Figure 1: El grupo observando a su amado compilador "Scrappy".

Informe Proyecto Especial
Autómatas, Teoría de Lenguajes y Compiladores
72.39

Grupo 7 - Scrappy

Iñaki Bengolea (63515), ibengolea@itba.edu.ar
Joaquín Eduardo Girod (63512), jgirod@itba.edu.ar
Christian Tomás Ijjas (63555), cijjas@itba.edu.ar
Felix Lopez Menardi (62707), flopezmenardi@itba.edu.ar

22 de Noviembre de 2023

Contents

1 Introducción

Para el presente Proyecto Especial de la materia Autómatas, Teoría de Lenguajes y Compiladores el grupo desarrollo un Lenguaje que permitiera simplificar y acerca a un público más general el web scraping, con todos los beneficios que se pueden obtener del mismo.

Se partió de una idea principal –presentada previamente a la cátedra– y con un manojo de cambios se llegó a la versión final de este lenguaje, con el cual estamos más que satisfechos y esperamos que sea del agrado de todo aquel que lo use.

A grosso modo, el lenguaje busca agilizar operaciones simples como el scraping de de diferentes elementos de páginas web identificados por su tags. A su vez es posible definir variables la cuales más adelante serán interpoladas por sus respectivos valores, para evitarle redundancias al usuario. Se permite también ingresar usuario y contraseña para sitios que requieren de autenticación. A continuación se ahondara en los diferentes aspectos del lenguaje, el compilador y su desarrollo.

2 Consideraciones Adicionales

Se incluye una breve guía para ejecutar los archivos generados por el compilador, también incluida en el archivo README.md del repositorio.

Primero será necesario descargar e instalar Node.js desde su sitio oficial.

Luego, desde el directorio en el cual se encuentra el archivo generado:

```
>>  npm init -y
```

En caso de no haber sido descargada previamente la librería de Puppeteer:

```
>>  npm install puppeteer
```

Finalmente, para correr el script archivo generado:

```
>>  node scrappy.js
```

También cabe frenar brevemente sobre el funcionamiento de las variables y como el compilador realiza la interpolación de la mismas, ya que el uso correcto de las mismas le ahorrara más de un dolor de cabeza al usuario.

El compilador busca variables comenzando por el caracter '$' y terminando ya sea en el caracter '/' o con el fin de la palabra, en caso de que la variable no sea reconocida por el compilador se arroja un error –exceptuando el caso de '$HOME'. Observese con ejemplos:

```
var root Downloads;
...
$root/scraping;
```

```
var root Downloads;
...
$root;
```

En los primeros dos casos las variables están usadas correctamente, tras interpolar se obtendrá respectivamente:

```
Downloads/scraping;
```

```
Downloads;
```

Pasamos a un ejemplo erróneo:

```
var root Downloads;
...
$root_scraping;
```

En este caso la variable detectada por el compilador será 'root_scraping', pero al verificar la lista de variables, la misma no será encontrada, arrojando un error de compilación.

Se hará una escueta explicación con ejemplos del estado final del lenguaje, para clarificar dudas que pudiesen surgir de los cambios –aunque pocos– generados desde la primera entrega, en la cual se entrego la idea del proyecto.

Los programas del lenguaje pueden ser divididos en cinco bloques: var, from, retrieve, to y auth. Los bloques se deben encontrar indefectiblemente en el orden anterior, aunque no todos los bloques son obligatorios –para ser más precisos el bloque var y el bloque auth son opcionales–.

En el primer bloque es donde se declaran variables con la forma:

```
var var_name var_value;
```

No hay un límite para el número de variables declaradas, y a su vez es posible no declarar variables.

En el bloque from se espicifica la o las url de donde se desea scrapear de la siguiente manera:

```
from {
    https://www.example.org;
    ...
    http://www.foo.com;
}
```

Proseguimos con el bloque retrieve, aquí se puede aclarar que tipo de elementos se desea scrapear y opcionalmente se puede agregar un id. Por el momento se cuenta con una lista acotada de tags permitidos: html, head, title, body, h1, h2, h3, h4, h5, h6, img, a, p, ul, ol, dl, li y div. Por ejemplo:

```
retrieve {
    p;
    ...
    div id special_div;
}
```

Así como ya hablamos de un bloque donde se especifica el origen de los datos, también es necesario un destino (o varios), y es aquí que entra en juego el bloque to. Muy similar a su contraparte, tiene la siguiente forma:

```
to {
    Downloads/folder;
```

```
    ...
    scraping/folder/subfolder;
}
```

Último, pero no por eso de menor importancia, llegamos al bloque de autenticación, en esta iteración del proyecto únicamente válido para el campus del ITBA. Luce de esta manera:

```
auth {
    username myUsername;
    password myPassword;
}
```

Respecto de la entrega de frontend los cambios mas notables son el cambio de keywords de mínusculas a mayúsculas, realizado por estética y convención; por otra parte se quito el '=' al declarar variables, ya que no se consideró necesario y contrastaba con el resto de las declaraciones del lenguaje.

De manera ilustrativa se incluyen ejemplos de programas válidos incluidos en el README.md:

```
var root Downloads;
var sf series;

from {
    url https://es.wikipedia.org/wiki/Neon_Genesis_Evangelion;
    url https://es.wikipedia.org/wiki/Hideaki_Anno;
}

retrieve {
    div id mw-normal-catlinks;
    a;
    h1;
    img;
}

to {
    $root/$sf/evangelion;
    $root/$sf;
}
```

```
from {
    url https://campus.itba.edu.ar/ultra/courses/_27584_1/outline;
}

retrieve {
    html;
    img;
}

to {
    Facultad/campus;
}
auth {
    username myUsername;
    password myPassword;
}
```

3 Descripción del Proyecto y Fases del Compilador

El proyecto, como fue visto a lo largo del cuatrimestre en las clases prácticas consiste de dos secciones principales: frontend y backend. Vamos tratar sobre como se llevaron a la práctica estos conceptos y las fases del compilador que los atraviesan. Usaremos el siguiente diagrama provisto por la cátedra a modo de mapa.

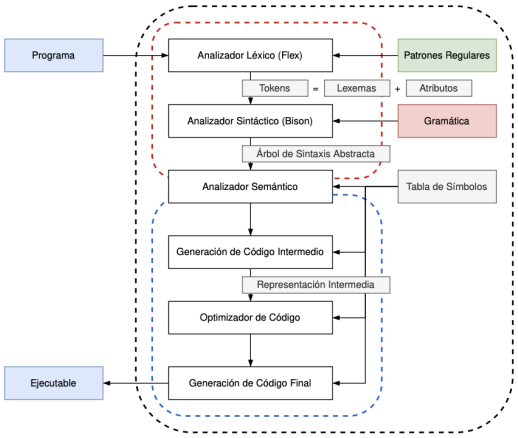

Figure 2: La arquitectura de un compilador y sus componentes principales.

Comenzamos por el frontend, donde se encuentran el analizador léxico y sintáctico. En el analizador léxico –contenido en un archivo de flex–, se encuentran las definiciones de los patrones y expresiones regulares utilizados por el escáner de entrada del compilador. En el caso presente, y con los fines del lenguaje, se pudo prescindir de scope y type-checking, ya que únicamente se manejaron strings y el único scope fue el global. Por lo tanto el resultado final resulta fácil de comprender e intuitivo.

El analizador sintáctico fue desarrollado en bison, y cuenta con la particularidad de contar con numerosas estructuras recursivas, hecho decisivo para más tarde definir las estructuras que actuarían como la columna vertebral del proyecto: el árbol de sintaxis abstracta y la tabla de símbolos.

Para está tercer entrega, se partió desarrollando a la par el árbol de sintaxis abstracta y el archivo bison-actions.c de manera de permitir que la creación e inserción de nodos al árbol fuese en simultaneo al poblado de la tabla de símbolos, de particular interés por encontrarse en la intersección entre el frontend y el backend. El árbol consta de una estructura con listas compuestas por líneas para los bloques posibles dentro del lenguaje: variables, from, retrieve, to y auth, que se construye a medida que se recorre el programa.

Sin embargo, el eje del proyecto resultó ser la tabla de símbolos, y es mediante

a ella que se posibilita la generación de código posterior. Como se menciono anteriormente, en el trabajo actual la tabla de símbolos se completó con el archivo bison.c como intermediario. Sentimos que fue crucial seleccionar una estructura para la tabla de símbolos prolija, eficiente, fácil de manejar y navegar, que aunque inicialmente no fue el caso, termino por optimizar todas las operaciones realizadas al contar únicamente con los datos pertinentes ordenados de manera coherente. Adicionalmente, la tabla de símbolos se encargó de detectar los errores de backend e interpolar variables; los errores son manejados agregandose a una lista sin detener la compilación.

Una vez llegada la instancia de generación de código, la mayoría del trabajo ya se encontraba hecha. Fue necesario generar código de JavaScript declarando arrays con los datos necesarios para que, usando Puppeteer, el programa encontrase los datos requeridos. Como el resto del programa de salida no varía dependiendo del programa de entrada, este pudo ser generado estáticamente. En esta etapa el mayor desafío resulto ser la generación de código que pudiese cumplir con los objetivos establecidos para el lenguaje en la primer entrega.

Por último se modifico el main dado para la cátedra para orquestar todos los componentes que fueron discutidos en la sección. Estas modificaciones no hacen más que inicializar la tabla de símbolos, llamar a la función que se encarga de interpolar variables, en caso de haber errores imprimirlos y retornar, y en caso contrario generar el programa de salida y liberar la memoria utilizada para tabla de símbolos.

4 Dificultades

A pesar de no contar con grandes inconvenientes, hubo dos principales dificultades con las cuales el grupo se topó. En primer lugar, se pagó por no planificar con anticipación a la hora de desarrollar el árbol de sintaxis, por lo cual se tuvo que cambiar el esquema más veces de lo necesario. Una vez que se trazó con claridad un plano a partir de los requerimientos principales de nuestro lenguaje el avance resultó mucho más continuo.

La segunda dificultad encontrada se podría decir que fue infligida por mano propia, ya que estuvo relacionada al uso de Puppeteer, con el cual el grupo no se encontraba familiarizado, particularmente en el área de imágenes y autenticación. En cuanto a autenticación el problema principal fue la heterogeneidad de sitios de logins que se encuentra en la red: por ejemplo facebook, cuenta con un login tradicional, twitter cuenta con un login en dos pasos y hay sitios en los que primero hay que abrir popus para ingresar los datos. Luego de varios intentos fallidos de encontrar botones a través de scrapping para después rellenar los datos se decidió que para esta entrega lo más sensato era limitar la autenticación al campus del ITBA. A pesar de esto no se verifica que el url ingresado ya que la idea del lenguaje es que la autenticación sea lo más cercana a universal posible, y se considera la autenticación en el campus como una demostración. Sin embargo, esto resultó ser un motivo de orgullo para el grupo, pensando en la dificultades que se le ahorrará al usuario de Scrappy.

5 Futuras extensiones y/o Modificaciones

Teniendo en cuenta el amplio panorama de posibilidades que ofrece Puppeteer, hay varias direcciones que podría tomar el grupo. Inicialmente se considera dar al usuario mayor poder de decisión en lo que respecta al formato de los datos obtenidos, pudiéndose lograr esto de manera limpia y eficiente mediante al uso de modificadores. Otra funcionalidad útil se piensa en implementar a futuro sería permitir la descarga de archivos en un sitio web filtrando por formato. De igual manera, consideramos que el rumbo que tome el proyecto a partir de este punto inicial no deberá depender tan solo de nuestras propias consideraciones, pero más bien de las necesidades que les surjan a quienes lo usen.

Nos gustaría también optimizar el script de js en el cual se basa el lenguaje para permitir la descarga directa de imágenes, ya que por el momento solo se puede obtener su src. Además, el grupo buscará ampliar las opciones de autenticación y ofrecer soporte para la mayor cantidad de sitios posibles.

6 Conclusiones

Tras quedar todo dicho y hecho, consideramos que el Proyecto Especial ha sido una gran oportunidad para materializar los conceptos visitados a lo largo del cuatrimestre, bajándolos de entre las nubes –donde toda teoría se haya–. En una época repleta de entregas podemos concluir que este trabajo ha sido de los más disfrútales y que más satisfacción nos ha generado viendo el fruto de nuestras labores. Agradecemos la oportunidad de conocer un poco más íntimamente como es que funcionan lo lenguajes de programación a los que tan asiduamente hemos sido expuestos a lo largo de los últimos años y humildemente esperamos que lo expresado en esta conclusión sea palpable en el trabajo que hemos entregado.

7 Referencias y Bibliografía

- Clases teóricas y prácticas de Autómatas, Teoría de Lenguajes y Compiladores
- (2023-05-18, v0.2.0) Backend
- https://github.com/agustin-golmar/Flex-Bison-Compiler
- (2009) Levine - flex & bison