Facultad de Ciencias, UNAM.
Programación declarativa.
Proyecto.

Emiliano Galeana Araujo
Juan Alfonso Garduño Solís

24 de junio de 2020

1. Introducción

A grandes rasgos nuestro proyecto es un programa que a partir de un archivo fuente con notas musicales genera ondas de sonido correspondientes a dichas notas para después reproducirlas con un programa externo. Propusimos este tema de proyecto por curiosidad y gusto; curiosidad porque no teníamos mucha idea de como generar sonidos con algún lenguaje de programación y mucho menos con un lenguaje de programación declarativo, pero sabíamos que seguramente era posible y queríamos intentarlo, y gusto porque a quién no le gusta la música. Naturalmente elegimos Haskell porque si nos íbamos a meter en un terreno desconocido, al menos queríamos conocer un poco el lenguaje, y honestamente fue una buena decisión. A continuación explicamos las ideas fundamentales para desarrollar el proyecto, como ejecutar el programa resultante y las conclusiones que pudimos obtener a partir de esta experiencia.

2. Justificación del diseño

Para explicar el desarrollo se deben tener en mente ciertos conceptos de sonido y música. El sonido es una vibración que se propaga, naturalmente en forma de onda y según la forma que tenga la onda es el sonido que podemos percibir, por ejemplo la intensidad del sonido está en función de la amplitud de la onda y el tono del sonido está en función de la frecuencia. Si tomamos una onda, por ejemplo sinusoidal, y modificamos su frecuencia a 440Hz decimos que la nota que genera es un *La*, y como no estamos generando sus armónicos decimos que es una nota pura. Con esto explicado vamos a seguir el proceso por el que pasa el archivo de entrada hasta su reproducción. Entonces, cuando invocamos al función que inicia todo `interactive` lo primero que vamos a hacer es separar el archivo de entrada en una tupla `(t,c)`, dónde `t` va a ser la duración de las notas y `c` va a ser una cadena que contiene todas las notas del archivo, la función encargada de esto es `parseFile` y se auxilia de `concater` y `cleanComment`, a continuación llamamos a la función principal `play` con `t` y el resultado de aplicar la función `translate` a `c`, `translate` lo que va a hacer es arrojar una lista de números flotantes sólo para las notas contempladas, por ejemplo, en nuestro caso tenemos en la lista `notes` 12 posibles notas con sus respectivos ordenes, por lo que los 12 primeros elementos de la lista `['A'..]` tienen una nota asociada, las primeras 7 para las notas naturales y las 5 restantes para las notas sostenidas. Con esto terminamos el proceso de análisis sintáctico del archivo, cabe resaltar que gracias a este diseño agregar notas se vuelve trivial, solo se deben agregar las notas deseadas a la lista `notes` del archivo *Proyecto.hs* sin perder de vista que los elementos de tipo `Note` representan a una nota en la escala musical, esta nota está representada por su nombre y por un lugar en la escala, esto quiere decir que la escala tiene un orden, por ejemplo `Do` es la primer nota, por lo que tiene asociado el número 0.0. La razón de usar flotantes y no

enteros es que eso nos evita parsearlo en la parte de `Proyecto.wave`.
Retomando la explicación del funcionamiento de la función `play`, en esta es que vamos a utilizar todo lo que se explicó al principio sobre el sonido, esta función lo primero que va a hacer es guardar en un archivo binario las ondas que generan las notas que acabamos de traducir a números, lo importante es como construimos las ondas. De esto se encarga la función `wave` que toma la duración deseada de las notas y las notas para a continuacion calcular una por una frecuencia con la función `fp` que se basa totalmente en la referencia[1] y seguidamente construya la verdadera onda con `freq`, justo en `freq` es cuando se ve como modificamos la amplitud de la onda con la constante `volume` y la duración con la constante `sampleRate` y el periodo de la onda que estamos utilizando (claramente el seno) 2π.
Construidas y guardadas las ondas solo nos falta reproducirlas, cosa que vamos a hacer con `runCommand` y el parámetro

```
ffplay -f f32le -ar %f cancion.bin
```

Dónde:

ffplay es la herramienta de FFmpeg que vamos a utilizar.

-f f32le forza a ffplay para que leea el archivo como números flotantes little-endian.

%f es sustituido por el sample rate (número de muestras a reproducir por segundo).

cancion.bin es el archivo a leer.

Y con esto terminamos la ejecución del programa si todo sale bien.

3. Ejecución

Para ejecutar correctamente el programa se debe de tener el software FFmpeg instalado en la computadora ya que este es el encargado de reproducir las notas generadas por nuestro programa a través de la herramienta `ffplay`, la instalación no debería ser complicada. Para el desarrollo del proyecto se utilizó GHC, así que se recomienda que el modo interactivo que se utilice para cargar el proyecto sea `GHCi`, no garantizamos el correcto funcionamiento del proyecto si se utiliza un compilador distinto.

Además de tener el archivo de entrada (la canción), el cuál debe estar construido sigueindo las siguientes reglas:

- El archivo puede tener comentarios, las lineas del archivo que sean un comentario deben de comenzar con el caractér **#**.

Ejemplo: `# Este es un comentario`

- La primer linea del archivo que no sea un comentario debe de ser un número (no necesariamente entero), este número representa el tiempo en segundos que todas las notas van a durar.

Ejemplo: `0.25`
Ejemplo: `1`

- Las lineas siguientes al número del tiempo deben ser comentarios o notas en la notación anglosajona musical separadas por un espacio.

Ejemplo: `C C G G A A G`
Este ejemplo representaría las notas `Do Do Sol Sol La La Sol`

Los archivos `estrellita.je, titanic.je` son ejemplos de archivos con una entrada bien construida y codifican la canción que su nombre lo indica. Suenan relativamente bien, pero no tanto.

Existen dos maneras de ejecutar el programa, con el intérprete y compilando.

3.1. Intérprete

Una vez cumplidos los requisitos anteriores (el archivo bien hecho) se debe abrir una terminal en un directorio que contenga los tres módulos del proyecto, los cuales llevan por nombre `Main.hs, Music.hs, Proyecto.hs` y el archivo que contiene la canción a ejecutar, vamos a decir que este archivo es `estrellita.je`.

En la terminal vamos a ingresar al modo interactivo de GHC con el comando `gchi Main.hs`, ya que estamos dentro con el comando `interactive ''estrellita.je''` se ejecuta el programa y se reproduce la cancion al cabo de unos segundos.

3.2. Compilar (make)

Para esta opción, se requiere el paquete make y las opciones para nuestro programa son:

- `make all`, el cuál nos muestra todas las opciones (Las que se muestran a continuación).
- `make compile`, el cual compila el proyecto.
- `make run FILE=''estrellita.je''`, el cual ejecuta el programa y pasa como argumento el archivo `estrellita.je`.
- `make clean`, el cual limpia los archivos que se generan cuando se compila, así como el ejecutable y el archivo que interpreta `ffplay`.

Adionalmente, se puede compilar el proyecto con las instrucciones que se encuentran en el `Makefile` (En caso de que no se tenga el paquete `make` o se tenga desconfianza de lo que este pueda hacer).

4. Trabajo a Futuro

Existen varias bibliotecas[2] que se pueden utilizar para generar un sonido más puro, incluso usando varios instrumentos; Sin embargo nos fue complicado usarlas, por cuestiones de `Cabal`, o del reproductor de música (Se necesita un reproductor que pueda interpretar archivos `MIDI`). Así que portar el proyecto para que pueda producir sonidos más puros sería un buen avance.

Otra cosa es que cualquier música que queramos reproducir, las notas tienen un tiempo defindo y es el mismo (La primer línea de los archivos `*.je`), poder cambiar el sonido en distintas notas sería un buen avance para producir música más del estilo que conocemos. Lo último es que, debido a como generamos el sonido, algunos archivos pueden llegar a pesar mucho (Como el `cancion.bin` de `titanic.je`), esto por el número de ondas que generamos, como se dijo anteriormente, ese número fue el más aproximado a un sonido grave, reducir el tamaño de los archivos de salida sería algo ideal para una segunda versión del proyecto.

5. Conclusiones

Normalmente como programadores no salimos de ciertas zonas de confort (paradigmas) porque son con las que nos instruyeron o son las maneras más naturales de pensar una solución a los problemas que se nos pueden presentar, sin embargo hay formas de solucionar los mismos problemas con distintos enfoques. Para realizar un ejercicio como el presentado en este proyecto lo último que alguien normalmente pensaría sería programarlo en Haskell o cualquier otro lenguaje declarativo porque es difícil ver el camino a seguir, pero es posible y tal vez hasta es más fácil y rápido que el camino habitual. Si bien el proyecto honestamente no tenga mucha utilidad, es el fruto del uso de prácticamente todo lo revisado en el curso, desde utilizar elementos que nos brinda el lenguaje y el paradigma como listas por comprensión, caza de patrones, mónadas u operadores fold, hasta estrategias (mañas) para solucionar problemas con

cierto patrón. Claro que además de la capacidad de modelar el problema en la computadora, en este caso particular modelar las ondas de sonido, dicha capacidad adquirida a lo largo de la carrera.

Entonces, en conclusión podemos decir que el proyecto fue divertido, aprendimos varias cosas (No solamente de computación), incluso estructuras de datos que nunca habíamos usado en un entorno declarativo:

```
translate (x:xs) =
if isAlpha x
then
let
posible_note = Map.lookup x mapi
in
case posible_note of
Just (note, ord) -> [(ord + 40)] ++ translate xs
Nothing -> [] -- omite las letras que no estn.
else translate xs
```

Podría ser algo que desarrollemos mejor en un futuro, no es de mucha utilidad, ya que existen mejores maneras de crear música, incluso con código, más aún, herramientas que utilizan haskell[3], pero sin duda es algo distinto a todo lo que habíamos hecho antes en la carrera.

Referencias

[1] *Physics of Music - Notes*. https://pages.mtu.edu/~suits/NoteFreqCalcs.html.

[2] P. Hudak, "The haskell school of music," Yale University Department of Computer Science, 2011.

[3] *TidalCycles*. https://tidalcycles.org/index.php/Welcome.

[4] Lemnismath, "¿por qué tenemos 12 notas musicales? | música y matemáticas," 2018. https://www.youtube.com/watch?v=P7iC-fbdKmQ&t=1s.