Instituto Tecnológico de Buenos Aires

TEAM BUILDER

TLA- 1C-2023

GRUPO 12

Integrantes:

Tomas Camilo Gay Bare 62103

Tobías Perry 62064

Manuel Esteban Dithurbide 62057

Martin Hecht 62041

Abstract:

El siguiente documento detalla el trabajo realizado para el trabajo práctico de la materia Autómatas, “Teoría de Lenguajes y Compiladores” del ITBA, primer cuatrimestre 2023. El programa realiza la compilación de un lenguaje creado, diseñado para describir datos sobre un partido de fútbol, tales como el nombre del equipo, sus jugadores, el resultado y la fecha del partido. El compilador genera código de python que utilizando la librería Pillow, genera una imagen con los datos provistos.
El trabajo es la tercera entrega, y está realizado sobre las dos iteraciones previas. Se corrigieron las observaciones realizadas por el corrector y se logró cumplir con los objetivos de la primera entrega los cuales plasmamos en los tests.

TABLA DE CONTENIDOS

1 - IDEA SUBYACENTE Y OBJETIVOS DEL LENGUAJE

La idea del proyecto es crear un lenguaje para generar imágenes representativas de formaciones de equipos de fútbol. El compilador de dicho lenguaje genera un archivo “imageGenerator.py” en lenguaje Python el cual al ser ejecutado genera dicha imagen.

Para saber más acerca del lenguaje, su compilación y ejecución, leer el README.md que se encuentra en el repositorio.

2 - CONSIDERACIONES ADICIONALES

Debido a que el compilador genera un archivo en lenguaje Python, el usuario que desee generar la imagen debe tener configurado el compilador de Python con la siguiente librería:

- Python Pillow 9.1.0 (https://python-pillow.org/)

3 - DESARROLLO DEL PROYECTO Y LAS FASES DEL COMPILADOR

3.1 - Generalidades

El compilador está basado en el repositorio Flex-Bison-Compiler. Se hicieron las siguientes modificaciones para adaptarlo al lenguaje.

1. Modificación de la gramática y las acciones (Flex y Bison)
2. Parse de los datos a las estructuras.
3. Validación de datos
4. Generador de código Python

Agregado a todo esto se agregaron los test pertinentes para los casos de aceptación y de rechazo.

3.2 - Arquitectura

La arquitectura del compilador se estructuró en 4 módulos. Por un lado el analizador léxico (Flex) y sintáctico (Bison) donde se valida el formato de la entrada. Luego se arma la tabla de símbolos en la cual se generan listas con los datos de la entrada. Luego pasa por el validador y finalmente se genera el código python. Se puede resumir con el siguiente diagrama:

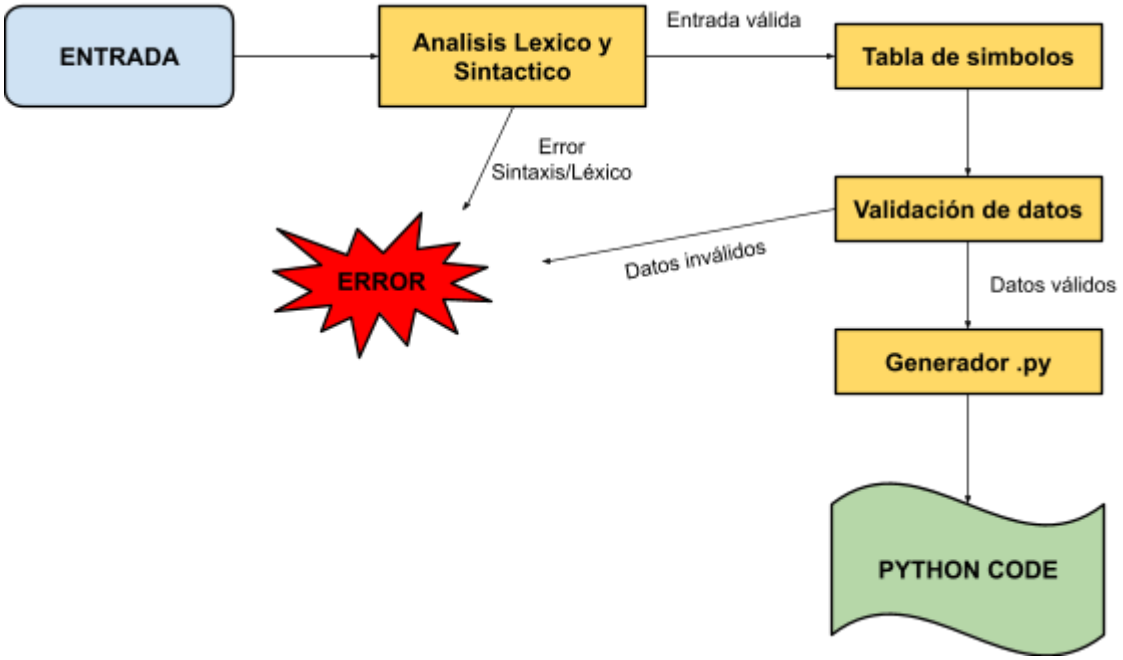

Figura 1.0: Arquitectura del compilador.

3.3 - Frontend

El frontend está dividido en dos partes: el análisis léxico y el sintáctico. En los archivos flex parseamos la entrada del cliente a las funciones que se deben ejecutar para que los archivos de bison reciban los símbolos y pueda analizar correctamente. El bison llama a las funciones que trabajan con el analizador semántico en el backend.

3.4 - Backend

Nuestro backend consiste del árbol sintáctico, de la tabla de símbolos y del módulo generador de código. El árbol sintáctico se genera a través de las funciones “grammarAction” y dentro de cada una de ellas se popula la tabla de símbolos.

Para la tabla de símbolos la estructura que mejor se adapta a nuestras necesidades es una linked-list. Creamos una estructura con tres listas dentro, una para los jugadores, otra para los suplentes y otra para las formaciones. Estas listas fueron necesarias para realizar ciertas validaciones sin recorrer el árbol, tales como corroborar que la cantidad de jugadores se corresponda con la cantidad de jugadores en la suma de las posiciones en las formaciones provistas (ej: 11 jugadores y 4-4-2 [+1]).

El archivo de generación de código toma los datos de las listas mencionadas previamente y del árbol sintáctico para generar código de

python que al ejecutarse, genera una imagen con la formación del equipo, el nombre, los suplentes, el resultado y la fecha. En dicho archivo es que se realizan las verificaciones semánticas con una función “validator”. El generator retorna un código de error y un archivo vacío, o un código de éxito y el archivo correspondiente a los datos provistos por el cliente.

La imagen generada tiene el siguiente formato:

Figura 2.0: Imagen generada por el compilador

3.5 - Sintaxis

El lenguaje se estructura de la siguiente manera:

```
START
TEAM 'teamName'[opcional] OF playerAmount PLAYERS
FORMATION
formation [pueden ser mas de una]
Tag LINEUP/LINEUPNONUM
number: 'player name' [en caso de LINEUPNONUM solo se requiere nombre del jugador] {lista}
SUBSTITUTES/SUBSITUTESNONUM [debe coincidir con el tipo de LINEUP] [opcional]
number: 'player name' {lista}
METADATA [opcional]
DATE: 'fecha' [opcional] RESULT 'result' [opcional]
END
```

Debido a que la generación de imágenes es compleja, los datos ingresados tienen que cumplir con las siguientes condiciones:

1. La cantidad de defensores (4-4-2 = 4 def - 4 mid - 4 att) en una línea no puede ser mayor a 4.
2. La fecha debe estar en formato ISO 8601. Ej: 2023-24-05
3. En caso de usar LINEUPNONUM se debe usar el SUBSTITUTESNONUM correspondiente. En el caso contrario, en el uso de LINEUP se debe usar SUBSTITUTES a secas.
4. Los strings no son contextos, pero de todos modos incluímos caracteres especiales para ampliar los símbolos aceptados.
5. Se verifica que cada jugador tenga un número de camiseta único.
6. El proyecto soporta correctamente hasta 15 suplentes por cuestiones visuales de la imagen generada.

En referencias se encuentran ejemplos de uso.

3.6 - Testing

Los tests realizados son los siguientes:

Aceptación:
- 01: Formacion futbol 5
- 02: Formacion futbol 8
- 03: Formacion futbol 11
- 04: Formación sin suplentes
- 05: Formación sin fecha ni nombre de equipo
- 06: Formación sin metadatos
- 07: Formación sin numeración
- 08: Doble formación

Rechazo:

- 01: No cumple formato de formación. (Ej formación correcta: 4-4-2)
- 02: Caracteres inválidos en las variables
- 03: Formato de fecha incorrecto
- 04: Jugadores sin numerar cuando se requiere numeración
- 05: Campo de sustitutos presente pero vacío
- 06: Campo de titulares y sustitutos con opción de numeración alternada
- 07: Campos de titulares vacío
- 08: Formato de resultado invalido
- 09: La cantidad de jugadores no se corresponde a la formación
- 10: La cantidad de jugadores no se corresponde a la formación
- 11: La cantidad de jugadores instanciados es mayor al numero de jugadores
- 12: La línea de defensores es mayor a 4 jugadores
- 13: Dos jugadores tienen el mismo número

4 - DIFICULTADES

Durante el desarrollo del proyecto nos encontramos con las siguientes dificultades:

1. Al armar el generador.c, el cual genera un código en lenguaje python, surgió la dificultad de no saber la sintaxis de dicho lenguaje. Esto causó demoras en el desarrollo mientras el equipo aprendía e investigaba cómo generar las imágenes.

2. Durante el armado del árbol de sintaxis tuvimos algunas complicaciones debido a que hay una relación doble entre los tokens, las unions del archivo bison-grammar-1 y las estructuras definidas en el árbol, y no siempre correspondimos los nombres correctamente.

5 - FUTURAS EXTENSIONES Y MODIFICACIONES

A futuro se podría aceptar cualquier tipo de formación sin límites de cantidad de jugadores ni suplentes. Esto conlleva cambiar el archivo de salida optimizando los diferentes casos de uso.

Otra extensión que se podría implementar sería el agregado de imágenes propias de cada jugador. Su implementación no sería de mayor dificultad, debido a que simplemente hay que agregar una sintaxis extra para que el usuario inserte la ruta del archivo imagen y que el generador.c valide la existencia de dicho archivo.

Esta misma extensión se puede aplicar a un escudo del equipo, foto del equipo completo, camiseta, etc, etc.

Poniendo como foco la UX del programador a la hora de usar el lenguaje, sería óptimo la implementación de syntax-highlighting o la implementación de un compilador con modificaciones en vivo, donde el usuario pueda ir viendo la imagen final a medida que programa (Similar a https://es.overleaf.com/ LATEX)

En términos de las modificaciones la más significativa, sería optimizar el código python para generar una imagen de mejor calidad y estética.

6 - REFERENCIAS Y BIBLIOGRAFÍA

Algunos ejemplos:

```
START
TEAM 'The Octopus' OF 8 PLAYERS
FORMATION
4-2-1
LINEUP
1: 'Martinez Pedro'
2: 'Kim Soo-hyun'
3: 'Jones Emma'
4: 'Gonzalez Diego'
5: 'Wong Wei'
6: 'Davis Michael'
7: 'Lee Jae-yong'
8: 'Gomez Sofia'
SUBSTITUTES
9: 'La Maquina'
10: 'Pedrito'
METADATA
DATE: '2021-09-03' RESULT: '2-1'
END
```

```
START
TEAM OF 5 PLAYERS
FORMATION
2-1-1
LINEUP
1: 'El Manco'
2: 'Alves Daniel'
3: 'Pedrito Lopez'
4: 'Gaston Perez'
5: 'Pepito Radiactivo'
SUBSTITUTES
6: 'La Maquina'
```

```
7: 'Pedrito'
METADATA
RESULT:'5-0'
END
```

```
START
TEAM 'The Red Devils' OF 5 PLAYERS
FORMATION
2-1-1
LINEUP
1: 'El Manco'
2: 'Alves Daniel'
3: 'Pedrito Lopez'
4: 'Gaston Perez'
5: 'Pepito Radiactivo'
METADATA
DATE:'2001-07-07' RESULT:'5-0'
END
```

CList utilizado en el código:

- https://github.com/AlexanderAgd/CLIST/blob/master/README.md

Repositorio de github del compilador original

- **https://github.com/agustin-golmar/Flex-Bison-Compiler**