instituto tecnológico de veracruz

Ensayo Final: Ruby

Catedrático: Martha Martínez Moreno

Por: Ernesto Álvarez Calderón
Jazmín Arlette Carrasco Cruz
Ricardo David Rojas Flores

ENSAYO DE AUTOMÁTAS II

Presentación

Una experiencia en general sobre Ruby.

Ruby es un lenguaje de programación ciertamente interesante, nuestra predilección por dicho lenguaje si bien, fue en un momento por haber sido objetivo de nuestro trabajo previo en la asignatura de **Lenguajes y Autómatas I** (a partir de ahora, LyA I), terminó asentándose como uno de los tantos lenguajes de programación que nos haría bien aprender en la actualidad, debido a sus peculiaridades como su dinamismo y escalabilidad.

Ruby está creciendo en el campo de la programación web, presente en frameworks trabajando en el ***backend*** como **Ruby on Rails** o simplemente en su versión más pura en diferentes ambientes donde la programación esté sometida dinamismo. Dinamismo, dinámico —el adjetivo que define la esencia de Ruby, y la raíz de nuestro estudio.

En la asignatura de **Lenguajes y Autómatas II**, se buscó dar continuidad al proyecto de LyA I, el cual consistió en la creación de un analizador léxico-sintáctico. Para realizar esto se tuvo que partir de la investigación de las unidades léxicas (también conocidas como tokens), continuar construyendo reglas gramaticales las cuales definieran a dichas unidades léxicas —dando luz a un analizador léxico. Posteriormente, siguió la creación de árboles sintácticos que retroalimentarían al analizador sintáctico que a su vez se basaba en el analizador léxico—. Fue hasta entonces que se decidió continuar con las siguientes fases en la creación de un compilador.

1. **Análisis Léxico**
2. **Análisis Sintáctico**
3. **Análisis Semántico <-*Inicio de LyA II***
4. **Generación de Código Intermedio**
5. **Optimización**
6. **Generación de Código Objeto**

Desarrollo

Análisis Léxico

En esta etapa tuvimos que realizar una investigación para obtener el **BNF** del lenguaje Ruby. En este se encontraban también las palabras reservadas del lenguaje, con muchas en común en comparación con otros lenguajes. En el aspecto de la programación fue le etapa más sencilla. Para este proyecto se utilizó **JavaCC**, un generador de analizador léxico/sintáctico, el cual facilitó esta tarea. Definimos nuestras unidades léxicas en nuestro archivo .jj y se hizo una interfaz gráfica.

Analizador Sintáctico

Para esta etapa, ya el código ingresado pasa las pruebas léxicas, es decir, que las palabras utilizadas formante parte del lenguaje.

La etapa sintáctica por su parte, buscó el poder detectar **inconsistencias gramaticales** durante la escritura de un programa. Es decir, que el orden de las palabras y metasímbolos concordaran con las reglas previamente establecidas. Para esto, editamos nuestro .jj de JavaCC y actualizamos nuestra interfaz. Cabe recalcar que este aspecto la mayoría de los lenguajes deben ser estrictos.

Analizador Semántico

Esta etapa fue honestamente la más complicada en términos de programación para nosotros. Ruby es un lenguaje muy **dinámico,** que permite cosas como omitir el tipo de dato y ser capaz de operar con valores multitipos. Es un lenguaje orientado a objetos que trata todo dato como un objeto, independientemente de su tipo por lo que dichos objetos pueden incluso tomar muchas otras formas diferentes.

Para realizar esta etapa seguimos empleando JavaCC, elegimos una **validación semántica** de las varias disponible (**operaciones aritméticas, asignación, entrada/salida, etc**.) Durante esta etapa realizamos uso intensivo de **tablas Hash**, estructuras de datos que se utilizan para almacenar un número elevado de datos sobre los que se necesitan operaciones de búsqueda e inserción muy eficientes. Una tabla hash almacena un conjunto de pares "(clave, valor)". La clave es única para cada elemento de la tabla y es el dato que se utiliza para buscar un determinado valor.

Nuestro equipo de trabajo optó por **la validación semántica de operaciones aritméticas**, debido al poco margen de escrutinio que nos suponía optar por la validación de asignación, como la dinámica de Ruby previamente mencionada implica al programador.

A raíz de esto iniciamos nuestra investigación de la mayoría de los errores generados con los operadores de **suma, sustracción, multiplicación y división**. La mayoría resultaron ser errores de tipo, como por ejemplo no poder dividr un número por una cadena, sin embargo hubo casos excepcionales como el poder multiplicar una cadena, lo cual retornaba esa misma cadena repetida n veces. Si bien Ruby es liberal al instante de inicializar, resultó quisquilloso al punto de requerir **conversiones implícitas**, es decir, indicar por medio de un método la conversión que permitiese realizar una acción determinada, a diferencia de otros lenguajes que automáticamente realizan la conversión sin alertar al programador.

Generación de Código Intermedio

La etapa de generación de código intermedio es aquella en la cual el programa semánticamente validado pasa a través de una **serie de transformaciones**, muy frecuentemente por una conversión de código de alto nivel a uno de bajo nivel, como **ensamblador,** a fin de ser interpretadas por el procesador. Para nuestro proyecto realizamos una conversión de la **estructura de control while** en Ruby a ensamblador y **Jasmin** (un ensamblador para Java), logrando así visionar la complejidad de las operaciones que debe ejecutar el microprocesador. Esta etapa fue algo difícil debido a la escasa información de Ruby en internet, si bien existe documentación en inglés, no hay muchos ejemplos que puedan servir de introducción a principiantes como nosotros, hubo que dedicar más tiempo del previsto a esta fase.

Optimización

Esta etapa le precedió a la generación de código intermedio, en ésta, se buscó **optimizar el código de Ruby** por medio de diferentes métodos. En esta asignatura se implementó una **calculadora,** lo cual fe interesante, ya que Ruby se caracteriza por ser veloz al trabajar con números enormes. También se midieron los tiempos de ejecución tras las optimizaciones, en Ruby, esto se logra por medio de la clase **Benchmark**. Esta clase contiene diferentes métodos para evaluar los tiempos de ejecución de cualquier bloque de código de Ruby. Algunos de ellos contrastan los tiempos del CPU, del usuario, real, total, etc.

Generación de Código Objeto

En nuestro caso, Ruby no genera código objeto, ya que es un lenguaje interpretado. El código se puede escribir en cualquier editor de texto siempre y cuando se guarde con la **extensión .rb**, para después ser ejecutado a través de la línea de comandos de Ruby de la siguiente manera: ***ruby programa.rb***. Por esta misma razón, su ejecución requiere de la instalación de Ruby, el cual está disponible para Linux, Microsoft, Mac OS, etc.