# 1 Introducción

El objetivo de esta práctica es familiarizarse con el uso de datos complejos en Lustre, como arrays y estructuras de datos, y trabajar con iteradores para manipular y acceder a estos datos. Además, se abordará el uso de diversos relojes en el lenguaje.

# 2 Datos estructurados e iteradores

## 2.1 Definición de los tipos de datos

En esta primera parte, se introduce el uso de tipos de datos estructurados y su aplicación en arrays. Como primer paso, se definirán los datos que se emplearán en la práctica.

### 2.1.1 Definición de un tipo de dato estructurado simple

Se define un tipo de dato estructurado (`struct`) llamado `sensorTemp`, que representa un sensor de temperatura con dos campos: `id`, un identificador entero del sensor, y `temperatura`, un valor real que indica la temperatura medida. Además, se implementa una función `sensor`, que recibe como entrada un entero y un real, y devuelve una instancia de `sensorTemp` con los valores proporcionados.

```
type sensorTemp = struct{id: int; temperatura: real};
function sensor(i: int; temp: real) returns (out: sensorTemp);
let
  out.id=i;
  out.temperatura=temp;
tel;
```

### 2.1.2 Creación de un array de datos estructurados

Se define un array `array_sensores` que almacena sensores de temperatura con valores ficticios de identificador y temperatura. Primero, se declara el tipo `array_sensores` como un array de tres elementos de tipo `sensorTemp`. Luego, se implementa una función `inicializar` que devuelve un array de sensores inicializado con valores predeterminados.

```
type array_sensores=sensorTemp^3;
function inicializar() returns (array: array_sensores);
let
  array = [
    sensor(1, 25.0),
    sensor(2, 30.5),
    sensor(3, 28.3)
  ];
tel;
```

## 2.2 Iteradores y procesamiento de datos

A continuación, se presentan ejemplos de código en Lustre que ilustran el uso de iteradores para procesar arrays.

### 2.2.1 Iteración sobre un array y cálculo de la temperatura media

Se implementan dos funciones para calcular la temperatura media de los sensores. La primera usa una metodología convencional, sumando cada temperatura y dividiendo entre el número total de elementos.

```
function temperatura_media1(array: array_sensores) returns (media: real);
var
  suma: real;
let
  suma = array[0].temperatura + array[1].temperatura +
         array[2].temperatura;
  media = suma / 3.0;
tel;
```

La segunda función aprovecha los iteradores `map` y `red` para extraer y reducir las temperaturas de manera eficiente.

El iterador `red` permite reducir un array a un escalar. Sin embargo, al tratarse de un array de un tipo de dato estructurado, es necesario en primer lugar extraer a un array los valores de las temperaturas mediante `map` para luego poder aplicar `red`. Por este motivo, la función `extraer_temperatura` se utiliza dentro del `map` para extraer cada temperatura de los elementos del array.

```
function extraer_temperatura(sensor: sensorTemp) returns (temp: real);
let
  temp = sensor.temperatura;
tel;

function temperatura_media2(array: array_sensores) returns (media: real);
var
  suma: real;
  a: real^3;
let
  -- Extrae las temperaturas del array de sensores
  a = map<<extraer_temperatura,3>>(array);
  suma = red<<+,3>>(0.0, a);
  media = suma / 3.0;
tel;
```

Finalmente, se define el nodo `main` para verificar que ambas funciones devuelven el mismo resultado.

```
node main() returns (media1, media2:real);
var
        array: array_sensores;
let
        array=inicializar();
        media1=temperatura_media1(array);
        media2=temperatura_media2(array);
tel;
```

**ACTIVIDAD:**
Crear una función `temperatura_media3` modificando la función `temperatura_media2` para que en vez de usar el iterador `map` y `red`, usar únicamente `fillred`.

### 2.2.2 Filtrado de sensores con temperaturas mayores a 28 grados

A continuación, vamos a recorrer el array de sensores y devolver el valor acumulado de las temperaturas, así como el número de sensores cuya temperatura sea mayor a 28 grados.

```
function filtro_temperatura28(s: sensorTemp) returns (b:bool);
let
    b = s.temperatura > 28.0;
tel;

node acumular_temp_filtrados(acumTemp: real; s: sensorTemp)
     returns (acc_temp: real; y:sensorTemp);
let
    acc_temp = if filtro_temperatura28(s) then acumTemp + s.temperatura
               else acumTemp;
    y=s;
tel;

node acumular_elem_filtrados(acumElem: real; s: sensorTemp)
     returns (acc_elem: real);
let
    acc_elem = if filtro_temperatura28(s) then acumElem + 1.0
               else acumElem;
tel;

node filtrar_sensores(array: array_sensores)
     returns (media:real; a_filtrado:array_sensores);
var
    suma:real;
    num:real;
let
    (suma, a_filtrado) = fillred<<acumular_temp_filtrados, 3>>(0.0, array);
    num = red<<acumular_elem_filtrados, 3>>(0.0, a_filtrado);
    media = suma / num;
tel;
```

Probemos el código mediante el siguiente nodo `main2`.

```
node main2() returns (val:real; array2:array_sensores);
var
        array: array_sensores;
let
        array=inicializar();
        (val,array2)=filtrar_sensores(array);
tel;
```

### 2.2.3 Modificación de temperaturas en sensores con ID impar

Se implementa un nodo que recorre el array mediante un iterador y modifica las temperaturas de aquellos cuyo identificador sea impar, incrementando su temperatura en 2 grados.

```
node es_impar(s: sensorTemp) returns (s1:sensorTemp);
var
    impar:bool;
let
    impar = (s.id mod 2) <> 0;
    s1 = if impar then sensor(s.id,s.temperatura + 2.0) else s;
tel;

node modificar_sensores_impares(array: array_sensores)
     returns (a_modif:array_sensores);
let
    a_modif = map<<es_impar, 3>>(array);
tel;
```

```
node main3() returns (array3:array_sensores);
var
        array: array_sensores;
let
        array=inicializar();
        array3=modificar_sensores_impares(array);
tel;
```

# 3 Uso de relojes

En esta sección se abordará el uso de relojes en Lustre, explorando su aplicación en la sincronización de procesos y el tratamiento de eventos a diferentes velocidades.

Supongamos que queremos desarrollar un sistema para gestionar la activación de una alarma en función de la temperatura y el estado de dos relojes: uno rápido y otro lento. La alarma se encenderá cuando ambos relojes estén activos simultáneamente y la temperatura supere los 28 grados. A continuación, desglosaremos las partes clave que componen este sistema.

Cuando ambos relojes estén activos al mismo tiempo y la temperatura capturada sea superior a 28 grados, se encenderá la alarma. Para ello, la temperatura se sincroniza con el reloj lento antes de ser evaluada. La activación de la alarma se determina fusionando los valores de ambos relojes mediante el operador `merge`, de manera que la alarma solo se activará si el reloj rápido y el reloj lento están activos simultáneamente y la temperatura sincronizada supera el umbral de 28 grados.

## 3.1 Tipos de datos y declaración de variables

Definimos un tipo de datos llamado `tipo_temperatura`, que representa valores reales de temperatura dentro del sistema. También declaramos un tipo `act`, que es un enumerado con dos estados posibles: `ACTIVADO` y `NOACTIVADO`, los cuales indican si la alarma está encendida o apagada.

```
type tipo_temperatura = real;
type act=enum{ACTIVADO,NOACTIVADO};
```

## 3.2 Generación de relojes

El primer módulo del sistema es un nodo llamado `generar_relojes` que permite devolver dos señales de reloj `clk_rapido` y `clk_lento` a diferentes frecuencias. El reloj `clk_rapido` es un reloj que se activa

cada 2 ciclos. Esto se determina mediante la condición `(c mod 2 = 0)`. Por otro lado, `clk_lento` es un reloj que se activa cada 4 ciclos. Esto se controla con la condición `(c mod 4 = 0)`. Este nodo recibe como entrada un contador c, que indica el ciclo actual, y devuelve el estado de ambos relojes.

Estos relojes serán utilizados en los siguientes módulos para sincronizar el procesamiento de datos.

```
node generar_relojes(c: int) returns (clk_rapido: bool; clk_lento: bool);
let
    clk_rapido = false->(c mod 2 = 0);  -- Activo cada 2 ciclos
    clk_lento  = false->(c mod 4 = 0);  -- Activo cada 4 ciclos
tel;
```

## 3.3 Filtrado de temperatura

El nodo `filtrar_temperatura` tiene como objetivo filtrar la temperatura solo cuando `clk_rapido` está activo. Recibe como entrada la temperatura y el reloj rápido, y devuelve la temperatura filtrada únicamente cuando `clk_rapido` está activado.

```
node filtrar_temperatura(clk_rapido: bool; temp: tipo_temperatura)
     returns (temp_filtrada: real when clk_rapido);
let
    -- Filtramos la temperatura solo cuando clk_rapido esté activo
    temp_filtrada=temp when clk_rapido;
tel;
```

## 3.4 Activación de la alarma

Este nodo gestiona la activación de la alarma en función de los relojes y la temperatura. Sus entradas son `clk_rapido`, `clk_lento` y `temp` (sincronizada con `clk_rapido`).

Lo primero será sincronizar la temperatura `temp` con el reloj lento (`clk_lento`) utilizando `current(temp) when clk_lento`. Esto asegura que la temperatura se almacene de manera sincronizada con el reloj lento.

A continuación, `clock_lento` y `clock_rapido` son los flujos asociados a cada uno de los relojes que luego se fusionan usando el operador `merge`. Esto permite que los valores de los relojes se tomen en cuenta en un solo flujo de trabajo. En este sentido, `clock_lento` depende de `clk_lento` y también de si la temperatura sincronizada (`temp_sincronizada`) es mayor a 28 grados. De cumplirse ambos, entonces `clock_lento` será activado. Del mismo modo, `clock_rapido` simplemente refleja el valor de `clk_rapido`.

En cuanto a `clock`, consiste en un flujo que estará activado cuando se establece la conjunción de los dos flujos anteriores `clock_lento` y `clock_rapido` (la condición `cond`).

Finalmente, La alarma se activará (`ACTIVADO`) si la condición `clock` es verdadera, y se desactivará (`NOACTIVADO`) en caso contrario.

```
node activar_alarma(clk_rapido: bool; clk_lento: bool;
                    temp: tipo_temperatura when clk_rapido)
     returns (alarma: act;temp_sincronizada:tipo_temperatura when
                    clk_lento; clock:bool);
var
    clock_lento, clock_rapido, cond:bool;
let
```

```
    -- Sincronizamos la temperatura en base al reloj lento
    temp_sincronizada = current(temp) when clk_lento;

    -- La alarma se activa si ambos relojes están activos y la
    -- temperatura sincronizada es mayor a 28
    clock_lento = merge clk_lento (true->temp_sincronizada>28.0)
                                  (false->false);
    clock_rapido = merge clk_rapido (true->true) (false->false);
    cond=clock_rapido and clock_lento;
    clock= merge cond (true->true) (false->false);

    alarma=if clock then ACTIVADO else NOACTIVADO;
tel;
```

## 3.5 Nodo principal

El nodo `main` es el nodo principal del sistema, donde se orquestan los demás nodos y se procesan los resultados. Lo primero es disponer de un contador c que se inicializa en 0 y se incrementa en cada ciclo. Esto permite generar los relojes en función del tiempo de ejecución.

A continuación se tiene que invocar el nodo `generar_relojes(c)` para generar los dos relojes, `clk_rapido` y `clk_lento`. Una vez hecho, se pasa la temperatura `temp` al nodo `filtrar_temperatura`, que devuelve `temp_filtrada` solo cuando el reloj rápido (`clk_rapido`) está activo.

Luego se pasa la temperatura filtrada junto con los relojes generados al nodo `activar_alarma`. Este nodo devuelve la alarma y la temperatura sincronizada en el reloj lento. El nodo `main` devolverá el estado final de la alarma (`alarma_final`), que puede ser `ACTIVADO` o `NOACTIVADO` dependiendo de la condición de los relojes y la temperatura.

```
node main(temp: tipo_temperatura)
          returns (clk_lento, clk_rapido:bool;alarma_final: act;
                   temp_filtrada:tipo_temperatura when clk_rapido;
                   temp_sincronizada:tipo_temperatura when clk_lento;
                   clock:bool);
var
    c: int;  -- Contador global para generar relojes
    alarma: act;
let
    -- Inicializamos el contador
    c = 0 -> pre(c) + 1;

    -- Generamos los relojes
    (clk_rapido, clk_lento) = generar_relojes(c);
    -- Filtramos la temperatura con el reloj rápido
    temp_filtrada = filtrar_temperatura(clk_rapido, temp);

    -- Activamos la alarma con la temperatura filtrada y los relojes
    (alarma,temp_sincronizada,clock) = activar_alarma(clk_rapido,
                                       clk_lento, temp_filtrada);

    -- El resultado final es el estado de la alarma
    alarma_final = alarma;
tel;
```

## 3.6 Resumen del flujo

Los relojes `clk_rapido` y `clk_lento` controlan el flujo de tiempo del sistema. La temperatura se filtra solo cuando el reloj rápido (`clk_rapido`) está activo. La alarma se activa solo cuando ambas condiciones de reloj están activas y la temperatura es mayor a 28 grados, utilizando el reloj lento para la sincronización de la temperatura. Finalmente, el sistema devuelve el estado de la alarma (`ACTIVADO` o `NOACTIVADO`), basado en la condición temporal y de temperatura.

## 3.7 Posibles mejoras o extensiones

Este sistema se puede extender o modificar de varias maneras, por ejemplo:

- Agregar más condiciones para activar la alarma en función de otros parámetros (como la velocidad de cambio de la temperatura).
- Mejorar la precisión de los relojes, utilizando más factores temporales.
- Integrar más sensores para monitorizar otras variables ambientales junto con la temperatura.

Este tipo de sistema es ideal para aplicaciones en sistemas de control en tiempo real, como la supervisión de temperaturas en entornos críticos donde la acción inmediata (como activar una alarma) es necesaria.

**ACTIVIDAD:**
`Implementar dos de las mejoras propuestas.`