Code Code Revolution

Trabajo Práctico
Autómatas, Teoría de Lenguajes y Compiladores

El Buen Libro

Fernán Oviedo
Axel Fratoni
Gastón Rodríguez

2do cuatrimestre 2017

Índice

Idea y Objetivo

En este trabajo práctico se llevó a cabo el desarrollo de un lenguaje de programación llamado **Code Code Revolution** (CCR). El lenguaje no aspira a resolver ninguna problemática de los lenguajes de programación modernos. Su objetivo es contribuir al conjunto de lenguajes esotéricos y demostrar una vez más la articulación efectiva de las herramientas tradicionales para la generación de compiladores.

La idea del lenguaje fue el producto de la necesidad y de las ganas de innovar con una mezcla de paradigma imperativo y declarativo. Básicamente se trata de simular a una persona que se desplaza por una grilla y ejecuta instrucciones a medida que se mueve. La reminiscencia de esta idea con el famoso juego *Dance Dance Revolution* es lo que le da nombre al lenguaje.

Consideraciones

La consigna detalla que se debe entregar un compilador de lenguaje creado. Por cuestiones de modularización y reusabilidad, decidimos entregar dos compiladores, ambos necesarios para conseguir una versión ejecutable del código. Una descripción detallada de los motivos de esta decisión se encuentra en la próxima sección.

Desarrollo

El desarrollo del trabajo no sólo implicó el diseño de un lenguaje, sino también la realización de los compiladores que permitieran transformar programas en este lenguaje a código ejecutable directa o indirectamente. Estos compiladores se crearon utilizando *Lex* y *Yacc*, las dos herramientas tradicionales para generar analizadores léxicos y sintácticos respectivamente.

Estructura general

La idea detrás de todo programa en este lenguaje es la de un cursor que se desplaza por un tablero rectangular. En cada posición de este tablero, se encuentra una instrucción que el cursor puede ejecutar si se posiciona sobre ella. Así, un programa en *CCR* se compone de dos archivos, uno que define el tablero y otro que detalla los movimientos del cursor por el tablero.

Tablero

La definición del tablero implica especificar su tamaño y luego las instrucciones que irán en cada posición, pudiendo quedar posiciones vacías. Para ello, se definió una sintaxis y un conjunto de instrucciones (ver el *readme* del proyecto). Las posiciones en el tablero se indexan desde el par (1,1) hasta el par (N,M), siendo N x M el tamaño del tablero. Es un error indexar un casillero por fuera del tablero.

Movimientos

La definición de los movimientos no es más que una cadena de caracteres, cada uno de los cuales corresponde a un movimiento del cursor. Existen cinco movimientos posibles:

- U: mueve el cursor un lugar hacia arriba, es decir, de la posición (x,y) a la posición (x-1,y)
- D: mueve el cursor un lugar hacia abajo, es decir, de la posición (x,y) a la posición (x+1,y)
- L: mueve el cursor un lugar hacia la izquierda, es decir, de la posición (x,y) a la posición (x,y-1)
- R: mueve el cursor un lugar hacia la derecha, es decir, de la posición (x,y) a la posición (x,y+1)
- J: ejecuta la instrucción sobre la que se encuentra el cursor; si no hay instrucción en esa posición, no hace nada

En cualquier caso, es un error mover al cursor fuera del tablero.

Entorno

Además del tablero, hay otros dos factores de entorno a tener en cuenta: los *buckets* de números y los *buckets* de función.

En todo programa se encuentra definido un arreglo de 1024 *buckets* que permiten alojar valores enteros con signo de 4 bytes. El valor inicial de estos buckets es 0. Además, hay un único puntero a bucket que indica cuál es el bucket que se está mirando actualmente. Este puntero arranca apuntando al primer *bucket*. Los *buckets* se indexan desde 0 hasta 1023. La utilización de los *buckets* se detalla en el conjunto de instrucciones.

Los *buckets* de función funcionan de forma similar, pero contienen instrucciones para el cursor en lugar de números. A diferencia de los *buckets* numéricos, la definición de los elementos de estos *buckets* se hace en el código de los movimientos del cursor.

Comentarios

Para favorecer la legibilidad del código, se permite utilizar comentarios unilínea estilo C.

Etiquetas

Para poder hacer saltos a partes del código del cursor, se permite utilizar etiquetas en medio de los movimientos. Éstas se usan con una instrucción particular del conjunto de instrucciones.

Compilación

Para poder compilar el código, se dispone de dos compiladores: uno que compila el tablero y otro que compila los movimientos del cursor. Cada uno de éstos genera un archivo fuente en C. Al compilar estos archivos en C (por ejemplo, con *GCC*), se obtiene un ejecutable del programa especificado en CCR. En la generación de los compiladores también se genera una librería estática que debe ser utilizada en la compilación de los archivos fuentes de C. Para una descripción más detallada del proceso de compilación, véase el *readme* del proyecto.

La idea de tener dos compiladores es la de poder compilar por un lado tableros y por el otro movimientos del cursor. De esta forma, se permite la reutilización de tableros y movimientos sin necesidad de recompilar los respectivos archivos.

Gramática

Para una descripción de la gramática del lenguaje, véase el *readme* del proyecto.

Dificultades

La principal dificultad estuvo en alcanzar la *Turing-completitud*. Básicamente, cuanto más esotérico se quería hacer, más costaba hacer al lenguaje *Turing-completo* (o incluso poder decidir si lo era). El lenguaje en estas condiciones es el resultado del balance más adecuado que se logró.

Futuras extensiones

Entrada y salida

De todo compilador, es deseable que, en caso de haber un error en el código fuente, no produzcan ningún archivo con el código compilado. Siendo que los compiladores que desarrollamos leen el programa fuente por entrada estándar y escriben el programa compilado por salida estándar, no conocen la noción de dejar o no dejar un archivo. Hay dos formas de saber si la compilación fue exitosa o no. Una es mirar la salida de error del compilador, la cual muestra un mensaje de error si se encontró algún problema con el archivo fuente. La otra es mirar el valor de retorno (como siempre, 0 significa éxito, distinto de 0 significa error). Sería deseable que en una próxima versión se le dé al compilador el nombre del archivo que se desea generar por línea de comando y que no lo cree (o lo cree y lo borre) si hay algún error en la compilación. Tambień sería deseable que el archivo fuente no llegue por la entrada estándar, sino también por la línea de comandos.

Múltiples fuentes de movimientos

Siendo que el lenguaje permite definir funciones y almacenarlas en *buckets*, uno podría querer tener uno o más archivos que sólo contengan definiciones de funciones para *buckets* y uno solo que contenga la serie de movimientos que debe realizar el cursor (análogo al *main* en C). De esta forma, se podría compilar cada uno y linkeditarlos posteriormente. Esta es una funcionalidad que los compiladores desarrollados no ofrecen.