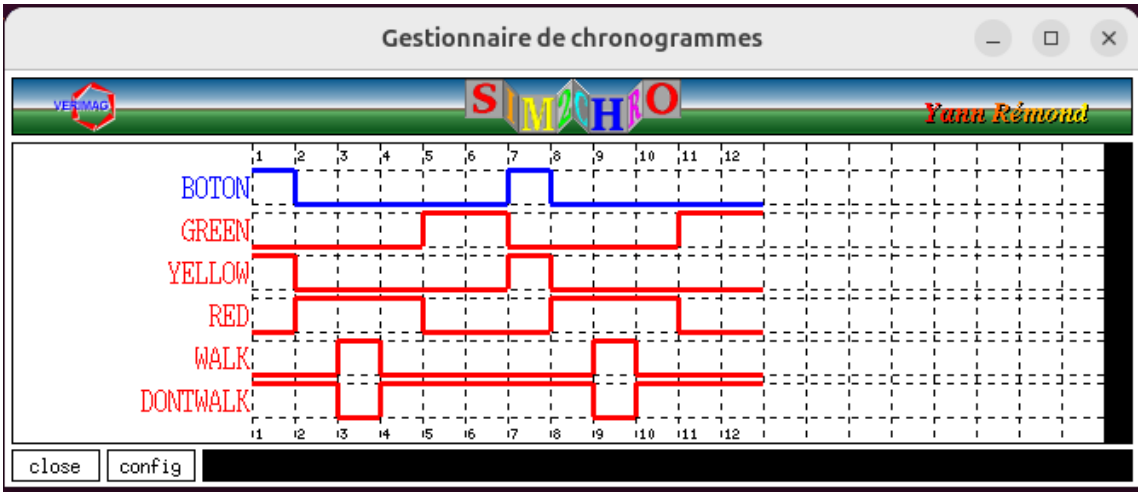

Figure 4: TrafficLight ciclo a ciclo

ACTIVIDAD: Siguiendo con el ejemplo:

- ¿qué ocurre si, además de pulsar el botón en el ciclo 1 y 7, también se pulsa el botón en el ciclo 3?

  Mostrar la salida obtenida mediante sim2chro.

- Explicar qué está ocurriendo en el código de Lustre.

3.1.2 Ejercicio 2

Se propone una modificación sobre el anterior ejercicio. Realizar los mismo pasos.

```
node TrafficLight2(BOTON:bool) returns (GREEN, YELLOW, RED, WALK,
                                        DONTWALK: bool);
var Fase, preFase: int;
let
  preFase = 0 -> pre(Fase);
  Fase    = if BOTON and (false -> pre(DONTWALK)) then 1
            else if 0 < preFase and preFase < 4 then preFase + 1
                 else 0;

  GREEN = Fase = 0;
  YELLOW = Fase = 1;
  RED = Fase > 1;

  WALK = Fase > 2 and Fase < 4;
  DONTWALK = not WALK;
tel
```

ACTIVIDAD: Siguiendo con el ejemplo:

- ¿qué ocurre ahora?

  Mostrar la salida obtenida mediante sim2chro.
- Explicar qué está ocurriendo en el código de Lustre.
- ¿Qué habría que cambiar en el código para que eso no sucediera?

  Mostrar la salida obtenida tras la modificación mediante sim2chro.

3.2 Especificación dos SEMAFOROS

Simulación de un semáforo sobre dos vías de un solo sentido: una principal y otra secundaria, tal y como se muestra en la Figura 5.

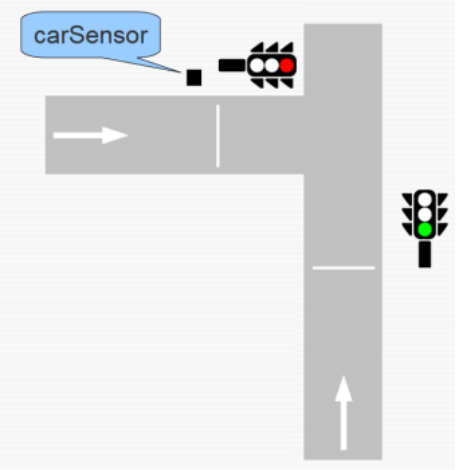

Figure 5: Especificación de un semáforo

Se trata de un sistema de dos semáforos que regula un cruce de dos calles (de sentido único). Cuando la vía principal está abierta, esta permanece en ese estado hasta que algún coche se encuentre a la espera sobre la vía secundaria. En otras palabras, por defecto, el semáforo 1 está en verde y el semáforo 2 en rojo. Cuando se detecta un coche en el semáforo 2 (entrada `carSensor`*), el sistema cambia el semáforo 1 a rojo, el semáforo 2 a verde. En ese momento, la vía secundaria queda abierta durante 4 segundos para volver a cerrarse y vuelve a la situación por defecto.*

Notas:

- Se sugiere la implementación de un nodo genérico `SEMAFORO` que disponga de seis salidas para las tres luces de cada semáforo (`VERDE`, `AMBAR` y `ROJO`).
- Suponer que el semáforo, al cambiar de verde a rojo, pasa un segundo en ámbar y que una vez que un semáforo se pone en rojo, el otro espera un segundo antes de ponerse en verde.

ACTIVIDAD: Realizar el código del ejercicio.

OPCIONAL: Intentar aplicar este problema al caso de tres semáforos.