Autómatas, Teoría de Lenguajes y Compiladores

Informe TPE

Coffee-Music

Integrantes:

- María Victoria Cavo - 53202
- María de los Ángeles Arlanti - 53373
- María de las Mercedes Varela - 51332
- Jeremías Aagaard - 53219

Índice

- Informe TPE
  - Índice
  - Idea subyacente y objetivos del lenguaje
  - Consideraciones Realizadas
  - Descripción del desarrollo del TP
  - Descripción de la gramática
  - Dificultades encontradas en el desarrollo del TP
  - Futuras extensiones
  - Referencias

Idea subyacente y objetivos del lenguaje

La idea original del proyecto en un principio era lograr componer, a través de un archivo de entrada, un tema musical en donde se podían definir componentes tales como estrofas, estribillos, las notas a tocar y otros atributos que hacen a la composición de la música.

Ante varias dificultades encontradas al intentar realizar dicho proyecto, se terminó optando por reducirlo a simplemente la interpretación de notas musicales, representadas en su sistema americano, donde cada nota es equivalente a una frecuencia específica para ser reproducida como audio. De esta forma se logró el objetivo principal del equipo de crear alguna forma de música a partir de una gramática, por más de que haya terminado siendo más reducidas de la que se esperaba inicialmente.

Consideraciones Realizadas

A lo largo del desarrollo del compilador se tomaron las siguientes consideraciones:

- El lenguaje podrá usar sólo variables del tipo *int*, y *music*, donde *music* es una cadena de caracteres que representa notas musicales.
- Todo símbolo '-' inmediatamente seguido por un número será tomado como parte del número, significando que es un número negativo, mientras que si está separado por un espacio será interpretado como la resta.
- No se ofrecerá el tipo char* o cualquier otro uso de cadenas que no sea el de *music*, ya que no se vio la necesidad dado el objetivo del compilador.
- Las variables pueden tener cualquier nombre formado por caracteres alfanuméricos, empezados por una letra.
- El manejo de las notas musicales y su transformación en frecuencias de audio se realizará del lado de los archivos de C.

Descripción del desarrollo del TP

Al comienzo ningún integrante del grupo lograba surgir con una idea medianamente interesante de una gramática a desarrollar, y fue cuestión de un buen tiempo hasta que uno de los integrantes del grupo, en su afán y cariño por la música, propuso crear un compilador para interpretar comandos que generen música, una idea muy bien recibida por el resto de los integrantes y desde entonces fue el objetivo del trabajo práctico.

La proposición inicial era crear una gramática que permita varios comandos para realizar la realización de música compuesta por varias melodías, con estribillos, estrofas y muchos otros componentes que hacen a una canción, pero luego por varios motivos y complicaciones se terminó por reducir a la composición de melodías simples formadas por notas musicales con una duración predefinida. Las notas que se aceptan son aquellas correspondientes a las frecuencias centrales y una octava más, lo cual corresponde a las octavas 4 y 5 en un piano, incluyendo sus sostenidos y bemoles. La tabla de frecuencias usada como referencia es la presente en Wikipedia.

Una vez que se dio por definida la gramática, se empezó su desarrollo en YACC y LEX.

Descripción de la gramática

La gramática elegida posee una sintaxis casi idéntica a la sintaxis del lenguaje C, debido a que se encontró esta gramática muy cómoda y de común entendimiento por todos los integrantes del grupo, al ya estar acostumbrados a programas sobre este lenguaje. Las características de la gramática elegida que la diferencian de aquella perteneciente a C son aquellas específicas que hacen de la composición musical, como por ejemplo el tipo de variable *music* que representa una sucesión de notas musicales, o la expresión reservada *writemusic()*, que llama a una función que genera un archivo *.wav* con la secuencia de notas musicales que recibe como parámetro.

Por más que la sintaxis sea como la del lenguaje C, la cantidad de palabras reservadas y la cantidad de instrucciones disponibles es mucho menor, ya que se decidió reducir toda la utilidad de la gramática a la composición de música. Por lo tanto, por ejemplo, los únicos tipos de variables que soporta la gramática son enteros (*int*) y secuencias de notas musicales (*music*), donde éste último es una cadena específicamente compuesta por notas.

La gramática realizada también permite crear ciclos de repetición, para poder repetir ciertas melodías, y realizar bloques condicionales, para generar ciertas cadenas musicales dadas condiciones específicas. Para lograr implementar estos ciclos y bloques condicionales fue necesario el uso de los operadores correspondientes.

A continuación se muestra la descripción de la gramática de acuerdo a sus construcciones:

Program ->	Main
Main ->	*main (void)* MainBlock
MainBlock ->	*{* Content *}*
Content ->	Variables Content | Variables = Expression *;* Content | *name* = Expression *;* Content | *while (* LogicalExpression *)* Block Content | *for (* ForExpression *;* LogicalExpression *;* ForExpression*)* Block Content | *if (* LogicalExpression *)* Block Content | *if (* LogicalExpression *)* Block *else* Block Content | *switch (* Expression *) {* Cases *}* Content | *do* Block *while (* LogicalExpression *) ;* Content | *break ;* Content | *writemusic (name) ;* Content | *writemusic (music) ;* Content | λ
Block ->	*{* Content *}*
Variables ->	Variable *;*
Variable ->	*type name* | *music name*
ForExpression ->	*name* = Expression | ForExpression , ForExpression | λ
Expression ->	LogicalExpression | ArithmeticalExpression | *music*
ArithmeticalExpression ->	Termin | *(* ArithmeticalExpression *) * (* ArithmeticalExpression *)* | *(* ArithmeticalExpression *) / (* ArithmeticalExpression *)* | *(* ArithmeticalExpression *) % (* ArithmeticalExpression *)* | *(* ArithmeticalExpression *) +* *(* ArithmeticalExpression *)* | *(* ArithmeticalExpression *) -* *(* ArithmeticalExpression *)*
LogicalExpression ->	BooleanValue | RelationalExpression | LogicalExpression *&&* LogicalExpression | LogicalExpression || LogicalExpression | *(* LogicalExpression *)*
RelationalExpression ->	*(* ArithmeticalExpression *) <* *(* ArithmeticalExpression *)* | *(* ArithmeticalExpression *) >* *(* ArithmeticalExpression *)*

| *(* ArithmeticalExpression *)* <= *(* ArithmeticalExpression *)*
| *(* ArithmeticalExpression *)* >= *(* ArithmeticalExpression *)*
| *(* ArithmeticalExpression *)* == *(* ArithmeticalExpression *)*
| *(* ArithmeticalExpression *) != (* ArithmeticalExpression *)*

BooleanValue -> *true*
| *false*

Termin *name*
| *number*

Cases -> *case* Expression *:* Block Cases
| *default :* Block

Referencias:

- No Terminales
- *Terminales*
- *Terminales con valor variable*
- λ simboliza la cadena vacía

Dificultades encontradas en el desarrollo del TP

La primer complicación encontrada recae en lo más básico del desarrollo de este trabajo práctico, en cómo lograr un gramática para generar música. Se requirieron varias horas de pensamiento y debate para llegar a una solución con la cual todos los integrantes se sintieron satisfechos y pudieran visualizar la construcción de dicha gramática, ya que siempre se estuvo de acuerdo con el objetivo pero la forma resultó una constante incógnita por mucho tiempo. La mayoría de los problemas encontrados se solucionaron al momento en que se tomó la decisión de reducir la gramática a la composición de melodías simples a partir de notas musicales, y abandonar el enfoque inicial que se había tomado, ya que resultaba muy complejo y de difícil realización.

Una vez que se definió por un proyecto más simple, se planteó la siguiente gramática:

```
S -> N / N | N
N -> HXT
H -> C | D | E | F |G |A |B
X -> M |#M | bM | λ
M -> - | M | ° | dim | Ø | m7b5 | maj7 | M7 | Δ | aug | + | #5 | 5+ | λ
T -> 5M' |6M' |  7M' | 9M' | 11M' | 13M' | sus2 | sus4
M' -> sus2 | sus4 | # | b | + | λ
```

Este tipo de gramática nos permitía realizar cualquier tipo de nota musical, dando lugar a una gran complejidad en su manejo, pero se terminó por simplificar aún más el proyecto y se lo llevó a unas pocas notas musicales que se pueden transformar fácilmente en frecuencias para ser reproducidas y reconocidas como música.

Otro problema, y seguramente el mayor de ellos, fue inmediatamente después de haber construído la gramática a utilizar. Al momento de compilar la gramática, tanto YACC como LEX generaban muchísimos errores, y hubo que analizarlos y solucionarlos uno por uno. Los más complicados eran errores de *reducción/reducción* (*reduce/reduce*) y de *reducción/desplazamiento* (*reduce/shifting*), los cuales se terminaron reparando al hacerle las modificaciones necesarias a la gramática construída. Una vez solucionada esta complicación, otro problema que surgió al momento de *parsear* un archivo de ejemplo, debido a que el archivo de salida del compilador desarrollado se encontraba vacío, por lo que luego GCC no tenía nada para interpretar. Una vez superados estos obstáculos y que se haya terminar de pulir los pequeños errores e incongruencias encontrados a cada avance que se logró, finalmente se pudieron reproducir todas las melodías deseadas.

Futuras extensiones

Existen varias formas de extender esta gramática para poder ampliar el uso que se le puede dar, con el fin de dar la posibilidad de generar composiciones musicales más complejas, pero a la vez más atractivas.

La idea inmediata y probablemente de implementación más sencilla sería permitir que se le asigne una duración a cada nota musical, permitiendo a quien componga la canción que cada nota suene por la cantidad de tiempo que él desee. La complejidad de esta extensión debería ser reducida, ya que una forma de implementarlo sería dictando la duración de cada nota a continuación de la nota misma en la secuencia de notas *music*. Por ejemplo: *“A30B80C30”*, lo que se interpretaría como *“tocar la nota A por 30 unidades de tiempo, luego la nota B por 80, seguido de la nota C por 30”*, donde las unidades de tiempo vendrían dadas por default y serían siempre las mismas, como ser milisegundos o centésimas de segundos.

Otra idea que surgió, más como un deseo y como una curiosidad interesante, fue la de posibilitar varias pistas de melodías. Lo que esto significa es poder hacer que suenen varias melodías en simultáneo, componiendo las mismas todas de la misma manera que la actual, pero pudiendo elegir en qué momento arranca cada secuencia musical y que suene más de una nota al mismo tiempo. Lamentablemente no surgió una solución a esta idea ya que la complejidad de la misma parece ser bastante elevada.

Seguramente haya varias ideas de fácil implementación que no fueron presentadas por falta tiempo o una falta de entendimiento en el tema, ya que resultó todo un desafío pensar y lograr desarrollar una gramática para componer música.

Referencias

Para la creación de archivos .wav se utilizó una biblioteca encontrada en internet, realizada por Douglas Thain (http://www.nd.edu/~dthain/courses/cse20211/fall2013/wavfile).

Otras páginas consultadas son las siguientes:

- http://www.tldp.org/HOWTO/Lex-YACC-HOWTO-4.html
- http://www.isi.edu/~pedro/Teaching/CSCI565-Spring11/Lectures/SyntacticAnalysis.Yacc.6p.pdf
- http://web.mit.edu/gnu/doc/html/bison_9.html
- https://books.google.com.ar/books?id=F3lWLs1iTAMC&pg=PA52&lpg=PA52&dq=yacc+y+lex+manejo+de+errores&source=bl&ots=5mCXhasiyq&sig=SJbv-8ZeLNQVcx-GMqAJRkPkV24&hl=es&sa=X&ved=0ahUKEwj91Y6AzbbJAhURmpAKHXXwAXcQ6AEIKTAC#v=onepage&q=yacc%20y%20lex%20manejo%20de%20errores&f=false
- http://www-h.eng.cam.ac.uk/help/tpl/languages/flexbison/
- https://www.ultimate-guitar.com/