Autómatas, Teoría de Lenguajes y Compiladores

Primer Cuatrimestre 2018

Trabajo Práctico Especial

Profesores

Arias Roig, Ana María

Ramele, Rodrigo Ezequiel

Santos, Juan Miguel

Grupo The Roxys

Della Sala, Rocío 56507

Giorgi, María Florencia 56239

Rodríguez, Ariel Andrés 56030

Santoflaminio, Alejandro 57042

Índice

Idea subyacente y objetivo del lenguaje

En el siguiente informe se describirá el desarrollo del lenguaje de programación Mate. El nombre proviene de la infusión, muy popular en la Argentina, dado que la idea de Mate es permitir a los programadores escribir código en lenguaje coloquial argentino haciendo uso de varios términos de la jerga propia de este país.

La idea es proporcionar un lenguaje intuitivo y fácil de usar con funciones básicas para programadores argentinos o para quienes hagan uso del acento argentino.

Consideraciones realizadas

El analizador sintáctico está programado en C y genera su salida también en lenguaje C. Se intentaron incorporar la mayor cantidad de operadores y bloques posibles, así como también distintos tipos de variables los cuales se detallarán más adelante.

El lenguaje, además de tener expresiones argentinas, también tiene mucha semejanza con el lenguaje de programación C, ya que hay ciertos aspectos de su sintaxis que son similares y que decidimos dejar así porque cambiarlo sería aún más confuso para quien desee programar en Mate. Es importante destacar que no se hace uso de tildes en el lenguaje para facilitar su escritura.

Descripción del desarrollo

Se hizo uso de la herramienta Lex para definir la gramática mediante el uso de tokens generando de esta forma los símbolos que integran el lenguaje. Con Yacc, se leen los símbolos que le proporciona lex y se genera luego el analizador sintáctico, es decir las reglas que permiten definir a un programa como parte del lenguaje Mate. Cuando compilamos el programa con Mate generamos el código en .c de un archivo previamente en .arg (formato utilizado por Mate). Luego en el *makefile* se compila ese programa en código C y se lo ejecuta.

Se encuentran incluidos los siguientes programas a modo de ejemplo para probar y poder visualizar la sintaxis de `Mate`:

- `Tateti`: Se incluye el clásico juego como test de prueba.
- `mayor`: Muestra el mayor número de un arreglo de números estático.
- `mayor_2`: Muestra el mayor número de un arreglo de números que ingresa el usuario.
- `conversor`: Permite pasar cierto monto de pesos a moneda extranjera y viceversa utilizando la cotización de la fecha de entrega del trabajo.
- `el_diego`: Cuenta números del 0 hasta el 9, nombrando jugadores correspondientes a tal número de camiseta, para finalmente mostrar un mensaje indicando que encontró al mejor de todos.
- `calculadora`: Permite realizar diferentes operaciones entre dos números ingresados por entrada estándar.
- `factorial`: Calcula el factorial de un número ingresado por entrada estándar.

Descripción de la gramática

Para indicar el comienzo y el fin de un programa se hace uso de `buenas` y `ni nos vimo` delimitando así el código interno del programa.

La gramática admite distintos tipos de datos:

- El tipo `entero` que vendría a ser equivalente a int en C.
- El tipo `con coma` que vendría a ser equivalente a float en C.
- El tipo `letra` que vendría a ser equivalente a char en C.
- El tipo `frase` que vendría a ser el equivalente al tipo String en lenguajes como Java.
- También existen arreglos de enteros y de frases llamados `mix de números` y `mix de frases` respectivamente.

Se presentan los siguientes tipos de bloques condicionales encontrados en el lenguaje C, así como también en la mayoría de los lenguajes de programación (todos se asemejan a C en cuanto a uso de paréntesis y llaves):

- Bloques *if* renombrado `chequeame si` y *else* renombrado `sino`.
- Bloques *do-while* llamados `metele-siempre que`, así como *while* solos (`siempre que`).
- Ciclos *for* llamados `vengan de a uno empezando por`. Estos ciclos solo admiten el decremento o incremento de una variable de a uno.
- Bloques *switch* llamados `depende de lo que venga en` así como los operadores llamados `si es un` (*case*), `si no queda otra` (*default*) y `basta chicos` (*break*).

Existen también funciones de impresión (`tirame la posta`) y de escaneo (`leete algo`).

Para asignar un valor a una variable en vez de utilizar el símbolo “=” se utiliza la palabra `sos`. De esta forma queda, por ejemplo: `i sos 5` para asignarle el valor 5 a la variable de tipo entero i. Para terminar toda instrucción se hace uso del ‘;’ al igual que en la mayoría de los lenguajes que utilizamos.

En cuanto a las variables booleanas *false* y *true* sus equivalentes son `chamuyo` y `la posta`.

Los operadores lógicos con los que cuenta `Mate` son:

- `o` (*or*)
- `y` (*and*)
- `ni ahi` (*no*)

Por otro lado, los operadores aritméticos son:

- `mas`
- `menos`

- `dividido`
- `por`
- `modulo`
- `aplicar tarifazo` (incrementar en uno)
- `devaluar` (decrementar en uno)

Por último, los operadores relacionales fueron determinados de la siguiente manera:

- `es menos piola que` (<)
- `es mas piola que` (>)
- `es` (=)
- `nada que ver con` (!=)
- `es o es menos piola que` (<=)
- `es o es mas piola que` (>=)

También existen constantes determinadas las cuales son:

- `el Diego` es equivalente a ingresar el número 10.
- `un palo` es equivalente a ingresar 1000000.
- `una luca` es equivalente a ingresar 1000.
- `una gamba` es equivalente a 100.

En la carpeta del proyecto se encuentra el archivo BNF que describe con mayor precisión la sintaxis del lenguaje.

Dificultades encontradas

En un primer lugar nos hubiera gustado implementar un árbol de parsing (AST) pero se nos hizo muy complejo por lo cuál decidimos dejarlo de lado.

Al compilar el parser llamado mate.y la terminal mostraba un warning *warning: 1 shift/reduce conflict [-Wconflicts-sr]*. Resulta que había reglas redundantes en el parser, ya que había producciones escritas distintas que terminaban llevando a lo mismo.

Futuras extensiones

Se podrían agregar ciclos `for` que permitan que el incremento o decremento sea mayor a uno, ya que por como está ahora, el lenguaje solo permite aplicarle a la variable un incremento o decremento y no una asignación.

Además, al ser un lenguaje que utiliza expresiones coloquiales podrían definirse más tokens que mapeen a un mismo símbolo, porque puede que el programador se sienta más familiarizado con ciertos términos (por ejemplo, podría inicializarse un programa con ‘buenas’ así como también con ‘hola’). Esto le daría cierta libertad al usuario para decidir que términos desea usar.

También hubiera sido deseable implementar comentarios en nuestro lenguaje, lo que podría llegar a ser logrado muy fácilmente logrando algo similar a lo que se hace en C y en otros lenguajes como Java *(/* comentario */).*

Una adición fácil que se podría agregar a nuestro lenguaje serían los arreglos de números con coma (*floats*), ya que podría ser implementado de manera similar a los demás arreglos.

Por último, consideramos que el próximo paso podría ser agregar la posibilidad de definir matrices utilizando arreglos, de la misma manera en la que se hace en C.

Referencias

- *https://github.com/faturita/YetAnotherCompilerClass* Lex and Yacc samples and tools repository for Languages and Compiler class
- *http://matt.might.net/articles/grammars-bnf-ebnf/* The language of languages
- *https://www.youtube.com/watch?v=54bo1qaHAfk* Part 01: Tutorial on lex/yacc
- *https://www.youtube.com/watch?v=__-wUHG2rfM* Part 02: Tutorial on lex/yacc.