Bb: Lenguaje Musical

Cristina Raluca Vijulie, Victor Hugo Rivero

June 2, 2017

1 Introducción

1.1 Descripción del lenguaje

Bb es un lenguaje de programación que genera archivos de sonido. El compilador usa la librería *"javax.sound.midi"* para generar un archivo MIDI a partir del código en Bb.
Bb es un lenguaje compilado, procedural y funcional, de tipado dinámico.

1.2 Objetivos

El objetivo de Bb es proporcionar una herramienta simple de generación de sonido automatizable tanto a músicos como programadores. La mayoría de progrmaas de edición de partituras requieren cierto conocimiento de nociones musicales y además muchas veces dificultan la tarea de escribir musica, ya que se tiene que hacer "a mano". Muchas veces queremos oir como suenan conjuntamente ciertas notas o ver cual es la mejor combinacion entre un conjunto de notas, pero los editores de musica hacen de esto una tarea pesada, teniendo que escribir las notas una por una o copiar y pegar trozos ya escritos. Con Bb es muy fácil escribir funciones que modifiquen un acorde o eneren acordes, escalas o melodías a partir de notas base.

1.3 Funcionalidades principales

El lenguaje Bb nos permite crear y reproducir una variedad de entidades musicales:

- **Nota**:
  Una nota se define por una letra de 'A' a 'G', definiendo el tono, y opcionalmente un número del 0 al 9 para definir una octava. Si ninguna octava es especificada se usará la octava céntrica en un teclado de piano. Una nota puede tener también una alteración, que puede ser '#' para subir un semitono o 'b' para bajarlo. Ejemlos de notas serían por ejemplo:

  $$\mathbf{D\ b}, \mathbf{B\#8}, \mathbf{C8}, etc.$$

- **Acorde**:

  Un acorde es formado por una serie de una o más notas escritas entre paréntesis que se tocan a la vez. Ejemplos de acordes serían:

$$(C, E, G), (C3, E1, G8), (C3, Eb3, G3, C4, Eb4, G4)$$

- **Melodia**:

  La melodía es una entidad reproducible formada por Notas, Acordes y una definición de instrumento, es decir, a partir de una melodía se puede generar un sonido MIDI, ya que la melodía, además de las dos entidades mencionadas anteriormente, asigna una durción a cada uno de sus componentes, y también define el instrumento que debe generar el sonido. Así pues, cuando escribimos una nota o un acorde dentro de una melodía debemos asignarle una duración en tiempos. Estas duraciones se expresan como números naturales. Podemos usar el operador [ ] para dar la misma duración a un conjunto de notas y acordes de la siguiente manera:

$$\textbf{Melody}(1[E, F\#, G, (A, Bb, C)].8*)$$

  Esta melodía usa el instrumento 1, que es Piano. Luego toca por orden las notas y acordes especificados, todos con duración 8*.

- **Polifonía**

Las notas y acordes son atómos de las melodías y polifonias, por tanto pueden existir sin tener definida una duración, lo que nos permite asignarlas a variables y usar estas variables para crear melodías y polifonías, asignando la duración deseada en cada caso. Un ejemplo de polifonía es:

```
Poli {
Melody(15 [A 3,B 3,C 3,D 3,E 3,F 3,G 3,A 3,B 3,C 3].16*) |
Melody(40 [A 4,B 4,C 4,D 4,E 4,F 4,G 4,A 4,B 4,C 4].16*) |
}
```

Cualquier expresion musical con una duración definida puede ser reproducible. El lenguaje Bb transforma en MIDI todo lo que tocamos, por orden de ejecución. Para más facilidad y más funcionalidades, las notas y acordes se pueden asignar a variables, que luego podrán ser modificadas y reutilizadas para tocar coses distintas. Por ejemplo:

2 Gramática

2.1 Modificaciones principales respecto ASL

- Al tener que reproducir sonidos, se ha añadido la regla playable que contiene todos los sonidos que serán reproducidos en el MIDI generado. Esta regla se llama desde la regla instruction.

Existen reglas para las melodias, las notas, las polifonias y un imaginary token PlAYABLE para las variables de entidades musicales. También se diferencias IDs de variables no musicales de las musicales (NOTEID), que empiezan en minúsculas y en mayúsculas respectivamente.

- Las notas son un tipo de entidad musical, que representan una sola nota musical. Por lo que se han añadido tokens que codifican las notas en cifrado americano (C, D, E, F, G, A, B) y el token QUIET que significa no reproducir ningun sonido, como reglas para representarlas, como notabasica que tenga el pitch, la alteración, y la octava, y NOTE como imaginary token para agrupar los elementos de notabasica bajo una misma hoja del AST.

- Los acordes son otro de los tipos de entidades musicales, que representan un conjunto de notas musicales que se reproducen al mismo tiempo. Esto ha implicado añadir nuevas reglas, chord y subchord, para expresar una lista de notas y el imaginary token CHORD para tener todas la notas bajo una misma hoja.

- Las melodias son otro de las entidades musicales que soporta el lenguaje, y representa una sucesión de sonidos con duración. Las melodias tienen también reglas para expresarlas. También tiene un imaginary token MELODY para agrupar sonidos. Son la entidad que puede especificar instrumento específico.

- Las polifonías son la última entidad musical nueva, y representa un conjunto de melodías que suenan a la vez. Esto ha supuesto añadir las reglas polifon, que agrupa un conjunto de voices, y la regla voices que agrupa melodías o NOTEIDs. También ha supuesto la adición del imaginary token POLIFONE para agrupar a las voices.

- Tambien se pueden controlar en cierta medida los instrumentos que suenan. Para esto se ha agregado la regla instrument, para la instruccion de cambiar de instrumento por defecto, con un número del 0 al 127.

- Para controlar el tiempo de lo que se reproduce, se ha agregado la regla para la instrucción Speed, que permite modificar en cualquier momento del programa el tempo de la canción, en bpm. También se ha escogido agregar un imaginary token SPEED, para tener el valor de la velocidad.

- Para controlar el volumen de lo que se reproduce, también se ha agregado la regla volume. Se puede cambiar el volumen del sonido en cualquier momento. Para esto se ha agregado un imaginary token VOL.

- Al tener diferentes tipos de variables, también se necesitan formas diferentes de declararlos y asignar valores. Por lo que se ha modificado la regla assign para poder asignar valores a las entidades musicales explicadas anteriormente.

3 Semántica

3.1 Data

La clase Data se ha ampliado para poder albergar 4 tipos de datos auxiliares: Note, Chord, Melody, Polifony. El valor de cada uno de estos tipos de datos se guarda en una clase separada, siendo Note y Chord hijos de una clase genérica Sound. Esto es debido a que tanto la nota como el acorde tienen una duración (expresada en cuartos de nota), un volumen (del 0 a 127), un instrumento y un sonido. En este caso, la clase Sound tiene un array de sonidos (representados como int). Cuando se trata de una nota solo es válida la primera posición del array, y cuando se trata de un acorde el array tiene todas las notas que van a reproducirse a la vez. La melodía está implementada como un array de sonidos y la polifonía como un array de melodías.

3.2 BbTree

Se ha añadido una función para calcular el pitch de las notas (un entero a partir del nombre de la nota) durante el pre-procesamiento del AST. Para calcular este entero se ha usado la siguiente tabla de conversión de nota a valor midi.

Octave	Note Numbers											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

3.3 Player

El player se encarga del paso del conjunto de elementos Data a un elemento tipo Sequence con la información de la canción, lista para ser escrita en un fichero.

Esta clase usa la libreria midi de java *javax.sound.midi*. La función de esta clase consiste en interpretar los valores que tiene los Data y cargar una variable Sequence que luego se retorna. Este Sequence contiene un Track que se crea en el constructor de la clase.

Para conseguir cargar la información de los Data en el Track, se usan los mensajes de MIDI. La libreria usada permite cargar estos mensajes en el track. Estos mensajes son como instrucciones para generar sonido.

En el constructor se encuentran los mensajes que inicializan el fichero de Midi, activar la polifonia e instrumento por default, etc.

Luego tiene la función principal usada por el interprete, al que se le pasa un arraylist de pares de Data y Double. Esta función itera sobre el arraylist y dependiendo del tipo que contenga el Data (NOTE, CHORD, MELODY, POLIFONY), se setea el tiempo y el instrumento, y se llama a otra función auxiliar que iterará sobre cada nota del Data.

La función auxiliar que se llama desde la función principal, se encarga de llamar a otra funcion para cada sonido. Esta última función genera los mensajes de cada sonido en el momento que le corresponde.

Finalmente, se produce el mensaje que marca el final del midi.

3.4 Interp

- Run:

  Ahora la funcion Run además de ejecutar el main, escribe el resultado del código en forma de archivo MIDI y también por pantalla. La clase Interp tiene una instancia de Player, al que le pasa un array de parejas de Data y double. Eso es debido a que durante la ejecución del programa, la velocidad de reprodcción puede cambiar, y debemos decirle al Player según que velocidad se debe interpretar la duración de cada Data. También añadir que los Datas que le pasamos al player son solo de tipo musical (Note, Chord, Melody, Polifony).

- ExecuteInstructions:

  Se han añadido los casos necesarios para tratar las asignaciones de expresiones musicales, el bucle for, los playables, etc. Para la correcta interpretación de estos nodos también se usan fuciones auxiliares:

  - evaluateMusicNotation y evaluatePlayable:

    la primera función avalua el AST perteneciente a una notación musical y devuelve el data correspondiente. Funciona mano en mano con la segunda, que avalua un nodo del AST de tipo "playable", es decir, que se puede reproducir. Estos nodos se caracterizan por tener un hijo de un tipo básico (nota o acorde) y otros hijos para definir su duración. Cuando tenemos que asignar un playable a una variable, la funciona evaluateMusicNotation lee el tipo contenedor de los playables y usa la función evaluatePlayable para obtener el array de datas que ya contienen duración, volumen, instrumento, etc. En el caso de las melodías, se sobreescribe el instrumento acorde con el especificado en el AST de la melodía.

  - evaluateDuration:

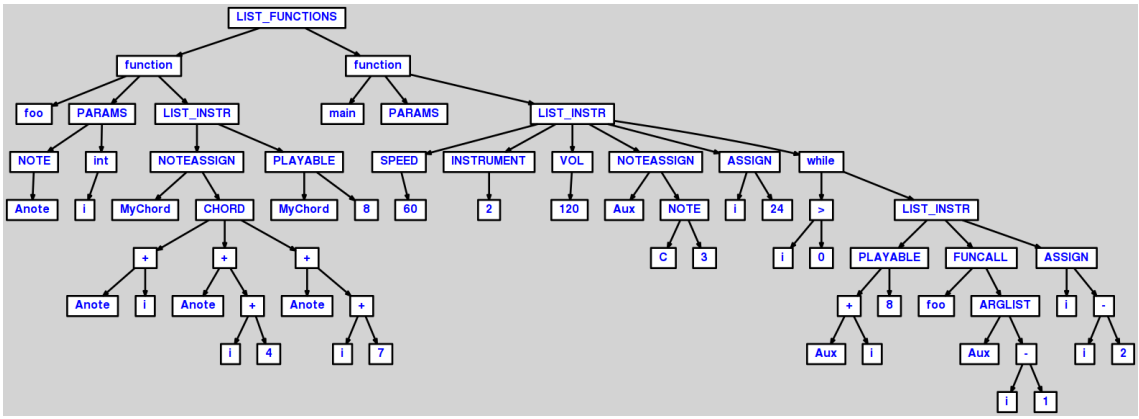
    Esta función devuelve la duración de un sonido en cuartos de nota. Para ello solo se tiene que dividir 4 por el valor escrito en el código.

– listArguments:

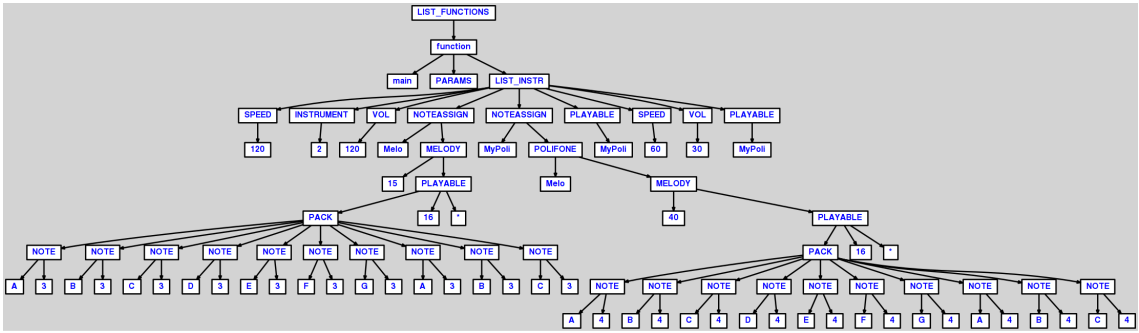
Se ha modificado la función para poder aceptar notas, acordes, etc. como parámetros de funciones.

4 Juegos de pruebas

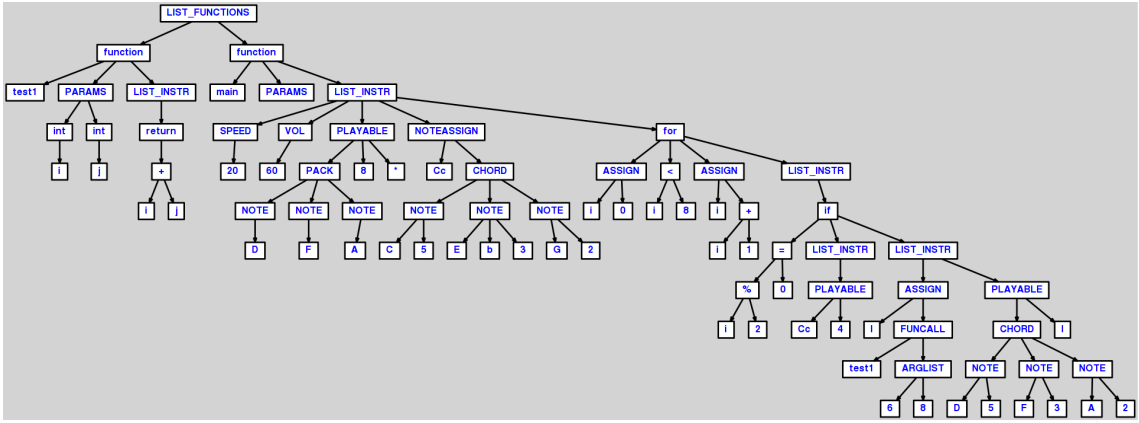
- chromaticFunctions.m: esta prueba está encarada a verificar el correcto funcionamiento de las funciones para resolver el tono de los acordes y notas, la función de subir el tono mediante la función auxiliar (+ expr), las funciones y la instruccion while.



- poliSpeed.m: aquí probamos el correcto funcionamiento de los polífonos, melodías y del uso de la función speed para cambiar la velocidad de una misma expresión musical.



- itereturn.m: en este programa probamos el return de las funciones, junto con instrucciones "if then else" y instrucciones for. También probamos los "packs" de notas, donde ponemos distintas notas con una misma duración.



- GiantSteps.m: en este programa se ha programado la base melódica de la famosa canción "Giant Steps". Se cambia varias veces de instrumento para probar la instrucción "Instrument" y también la llamada de funciones. Se omite el AST por cuestiones de espacio. El código es el siguiente:

```
function maj7( Note Rr, int l){
    (Rr, Rr (+ 4), Rr (+ 7), Rr (+ 11)).(l);
}
function dom7( Note Rr, int l){
        (Rr, Rr (+ 4), Rr (+ 7), Rr (+ 10)).(l);
}
function min7( Note Rr, int l){
        (Rr, Rr (+ 3), Rr (+ 7), Rr (+ 10)).(l);
}
function main(){
    Speed 60;
    Instrument 1;
    Vol 80;
    maj7( Note B,2); dom7( Note D , 2);
    maj7( Note G , 2); dom7( Note B b , 2);
    maj7( Note E b , 1);
    Instrument 40;
    min7( Note A , 2);  dom7( Note D , 2);
    maj7( Note G , 2); dom7( Note B b , 2);
    maj7( Note E b , 2);  dom7( Note F# , 2);
    maj7( Note B , 1);
    Instrument 32;
    min7( Note F , 2); dom7( Note B b , 2);
```

```
    maj7( Note E b , 1);
    Instrument 25; dom7( Note D , 2);
    maj7( Note G , 1);
    Instrument 19;
    min7( Note C# , 2); dom7( Note F# , 2);
    maj7( Note B , 1);
    Instrument 48;
    min7( Note F , 2); dom7( Note B b , 2);
    maj7( Note E b , 1);
    Instrument 73;
    min7( Note C# , 2); dom7( Note F# , 2);
}
```

5 Posibles extensiones no implementadas

5.1 Crescendo y diminuendo

Una buena opción extra seria crescendo y diminuendo, es decir, el aumento o la disminución gradual del volumen. Se podría indiar un volumen inicial y otro volumen final, y la porción de código que afecta el degradado.

5.2 Preincremento y postincremento

El operador de pre y postincremento ++ y – como en lenguajes como C, no es soportado por Bb. Tampoco soporta los operadores de asignacion +=, -=, *=, /=. Para implementar esta extensión, se habría de cambiar la grámatica y el interprete, mas no el módulo de MIDI.

5.3 Acceso a posiciones arbitararias de Polifonias y Melodias (array)

La posibilidad de usar arrays de cualquier tipo de variables, especialmente de tipos musicales sería útil para acceder a notas concretas de una melodía o de un acorde, y tambien poder modificar voces concretas de un políono sin tener que redefinirlo entero. (Por ejemplo modificar el instrumento de una de las voces de un polifono).

5.4 Armaduras

Las armaduras podrían ser útiles para evitar tener que escribir a mano todas las alteraciones cuando se está progrmando. Se podrían cambiar en cualquier lugar del codigo, similarmente a cómo se ha implementado la instruccion "Instrument".

5.5 Instruccion para octava

Asi como existe la instrucción para declarar el tempo, el volumen y el instrumento por default, también se podría hacer lo mismo para la octava. La octava por default es 3, lo que quiere decir que si una nota no explicita la octava que es, se asume que pertenece a la octava 3.

Esto puede ser útil en el caso, por ejemplo, que se quiera crear un midi con un acompañamiento de acordes, la octava 3 puede ser muy agudo para lo que se quiere, y podria ahorrar el explicitar el numero de la octava en cada nota. Además que puede ser una forma de buena costumbre declarar la octava por default antes de una sección, en vez de la octava para cada nota.

5.6 Trato especial para las percusiones

La percusión en MIDI no se trata igual que el resto de los instrumentos. Los instrumentos suelen timbres con diferentes frecuencias, mientras que la percusión no suele usar el mismo enfoque.

Un set de percusión suele consister en basicamente un hi-hat, un kick, un snare y quizás un cymbal. Pueden haber más componentes como toms de diferentes frecuencias, y sonidos más exoticos como cowbells.

5.7 Grupos de valoración especial (tresillos, quintillos, etc)

Los grupos de valoración especial consisten en acomodar un número de notas en un espacio donde cabe otra cantidad de notas, como los tresillos que es consiste en tocar 3 notas en donde caben 1.

El player soporta esto, la duración y en cuanto se puede partir depende de los ticks, que arbitrariamente es 48, esto quiere decir que, al ser un quarter note 48 ticks, se puede partir desde esta duración hasta 48. Los tresillos son compatibles, ya que 48 puede dividirse entre 3, pero para quintillos o septillos habria que aumentar el numero de ticks.

Por parte de la gramática, habría que idear una manera de indicar estos grupos. Una opción es usar la funcionalidad de pack para agrupar notas, y luego crear una notación para indicar que el pack es un tresillo, quintillo, etc.

Otra opción es crear una notación que indique que una duración es el tercio, quinto, etc de otra duración. Entonces individualmente se pueden declarar notas con duración de tresillo, quintillos,etc.