Parte 2: retardos y parámetros

En esta nueva sección veremos dos temas fundamentales en el diseño de procesadores de audio: el ***retardo simple*** con la primitiva `@`; y la ***realimentación*** (circuito en bucle), que requerirá el uso de la composición recursiva `A~B`, y nos permitirá conectar las salidas de A en las entradas de B y las salidas de B en las entradas de A.

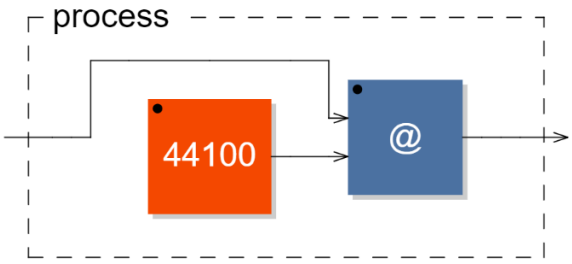
Ocasionalmente nos valdremos del uso de "comentarios", que son una forma de escribir texto que ***no será procesado, pero que sirve para dejar notas y aclaraciones*** de lo que hace nuestro código. Al final de una línea, o en líneas nuevas, podemos utilizar el símbolo `//` (dos barras diagonales) para explicar a Faust que lo que sigue será un comentario nuestro, y deberá ignorarlo en el procesamiento.

```
// esto es un comentario en una linea nueva!
process = _,_;   // esto es un comentario al final de una linea!
```

Contenidos | Parte 2

Ejemplo 1: Retardo monofónico de 1 segundo

Comencemos con un ejemplo muy simple, un retardo monofónico de 1 segundo. Para ello utilizaremos la primitiva de retardo simple `@`, la cual retardará nuestra señal en una cantidad de muestras especificada por nosotros. Con una cuenta muy sencilla encontramos que, si nuestra frecuencia de muestreo es de 44100 Hz (aunque podría ser cualquier otra), 1 segundo equivale a 44100 muestras, 0.5 segundos equivale a 22050 muestras, y así…

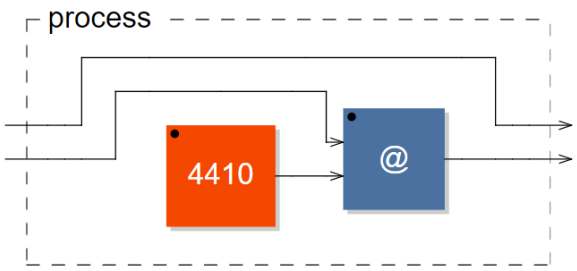



```
process = @(44100);

// tambien podria escribirse asi
// process = _, 44100 : @;
```

Como podemos notar, solo con retardar una señal no escucharemos una gran diferencia excepto porque tarda más en comenzar el sonido. Los retardos por si solos tienen muchas utilidades en el diseño de filtros, igualamiento de latencias y otros. Sin embargo, si buscamos lograr (y escuchar) un efecto estilo eco, necesitaremos *sumar este retardo a la señal original*. Eso iremos haciendo en los próximos ejemplos.

Ejemplo 2: Retardo de 0.1 segundos en el canal derecho

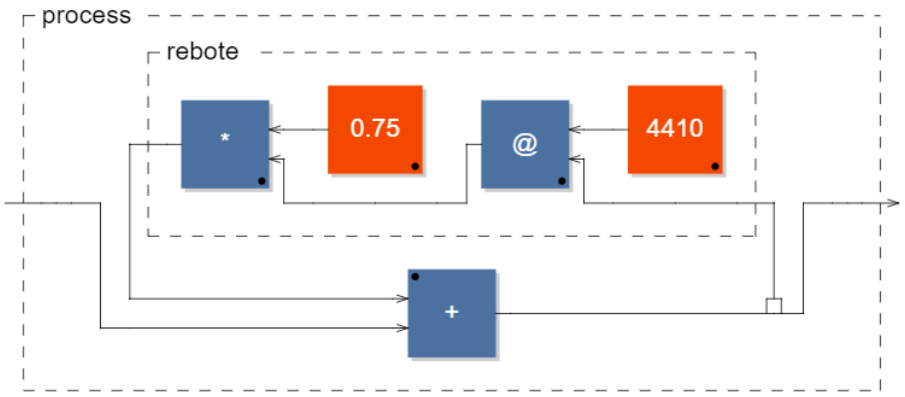
Para escuchar mejor el retardo, pongámoslo solo en el canal derecho y dejemos el canal izquierdo sin cambios. Tal como mencionamos antes, si necesitamos realizar un retardo de 0.1 segundos, la cantidad de muestras sería: 44100 * 0.1 = 4410.




```
process = _, @(4410);
```

Ejemplo 3: Rebote del sonido en una pared

Al combinar un retardo y una atenuación podemos simular el rebote del sonido en una pared. Claro está que una pared real no solo atenuará el sonido, sino que además filtrará el mismo de formas complejas y muy específicas, pero servirá para nuestro ejemplo. Crearemos una nueva definición llamada “rebote” (podría tener cualquier nombre) solo para hacer más sencilla la escritura.

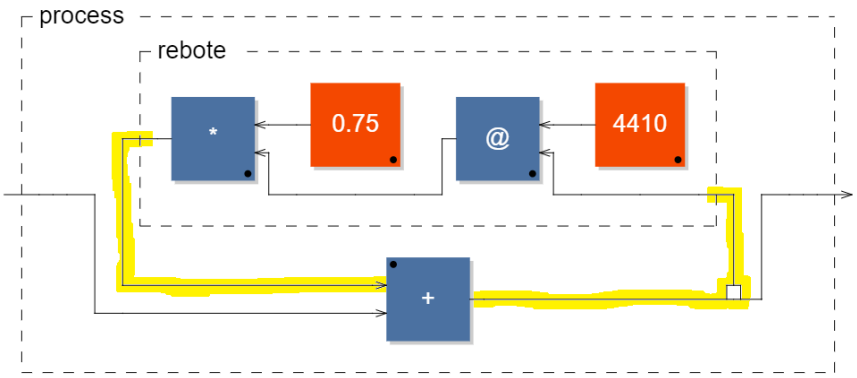



```
rebote = @(4410) : *(0.75);
process = rebote;
```

Como vimos anteriormente, el diagrama de bloques nos mostrará la definición “rebote” incluida dentro de “process”.

Ejemplo 4: Eco monofónico simple

Para simular un eco, lo que podemos hacer es crear un ***ciclo de realimentación***, donde una copia con retardo de una señal se sume a la original. Es lo que típicamente llamamos delay/eco con feedback/realimentación: no escucharemos una sola repetición del sonido original como en el ejemplo anterior, sino una serie de repeticiones que se irán atenuando progresivamente. Para lograr esto usaremos la composición recursiva `~`, que nos permitirá precisamente redirigir la salida de un bloque hacia su propia entrada, con algún tipo de proceso en el medio. La sintaxis será `A~B`, y la entenderemos así: la salida de A se conectará a la entrada de B, y la salida de B se conectará a la entrada de A.

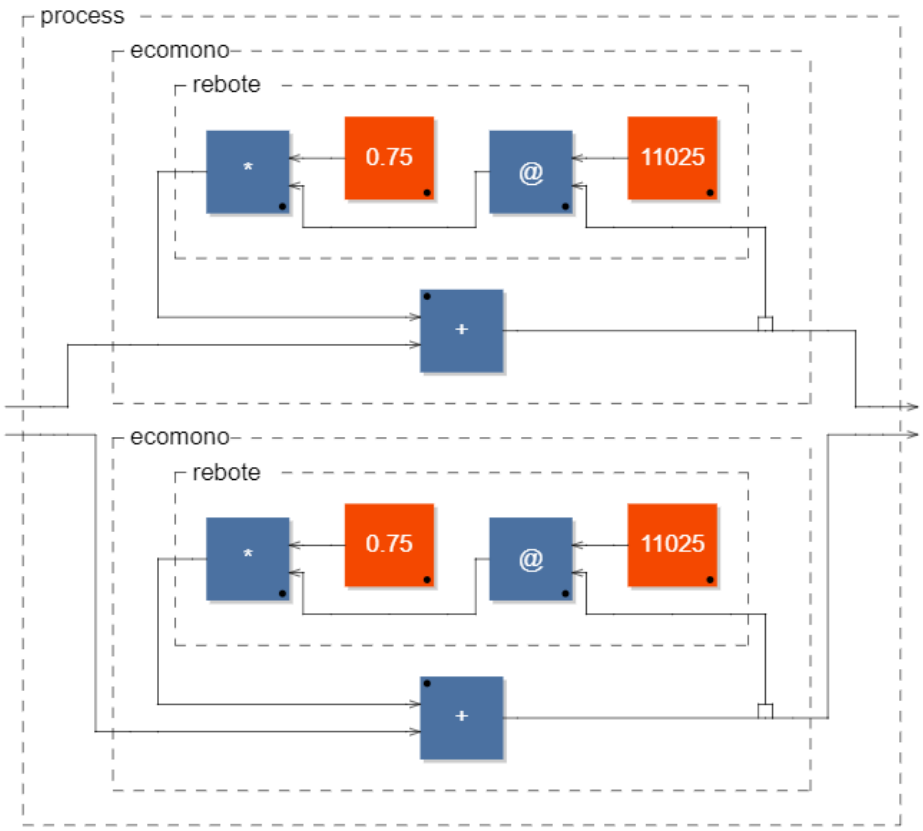



```
rebote = @(4410) : *(0.75);
ecomono = +~rebote;
process = ecomono;
```

Observemos en el diagrama lo que ocurre con el direccionamiento de la señal alrededor del bloque `+`. Podemos ver cómo la señal proveniente del canal izquierdo pasa por el bloque de suma, el cual es ***alimentado*** nuevamente por el resultado de la propia suma, creando este ciclo de ***realimentación***.

Ejemplo 5: Eco estereofónico

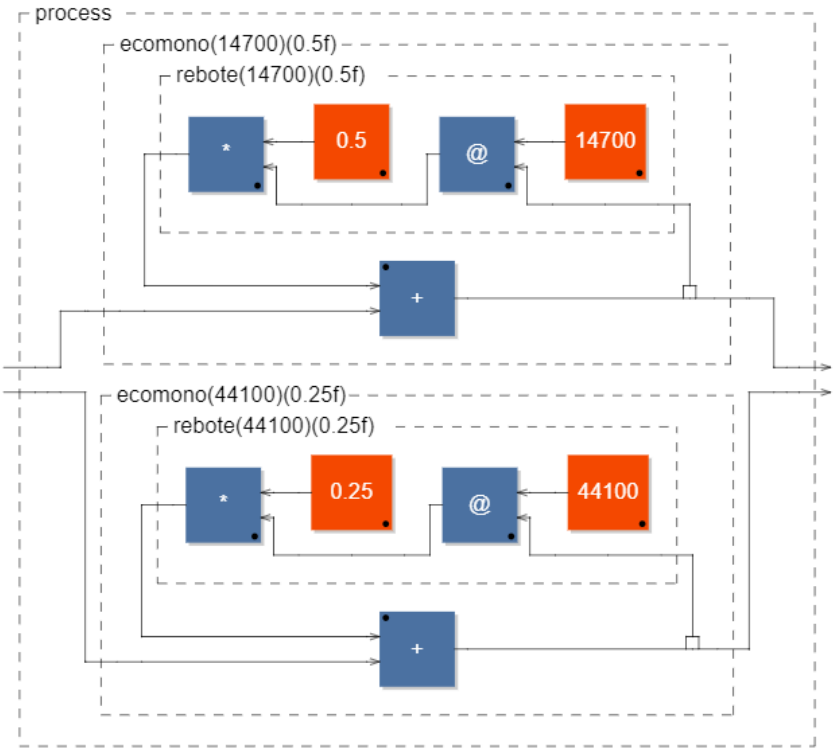
Habiendo creado nuestra definición “ecomono”, es muy sencillo ahora crear un eco estereofónico, simplemente lo definimos como dos “ecomono” en paralelo utilizando la composición paralela `,`. Con el mismo método incluso podríamos crear un efecto multicanal más allá del simple estéreo. Aprovechemos este ejemplo también para ver otra estrategia de definir el tiempo de retardo: si queremos “un cuarto de segundo” basta con escribir 44100/4.




```
rebote = @(44100/4) : *(0.75);
ecomono = +~rebote;
ecoestereo = ecomono, ecomono;

process = ecoestereo;
```

Ejemplo 6: Agregar parámetros

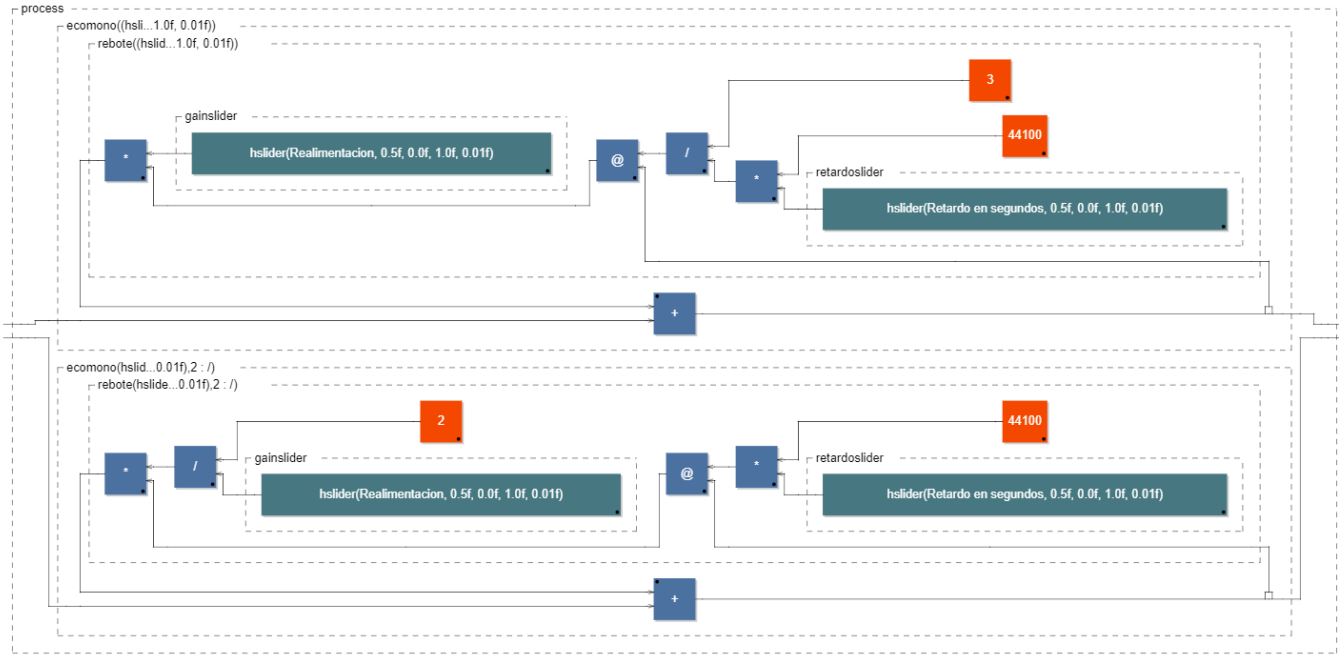
Ahora generalizaremos nuestro eco con ***parámetros*** para controlar su duración y nivel de realimentación, tal como hacemos con las perillas de un efecto en hardware o en un plugin digital. Al crear una definición podemos también crearle parámetros especificándolos entre paréntesis (y separados por comas), para usarlos luego como variables. Finalmente, podremos asignarle los valores que querramos en cada uso de nuestra definición, de manera que cada instancia pueda tener sus propias características independientemente de las demás. Veamos cómo, con una sola definición de "ecomono", tener distintos tiempos de retardo y ganancias de realimentación en ambos canales.




```
rebote(retardo, gain) = @(retardo) : *(gain);
ecomono(retardo, gain) = +~rebote(retardo, gain);

// ahora el canal izquierdo recibira la 3ra parte de tiempo de delay que el derecho
// y el derecho tendra la mitad de ganancia de realimentacion
ecoestereo(retardo, gain) = ecomono(retardo/3, gain), ecomono(retardo, gain/2);

process = ecoestereo(44100, 0.5);
```

Ejemplo 7: Control deslizante para la realimentación

Ahora, en lugar de valores fijos, podemos agregar un control deslizante para controlar el nivel de realimentación y el tiempo de retardo.

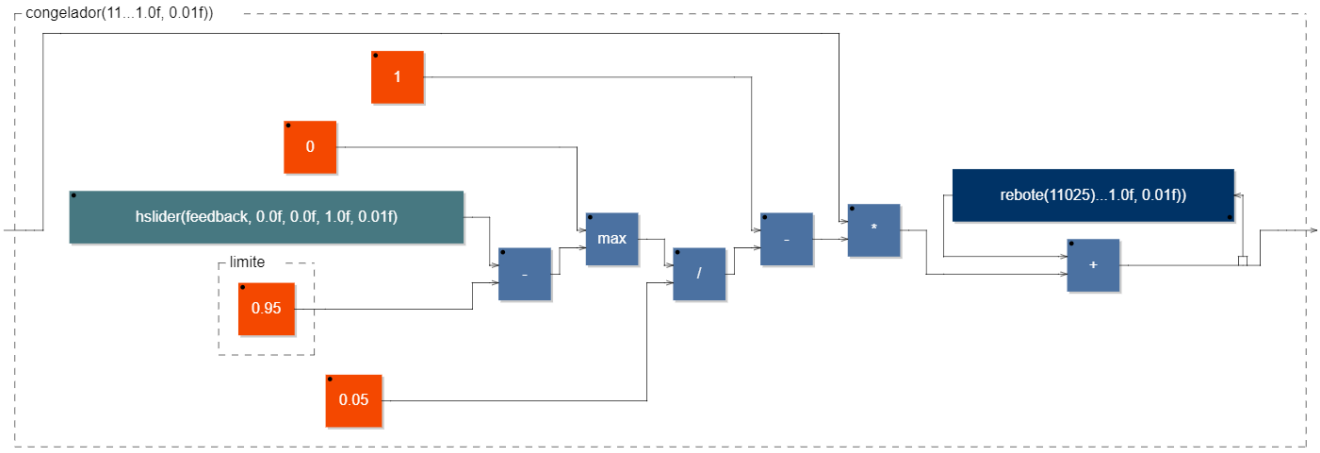



```
rebote(retardo, gain) = @(retardo) : *(gain);
ecomono(retardo, gain) = +~rebote(retardo, gain);

// mantendremos ambos canales ligeramente distintos
ecoestereo(retardo, gain) = ecomono(retardo/3, gain), ecomono(retardo, gain/2);

retardoslider = hslider("Retardo en segundos", 0.5, 0, 1, 0.01);
gainslider = hslider("Realimentacion", 0.5, 0, 1, 0.01);

// multiplicamos por 44100 para tener la cantidad de muestras de retardo
process = ecoestereo(retardoslider * 44100, gainslider);
```

Aunque el diagrama se vuelve cada vez más grande, no deja de ser una simple duplicación de los mismos procesos que venimos trabajando.

Ejemplo 8: Efecto de congelamiento

Solo para jugar un poco, nos gustaría evitar que el nivel de sonido aumente caóticamente cuando establecemos el nivel de realimentación en 1 o más allá, de manera que podamos "congelar" el eco logrado. La idea es cerrar gradualmente la entrada cuando el nivel de realimentación exceda un cierto umbral que llamaremos "límite". Para ello nos valdremos de la definición integrada o ***función*** `max(x, y)`. Esta función simplemente tomará dos argumentos (x, y) y nos dirá cuál valor es el mayor. Con un poco de creatividad matemática, podemos utilizarla para dejar pasar la señal sin cambios (multiplicándola por 1) mientras la ganancia no supere nuestro límite. En caso que lo supere, la señal de entrada se reducirá gradualmente hasta llegar a cero, pero sin afectar el nivel de la señal ya procesada con el eco. De esta manera podemos crear un efecto en el que un bloque de tiempo se repita indefinidamente a modo de congelamiento sonoro.




```
rebote(retardo, gain) = @(retardo) : *(gain);

// reducimos progresivamente la señal original si el gain supera un "limite"
// quedando solo la señal del retardo ya existente en el bucle de realimentacion
limite = 0.95;
congelador(retardo, gain) = *(1 - max(0, gain - limite) / (1-limite)) : +~rebote(retardo, gain);

ecoestereo(retardo, gain) = congelador(retardo, gain), congelador(retardo, gain);
process = ecoestereo(44100/4, hslider("Realimentacion", 0, 0, 1, 0.01));
```

Notaremos el efecto si dejamos reproducir la señal y subimos la ganancia de realimentación por encima de 0.95. Podemos hacer esto varias veces e ir eligiendo qué parte del sonido y con cuánto del eco acumulado se congelará.

Hemos visto cómo trabajar con retardos simples y bucles de realimentación. En la siguiente parte de este tutorial veremos cómo generar osciladores y algunas funciones nuevas.