Trabajo Práctico
Teoría de Lenguajes
2do Cuatrimestre - 2013

1. Introducción.

El SuperCollider[1] es un ambiente y lenguaje de programación para la síntesis de audio y composición algorítmica. Provee un lenguaje interpretado orientado a objetos que funciona como cliente a un servidor de síntesis de sonido en tiempo real. Es de código abierto (GPL) y usado por músicos, científicos y artistas que trabajan con sonido.

El objetivo del trabajo práctico es desarrollar un analizador sintáctico que verifique la sintaxis de un programa escrito en un lenguaje inspirado en el SuperCollider y que genere sonido. Lo que usaremos en este trabajo práctico es mucho más sencillo que el lenguaje original, aunque, esperamos, con suficiente poder de expresividad como para obtener resultados interesantes. El trabajo práctico tendrá dos entregas. La primera consistirá en escribir una gramática para un lenguaje inspirado en el SuperCollider(ver Seccion 4). Los detalles de su funcionamiento los veremos más adelante. Sin embargo, introduciremos algunos conceptos elementales y describiremos parte de su funcionamiento para que se pueda ir viendo que es lo que hace. Los objetivos de la segunda etapa se presentarán más adelante.

El lenguaje que se describe en este enunciado no posee todos los elementos de un lenguaje de programación y puede pensarse básicamente como expresiones algebraicas con arrays.

El tipo de dato que se utiliza es una concatenación de números reales a los que llamaremos *buffers*. Estos buffers pueden pensarse sencillamente como arrays de tipo *float*. Inclusive valores individuales como *0.5* serán considerados buffers de longitud 1.

El lenguaje permitirá crear buffers de distintas formas. Estos buffers podrán ser modificados mediante concatenaciones, redimensiones, y operaciones algebraicas sobre su contenido. Finalmente, en un caso típico, el buffer será enviado a un dispositivo de audio para producir sonidos.

[1] Sitio oficial del SuperCollider: http://supercollider.sourceforge.net/

2. Elementos del lenguaje.

2.1. Generadores:

Hay unas pocas funciones que crean nuevos buffers. Estas funciones se llaman *Generadores*. Algunas de estas funciones tienen dos formas de escribirse, una abreviada y otra completa.

- sin(f, a) : Genera una onda sinusoidal de f ciclos con amplitud a.
- lin(a, b), linear(a, b) : Genera una onda lineal que comienza con el valor a y termina con el valor b.
- sil(), silence() : Genera silencio.
- noi(a), noise(a) : Generan ruido blanco con amplitud a.
- v_1; v_2; v_3; ... : Permite escribir buffers explícitamente.
  (e.g. { 1; 2; 3; 4 }, { -1.0; +1.0 }, etc.)

Los generadores anteriores, salvo en la forma explícita, crean buffers de un tamaño predeterminado que tendrá relevancia más adelante y que está relacionado con la calidad del sonido. Es decir, cuantos valores (muestras) por segundo se enviarán al dispositivo de audio para producir los sonidos.

2.2. Operadores:

Además es posible realizar operaciones algebraicas entre buffers. Todos los operadores son binarios (toman dos argumentos) y se pueden usar por su símbolo o por su nombre. A continuación se muestran las operaciones válidas junto con ejemplos de cual será el resultado de usarlas entre dos buffers de longitud 2.

descripción	símbolo	nombre	ejemplo	resultado
concatenación	;	con	{a; b} ; {x; y}	{a; b; x; y }
mezcla	&	mix	{ a; b} & {x; y}	{ (a+x)/2; (b+y)/2 }
suma	+	add	{ a; b} + {x; y}	{ a+x; b+y }
resta	-	sub	{ a; b} - {x; y}	{ a-x; b-y }
producto	*	mul	{ a; b} * {x; y}	{ a*x; b*y }
división	/	div	{ a; b} / {x; y}	{ a/x; b/y }

Cuadro 1: Operadores

Algunos detalles que se deben notar. Las llaves '{' y '}' sirven para agrupar operaciones de la misma forma que los paréntesis en las operaciones matemáticas. El punto y coma ';' es el operador de concatenación, esto es lo que permite generar buffers explícitamente. Aunque el uso de las llaves no es obligatorio, se recomienda usarlas cuando se esté trabajando. Por ejemplo:

- 1; 2; 3; 4 : es válido, pero poco claro y potencialmente problemático.

- { 1; 2; 3; 4 }: es la forma recomendada de notación.

Para una operación aritmética *OP* de las anteriores entre un buffer *{ a; b; c}* y un valor numérico x (es decir, un buffer de longitud 1) el resultado será:

- x *OP* { a; b; c } = { x *OP* a; x *OP* b; x *OP* c }
- { a; b; c } *OP* x = { a *OP* x; b *OP* x; c *OP* x }

Los casos de operaciones entre buffers de distintas longitudes los veremos más adelante. Por ahora alcanza con esto para poder trabajar.

2.3. Métodos:

Estos buffers pueden pensarse como objetos con sus propios métodos.

- { ... }.play(s) : Reproduce el buffer como sonido a velocidad s.
- { ... }.post : Imprime el contenido del buffer en la salida estándar.
  (Solo útil para buscar errores)
- { ... }.loop(t) : Replica el contenido del buffer t veces.
  (e.g. {1,2}.loop(2) = {1,2,1,2})
- { ... }.tune([+/-] n) : Modifica el buffer en n semitonos.
- { ... }.fill(n) : Aumenta o reduce el tamaño del buffer de acuerdo a la calidad del sonido predeterminada.
- { ... }.reduce : Reduce el contenido del buffer de acuerdo a la calidad del sonido predeterminada.
- { ... }.expand : Aumenta el contenido del buffer de acuerdo a la calidad del sonido predeterminada.

El funcionamiento de los métodos *tune*, *fill*, *reduce* y *expand* se verá en detalle más adelante. Por ahora es suficiente decir que *tune* modifica el buffer para que la frecuencia a la que se reproduzca esté n semitonos más arriba o más abajo. El método *fill* es un cambio en la longitud del buffer, si se pide una longitud menor de la que tiene se perderán valores y si se pide una mayor se completará con ceros. Para utilizar una analogía, es como recortar o cambiar el tamaño de una imagen, pero sin comprimirla o estirarla. De forma similar, *reduce* y *expand* también cambian el tamaño del buffer, pero sin perder ni agregar valores, sino que interpolan los existentes. Continuando con la analogía, estas serán las operaciones para comprimir o estirar la imagen.

El resultado de usar cualquier método es siempre un buffer, por lo tanto es posible utilizar varios métodos consecutivamente. Por ejemplo:

- { 0; 1 }.loop(3).post

Dará como respuesta.

- 0 1 0 1 0 1

2.4. Programa:

Un programa en este lenguaje estará compuesto por una secuencia de buffers con sus correspondientes métodos separados por espaciadores (espacios, tabuladores, fines de linea).

Algunas llamadas a métodos pueden tener valores predeterminados (e.g. la velocidad de reproducción del play puede ser 1 si no se especifica otra cosa). Si no se va a pasar ningún parámetro en la llamada los paréntesis vacíos pueden ser omitidos.

Se considerará comentario, y podrá ser ignorado, a todo texto entre dos barras // y hasta el fin de linea.

Es posible usar espaciadores libremente para separar valores de los operadores y entre los parámetros.

El resultado de una operación devuelve un nuevo buffer. Por ejemplo, las siguiente expresiones son equivalentes.

- { 1; 2; 3; 4 } * 2 = { 2; 4; 6; 8 }
- { 8 / { 1; 2; 4 } } + 1 = { 8; 4; 2 } + 1 = { 9; 5; 3 }
- { 1; 2; 3; 4 } add { 1; 0; -1; 0 } = { 2; 2; 2; 0 }

3. Ejemplos:

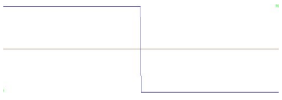
Con todos los elementos de lenguaje ya definidos, veamos qué cosas se pueden hacer con el mismo mediante unos ejemplos.

Algunas equivalencias:

- { { { ... } } } = { ... }
- 1; 2 = { 1 } ; 2 = 1 ; { 2 } = { 1; 2 }
- { { ... }.loop(4) }.play = { ... }.loop(4).play
- { sin(8); sin(4) }.play = { sin(8).play; sin(4).play }

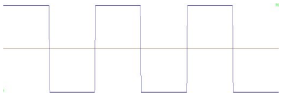
Onda cuadrada de un ciclo:

{ 1; -1 }.expand



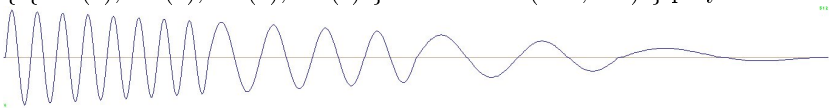
Onda cuadrada de tres ciclos:

```
{ 1; -1 }.loop(3).expand
```



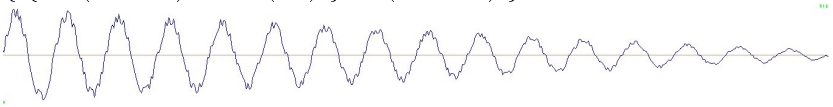
Drums:

```
{ { sin(8); sin(4); sin(2); sin(1) }.reduce * lin( 1.0, 0.0) }.play
```



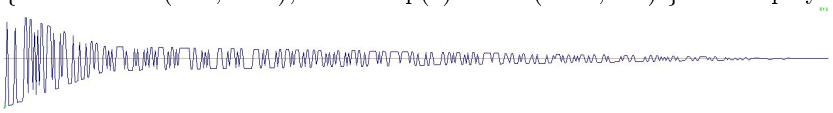
Snare:

```
{ { sin( 16, 0.9)+noise(0.1) }*lin( 1.0, 0.1) }.play
```



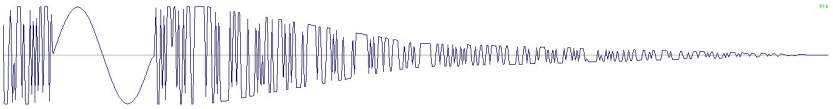
Hi-Hat:

```
{ noise*linear( 1.0, 0.25); noise.loop(7)*linear( 0.25, 0.0) }.reduce.play
```



Claps:

```
{ noise; sin(1); noise; noise.loop(3)*lin(1.0, 0.25); noise.loop(6)*lin(0.25, 0.0)
}.reduce
```



Instrumentos de percusión en una base rítmica:

```
{ {
   { sin(8); sin(4); sin(2); sin(1) }.reduce * lin(1,0); silence.loop(7)
} mix {
   { sin( 16, 0.9)+noise(0.1) }*lin( 1.0, 0.1); silence.loop(3) }.loop(2)
} mix {
   { { noi*lin( 1.0, 0.25); noi.loop(7)*lin( 0.25, 0.0) }.reduce; sil }.loop(4)
} }.loop(16).play
```

4. Detalle de la entrega

Se solicita un informe conciso, con ortografía y tildes revisadas y que contenga los siguientes items (además de introducción, conclusiones y demás elementos que consideren necesarios para que sea de calidad):

- una gramática libre de contexto para el lenguaje antedicho,
- decisiones tomadas y su justificación.

El informe debe entregarse impreso y además un archivo comprimido a la dirección:

tptleng@gmail.com.

Fecha de entrega: 23 de Octubre de 2013.