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 fue creado en Japón por Yukihiro Matsumoto mientras trabajaba como programador con lenguajes como Perl y PHP.

Según el creador de Ruby, el nombre del lenguaje lo decidió en honor a un colega suyo a partir de la piedra correspondiente a su mes de nacimiento. También se dice que es un juego de palabras relacionado con el lenguaje Perl, ya que en un principio su idea era solo crear una versión mejorada del lenguaje. En un principio, su intención fue crear Perl avanzado, para mejorar algunos detalles de dicho lenguaje, pero en lugar de mejorarlo, se vio tentado en crear uno propio a partir de sus lenguajes favoritos: Perl, Smalltalk, Eiffel y Lisp."

Características

- *Orientado a objetos.*
- *Cuatro niveles de ámbito de variable: global, clase, instancia y local.*
- *Manejo de excepciones.*
- *Iteradores y clausuras o closures (pasando bloques de código).*
- *Expresiones regulares nativas similares a las de Perl a nivel del lenguaje.*
- *Posibilidad de redefinir los operadores (sobrecarga de operadores).*
- *Recolección de basura automática.*
- *Altamente portable.*
- *Hilos de ejecución simultáneos en todas las plataformas usando hilos verdes, o no gestionados por el sistema operativo.*
- *Carga dinámica de DLL/bibliotecas compartidas en la mayoría de las plataformas.*
- *Introspección, reflexión y metaprogramación.*
- *Amplia librería estándar.*
- *Soporta inyección de dependencias.*
- *Soporta alteración de objetos en tiempo de ejecución.*

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.

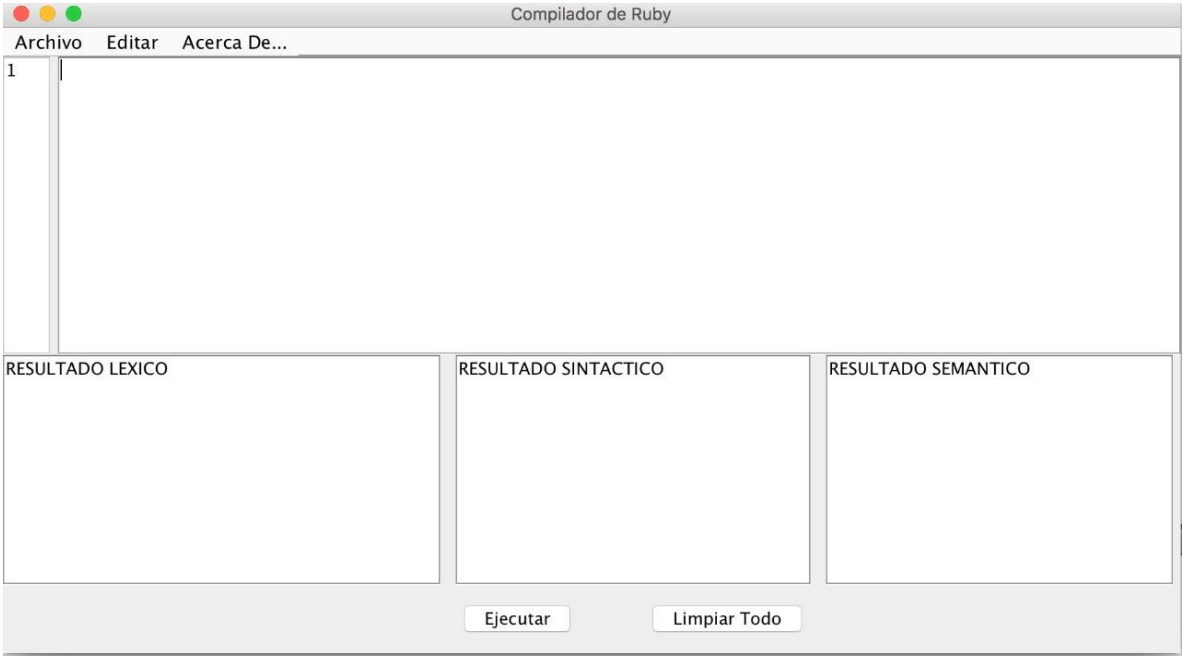
En la asignatura de **Lenguajes y Autómatas II**, se buscó dar continuidad al proyecto de LyA I, el cual consistió en la creación de un analizador léxico-sintáctico. Para realizar esto se tuvo que partir de la investigación de las unidades léxicas (también conocidas como tokens), continuar construyendo 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

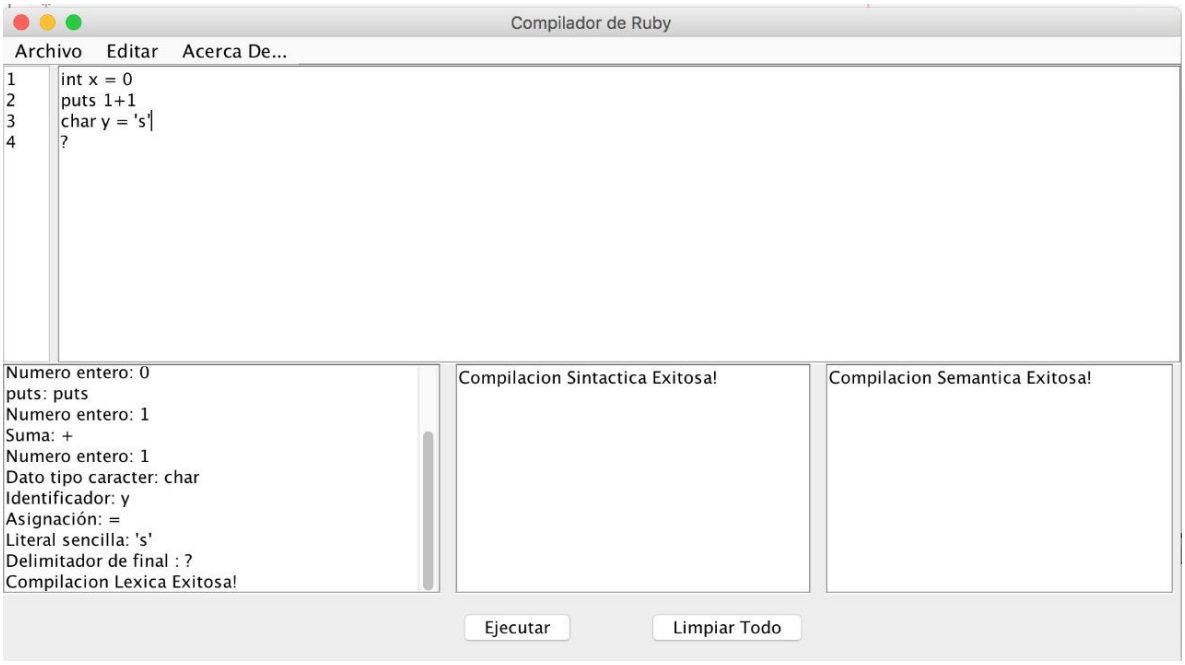
En esta etapa tuvimos que realizar una investigación para obtener el **BNF** del lenguaje Ruby. En este se encontraban también las palabras reservadas del lenguaje, con muchas en común en comparación con otros lenguajes. En el aspecto de la programación fue le etapa más sencilla. Para este proyecto se 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.



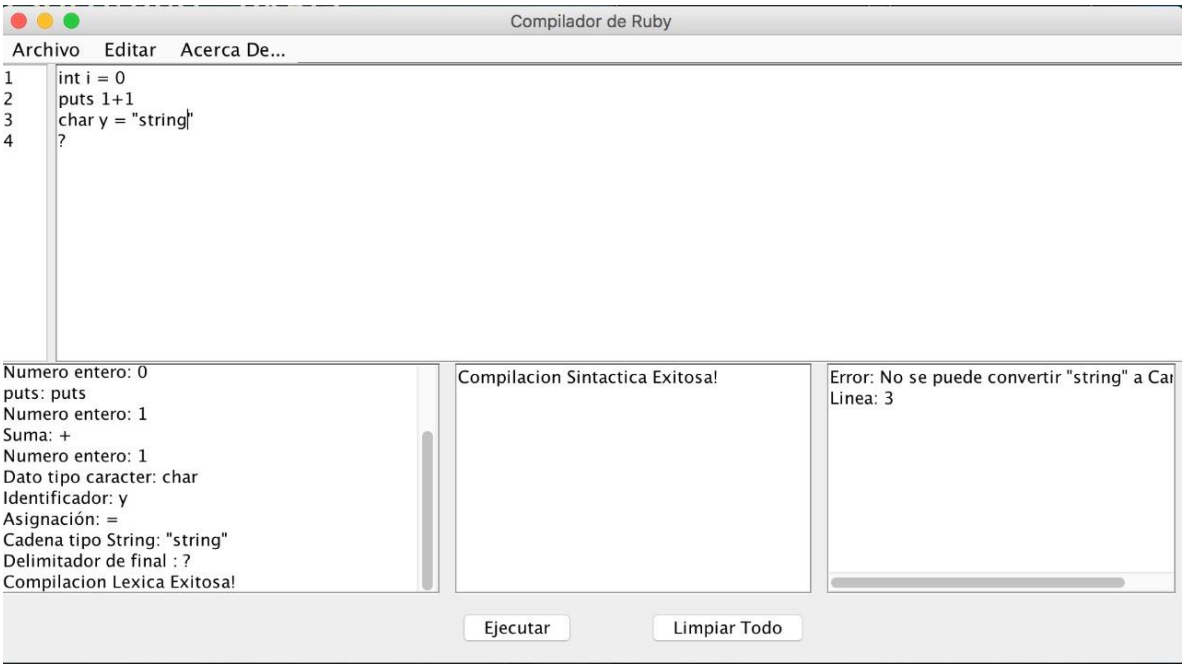
Analizador Semántico

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.

Código en Ensamblador

```
CS:0100 mov AX,5

CS:0102 mov BX,0

CS:0104 JE CS:0108 [Cambio a otra ubicacion en el segmento de codigo si
la bandera Zero esta encendida]

CS:0105 DEC AX [Decremento de AX]

[CODIGO INTERMEDIO]

CS:0107 JMP CS:0104

CS:0108 ... [Codigo en adelante despues del while]
```

Código en Jasmin

```
0100 mov AX,5

0102 mov BX,0

0104 [label1] 0108
```

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.

```ruby
require "benchmark"

time = Benchmark.realtime do
  (1..10000).each { |i| i }
end
puts "Time elapsed #{time*1000} milliseconds"
```

```
CALC3
require 'benchmark'


puts "////////////////////////////RUBY
CALCULATOR//////////////////////////"
puts "Ingresa una expresión"
es= gets.chomp

t1 = Benchmark.realtime do
nums=es.scan /^\(\d+[\+\-\*\/]+\d+\)[\+\-\*\/]+\d+/


puts "El resultado de #{nums} es: "+eval(es).to_s


end
puts
"////////////////////////////////////////////////////////////////////////"
puts "------------------------------------------------"

puts "La ejecución tomó "+(t1*1000).to_s+" milisegundos"
```

Tiempos de ejecución:

Programa	Tiempo de ejecución en tiempo real
Calc3	0.95 ms
Calc2	0.93 ms
Calc1	3.42 ms

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.

Glosario

Analizador Léxico: es la primera fase de un compilador consistente en un programa que recibe como entrada el código fuente de otro programa (secuencia de caracteres) y produce una salida compuesta de tokens (componentes léxicos) o símbolos.

Analizador Sintáctico: es un programa informático que analiza una cadena de símbolos de acuerdo a las reglas de una gramática formal. El término proviene del Latín pars, que significa parte (del discurso). Usualmente hace parte de un compilador, en cuyo caso, transforma una entrada en un árbol de sintáctico de derivación.

Analizador Semántico: es una fase de análisis semántico del programa fuente para tratar de encontrar errores semánticos y reúne la información sobre los tipos para la fase posterior de generación de código. En ella se utiliza la estructura jerárquica determinada por la clase de análisis sintáctico para identificar los operadores y operandos de expresiones y proposiciones.

Backend: el Back-End es el área que se dedica a la parte lógica de un sitio web, es el encargado de que todo funcione como debería, el back-end es la parte de atrás que de alguna manera no es visible para el usuario ya que no se trata de diseño, o elementos gráficos, se trata de programar las funciones que tendrá un sitio. El Back-End es la programación dura y pura, desde la programación de las funciones del sitio hasta bases de datos e incluso más.

Benchmark: también denominada prueba de rendimiento, es una situación a la cual se somete un programa en base a ciertos parámetros, por lo regular en relación de tiempo-recurso.

BNF: también conocida por sus denominaciones inglesas Backus-Naur form (BNF), Backus-Naur formalism o Backus normal form, es un metalenguaje usado para expresar gramáticas libres de contexto: es decir, una manera formal de describir lenguajes formales.

Código Intermedio: el código intermedio es un código abstracto independiente de la máquina para la que se generará el código objeto. El código intermedio ha de cumplir dos requisitos importantes: ser fácil de producir a partir del análisis sintáctico, y ser fácil de traducir al lenguaje objeto.

Código Objeto: en programación, se llama código objeto al código que resulta de la compilación del código fuente. Puede ser en lenguaje máquina o bytecode, y puede distribuirse en varios archivos que corresponden a cada código fuente compilado.

Conversión Implícita: se le denomina así al método por el cual se especifica la conversión de un tipo de dato a otro.

Estructura de control: es un bloque de código el cual cumple que cuenta con un inicio, desarrollo y fin bien establecidos.

Framework: es un marco de trabajo el cual puede estar compuesto de varias tecnologías que puede ser utilizado para diferentes fines.

JavaCC: es una librería hecha en Java para programar analizadores léxicos y sintácticos.

Lenguaje Dinámico: es un lenguaje que permite cierta flexibilidad en momentos precisos tales como asignación, conversión de datos, etc.

Optimización: es la etapa en la cual se busca reducir líneas u operaciones en un código fuente semánticamente válida para reducir tiempos de ejecución y recursos.

Perl: Perl es un lenguaje de programación diseñado por Larry Wall en 1987. Perl toma características del lenguaje C, del lenguaje interpretado bourne shell, AWK, sed, Lisp y, en un grado inferior, de muchos otros lenguajes de programación

PHP: PHP es un lenguaje de programación de propósito general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico.

Ruby on Rails: es un entorno de desarrollo web de código abierto que está optimizado para la satisfacción de los programadores y para la productividad sostenible. Te permite escribir un buen código evitando que te repitas y favoreciendo la convención antes que la configuración.

Tabla 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.

Unidad léxica: también conocida como token, es la unidad mínima de una gramática.

Tras haber concluido este curso de manera exitosa, si bien con sus altibajos como todo proyecto en la vida, el resultado final nos fue grato. En un futuro nos gustaría terminar este proyecto como se debe, aunque esto pueda llegar a requerir de mucho, mucho tiempo.

Explorar y estudiar cada etapa o fase en la creación de un compilador amplió nuestro panorama a la complejidad de algo que nos es tan común, pero que detrás guarda horas de esfuerzo y horas de vela, fruto del trabajo colectivo de aquellos que nos precedieron.

Hoy, más que nunca apreciamos su arduo trabajo, que en algún momento de su desarrollo debió haber pasado por todas o algunas de las fases descritas en este trabajo. A ellos, no nos queda más que decir: Gracias.

Bibliografía

FalconMasters. (2014). ¿Qué es front-end y back-end?. 28 de octubre del 2017, de FalconMasters Sitio web: http://www.falconmasters.com/web-design/que-es-front-end-y-que-es-back-end/

JavaCC. (2017). The Java Parser Generator. 22 de agosto del 2017, de JavaCC Sitio web: https://javacc.org/

Matz, hablando en la lista de correo Ruby-Talk, 12 de mayo del 2000

Matz, en An Interview with the Creator of Ruby, 29 de noviembre del 2001

Matz, en Blocks and Closures in Ruby, 22 de diciembre del 2003

tlazaro. (2017). NetBeans. 22 de agosto del 2017, de NetBeans Sitio web: http://plugins.netbeans.org/plugin/20277/javacc

UAEH. (2014). Análisis Semántico. 28 de octubre del 2017, de UAEH Sitio web: http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/autocontenido/autocon/134_anlisis_semntico.html

Wikipedia. (2014). Analizador Léxico. 28 de octubre del 2017, de Wikipedia Sitio web: https://es.wikipedia.org/wiki/Analizador_l%C3%A9xico

Wikipedia. (2014). Analizador Sintáctico. 28 de octubre del 2017, de Wikipedia Sitio web: https://es.wikipedia.org/wiki/Analizador_sint%C3%A9tico

Wikipedia. (2014). Código Objeto. 28 de octubre del 2017, de Wikipedia Sitio web: https://es.wikipedia.org/wiki/C%C3%B3digo_objeto

Wikipedia. (2014). Notación Backus-Naur. 28 de octubre del 2017, de Wikipedia Sitio web: https://es.wikipedia.org/wiki/Notaci%C3%B3n_de_Backus-Naur

Wikipedia. (2014). PHP. 28 de octubre del 2017, de Wikipedia Sitio web: https://es.wikipedia.org/wiki/PHP