Autómatas, Teoría de Lenguajes y Compiladores

Parte II: Backend

Grupo 8

Mariano Agopian, legajo: 62317, maagopian@itba.edu.ar

Matias Ignacio Luchetti, legajo: 62337, mluchetti@itba.edu.ar

Santiago Tomas Medin, legajo: 62076, smedin@itba.edu.ar

Sofia Paula Altman Vogl, legajo: 62030, saltman@itba.edu.ar

Tabla de Contenidos

1. Introducción: Idea General del Proyecto

El proyecto funciona como un parser de JSON para construir un gráfico representativo a un partido de fútbol, hockey o basquetbol. La salida genera un archivo HTML llamado "*match.html*" que equivale a una cancha del deporte elegido, con sus jugadores posicionados según la formación recibida en la entrada. El archivo de salida se encuentra en la carpeta "*out*". Si se quiere más información acerca del funcionamiento del proyecto, se encuentra un archivo README.md en el proyecto con una explicación más detallada.

2. Consideraciones Adicionales

Como se mencionó previamente, la salida del compilador es únicamente un archivo HTML. De esta manera, para ver la representación gráfica del archivo se debe correr el mismo. Para visualizarlo, durante el desarrollo del proyecto se utilizó el sitio web https://jsfiddle.net/ , donde simplemente se debe pegar el contenido del archivo generado en la sección de "HTML", y apretar el botón de "*Run*".

3. Desarrollo del Proyecto y las Fases del Compilador

3.a. Generalidades

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

- Modificación de la gramática y las acciones (Flex y Bison)
- Parseo de los datos a las estructuras.
- Validación de datos.
- Generación del archivo HTML.
- Tests de aceptación y rechazo.

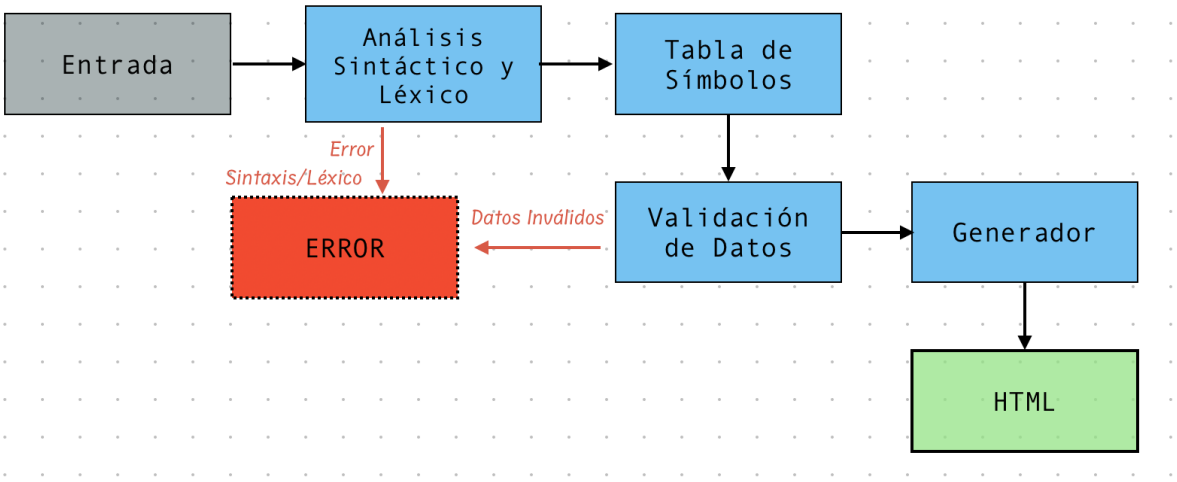
3.b. Arquitectura

La arquitectura del compilador se estructuró en 4 módulos. En primer lugar el analizador léxico (Flex), y luego el sintáctico (Bison) donde se valida el formato de la entrada. Ambos fueron proveídos por la cátedra y fueron ajustados en el desarrollo del trabajo para validar la entrada de manera correcta.

Por otro lado, se arma la tabla de símbolos en la cual se generan listas con los datos de los jugadores y equipos. La implementación de las listas ya que no encontramos librerías que nos parecieran simples teniendo en cuenta que únicamente usamos las funciones más comunes.

Por último, pasa por el validador y finalmente se genera el código HTML.

Para facilitar la visualización de la arquitectura del compilador, dejamos a continuación un diagrama con la lógica de la misma.



Imágen 1: *Arquitectura del compilador.*

3.c. Frontend

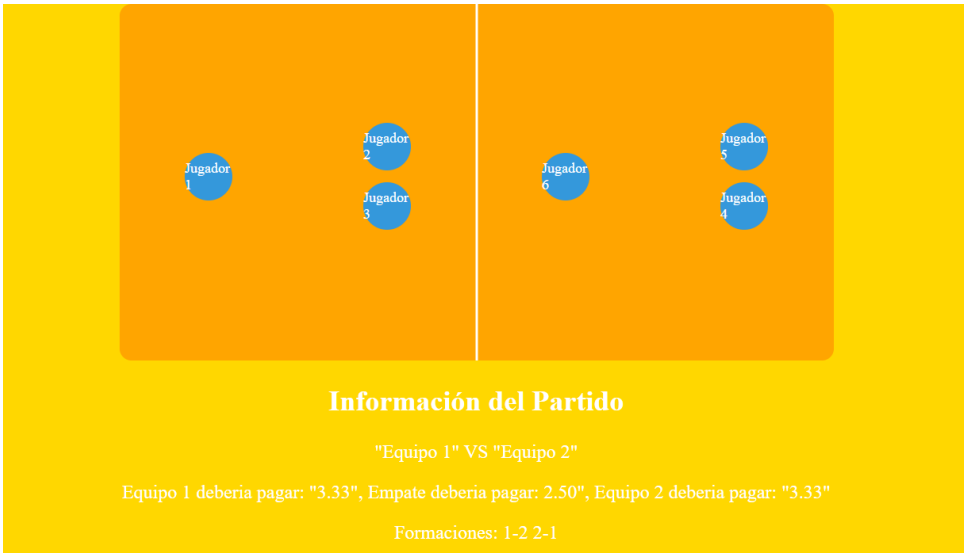
El frontend está dividido en dos partes: el análisis léxico y el sintáctico. En los archivos flex se parsea la entrada generando los tokens correspondientes y llamando a las funciones que se deben ejecutar para que los archivos de bison reciban los símbolos y pueda analizar correctamente. El bison llama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

3.d. Backend

El backend consiste del árbol sintáctico, la tabla de símbolos, el validador y el 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 utilizamos una "*simple list*", donde un nodo referencia al próximo jugador en la lista, y cada jugador tiene únicamente su nombre. Estas listas (una por cada equipo) fueron utilizadas para obtener los nombres de los jugadores sin tener que recorrer todo el árbol y soportar una matriz dentro del mismo. Algunas de las validaciones para las que utilizamos las listas fue para verificar que la formación tenga la cantidad de jugadores correspondientes, o que la cantidad coincida con aquella de un deporte válido (que no haya 3 jugadores para un partido de Futbol 5).

El archivo de generación de código toma los datos de las listas mencionadas previamente y del árbol sintáctico para generar código de HTML que al ejecutarse, genera una imagen con la formación de los equipos, el nombre de los mismos y la cuota de apuestas de cada resultado. El generator retorna un código de error y un archivo vacío, o un código de éxito y el archivo correspondiente a los datos de la entrada. La imagen generada tiene el siguiente formato:



Imágen 2: *Vista del html generado para un partido de basquetbol.*

En este caso, la entrada corresponde a un partido de baloncesto, por lo que la cancha y el fondo toman esos colores, mientras que las variaciones de hockey y fútbol toman distintos tonos de verde.

3.e. Sintaxis

La entrada deberá ser un archivo JSON, similar al mostrado en la imagen a continuación:

```json
{
  "deporte": "Basquet 3",
  "probabilidades": "30-40-30",
  "equipos": [
    {
      "nombre": "Equipo 1",
      "formacion": "1-2",
      "jugadores": [
        {
          "jugador": "Jugador 1"
        },
        {
          "jugador": "Jugador 2"
        },
        {
          "jugador": "Jugador 3"
        }
      ]
    },
    {
      "nombre": "Equipo 2",
      "formacion": "2-1",
      "jugadores": [
        {
          "jugador": "Jugador 4"
        },
        {
          "jugador": "Jugador 5"
        },
        {
          "jugador": "Jugador 6"
        }
      ]
    }
  ]
}
```

Imágen 3: *Ejemplo de archivo de entrada json.*

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

- Los campos “deporte”, “equipos”, “nombre”, “formación”, “jugadores” y “jugador” son obligatorios. En el caso de “formación” y jugadores deberán aparecer una vez por equipo, en el caso de “jugador” se podrá repetir varias veces dependiendo del deporte elegido. El campo “probabilidades” es opcional.
- Los campos “equipos” y “jugadores” deben ser vectores.
- La formación deberá tener mínimo dos tipos de jugadores distintos, y máximo tres (debe tener la forma X-X-X o X-X), teniendo como máximo 5 jugadores del mismo tipo.
- Todos los strings utilizados para dar valores a los campos deben estar encerrados por comillas.
- Las probabilidades deberán tener la forma XX-XX-XX donde se expresa el porcentaje de victoria del primer equipo, luego de empate, y luego del segundo equipo.

3.f. Testing

Los tests realizados fueron los siguientes:

Aceptados:

1. Partido de fútbol 11 con probabilidades de victoria
2. Partido de fútbol 11 sin probabilidades de victoria
3. Partido de fútbol 8 con probabilidades de victoria
4. Partido de fútbol 8 con probabilidades de victoria
5. Partido de fútbol 5 con probabilidades de victoria
6. Partido de futbol 5 sin probabilidades de victoria
7. Partido de hockey (8 jugadores) con probabilidades de victoria
8. Partido de hockey (8 jugadores) sin probabilidades de victoria
9. Partido de hockey (11 jugadores) con probabilidades de victoria
10. Partido de hockey (11 jugadores) sin probabilidades de victoria
11. Partido de basquet 5 con probabilidades de victoria
12. Partido de basquet 5 sin probabilidades de victoria
13. Partido de básquet 3 con probabilidades de victoria
14. Partido de básquet 3 sin probabilidades de victoria
15. Partido de fútbol 11 con “whitespaces”

Rechazados:

1. Partido de un deporte invalido (rugby)
2. Partido con probabilidades sin empate
3. JSON con clave “deportes” mal escrita (“deportess”)
4. JSON sin campo “jugadores” para uno de los equipos
5. JSON sin campo “deportes”
6. El valor del campo “jugador” contiene en su valor valores no alfanuméricos
7. El valor del campo “jugador” no está contenido por comillas
8. Partido de hockey con 10 integrantes por equipo
9. Partido donde un equipo tiene una formación invalida (4-5-1-3)

10. Partido donde hay claves extra como “liga”

4. Dificultades

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

- Originalmente nuestra idea era que el compilador genere una imagen con los jugadores posicionados en una cancha. Al comenzar a hacerlo, nos dimos cuenta que no era tan simple como pensábamos. Luego de hablar con la cátedra , se nos proporcionó varias opciones para reemplazarlo, como un script de python, usar SVG, o un archivo de HTML. Decidimos decantarnos por la última opción.
- Para la creación del árbol de símbolos, intentamos utilizar varias implementaciones de listas en C sacadas de distintas páginas o repositorios Git. Sin embargo, todas nos presentaron escenarios como no incrementar el “count” después de hacer un “*add*”. Nos dimos cuenta que necesitábamos únicamente funciones simples de listas como ya hemos hecho anteriormente, por lo que decidimos hacer una implementación propia.
- Durante la primera entrega, al hacer las actions de flex, no nos guardamos el valor del string ya que no era necesario. Únicamente debíamos verificar la sintaxis, por lo que no nos importaba el valor de un string por ejemplo. Al imprimir los datos mediante el uso del logger, comprendimos el problema e implementamos correctamente las acciones en flex.

5. Futuras extensiones y modificaciones

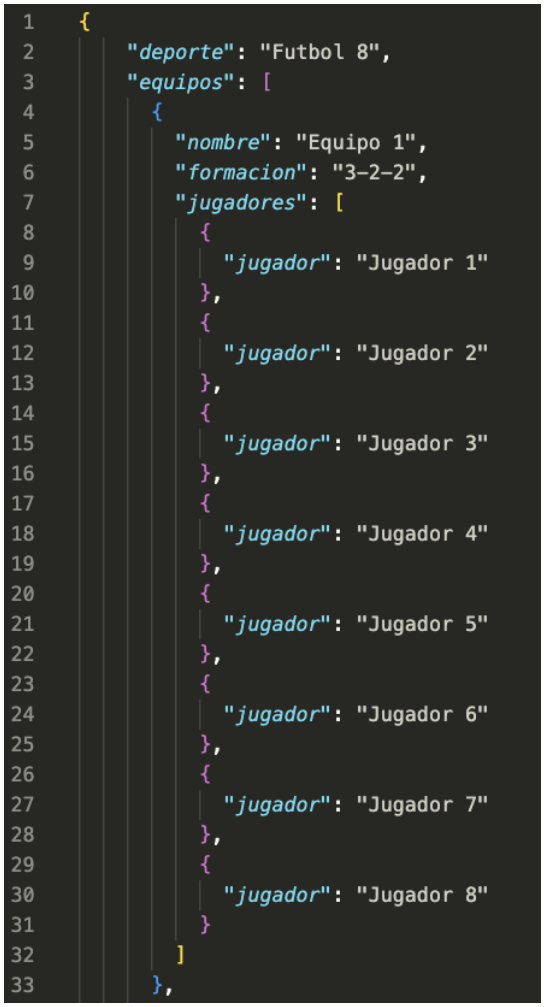
En un futuro, se podría aceptar distintos tipos de deportes colectivos como rugby, voley, handball, etc . Esto se podría llevar a cabo sin mucha dificultad, únicamente habría que agregarlo en la parte de sintaxis y al enum de “`SportType`”.

También se podría implementar un “banco de suplentes”, donde los primeros N jugadores (siendo N el número de jugadores necesarios para jugar el deporte), se incluyan en la cancha, y se muestre una serie de jugadores disponibles para realizar cambios como los tienen todos los deportes que elegimos. Su implementación implicaría cambiar reglas de validación y cambiar algunas clases del HTML para que se muestren de manera distinta

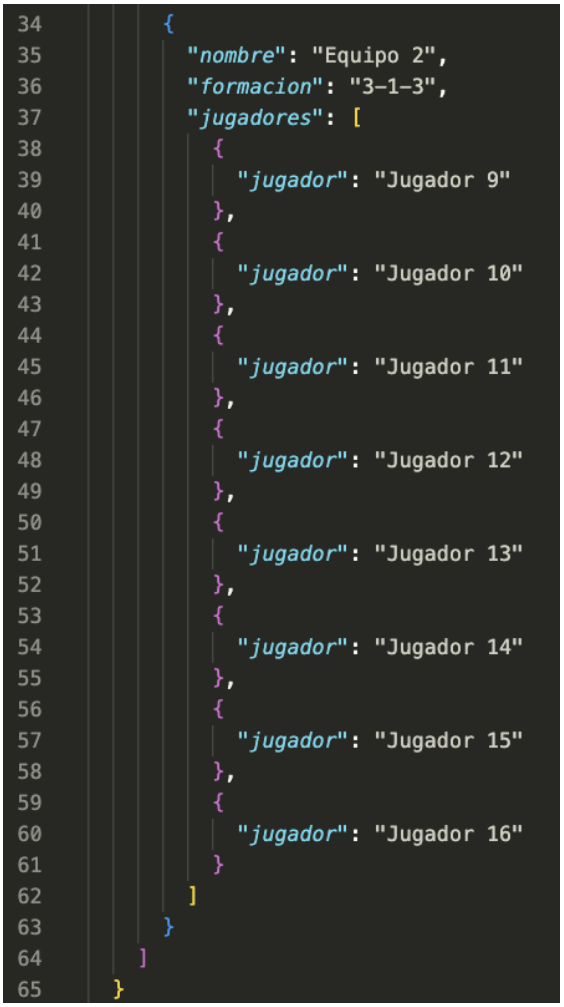
Por último, el archivo HTML generado muestra correctamente a los jugadores en la cancha pero carece de muchas animaciones. Se podría agregar más detalle al código HTML de manera que se agreguen dibujos de arcos, aros, pelotas, o más símbolos relacionados al deporte del partido.

6. Referencias y Bibliografía

6.a. Ejemplos de Tests: Entrada y Salida



```
{
    "deporte": "Futbol 8",
    "equipos": [
      {
        "nombre": "Equipo 1",
        "formacion": "3-2-2",
        "jugadores": [
          {
            "jugador": "Jugador 1"
          },
          {
            "jugador": "Jugador 2"
          },
          {
            "jugador": "Jugador 3"
          },
          {
            "jugador": "Jugador 4"
          },
          {
            "jugador": "Jugador 5"
          },
          {
            "jugador": "Jugador 6"
          },
          {
            "jugador": "Jugador 7"
          },
          {
            "jugador": "Jugador 8"
          }
        ]
      },
```



```
      {
        "nombre": "Equipo 2",
        "formacion": "3-1-3",
        "jugadores": [
          {
            "jugador": "Jugador 9"
          },
          {
            "jugador": "Jugador 10"
          },
          {
            "jugador": "Jugador 11"
          },
          {
            "jugador": "Jugador 12"
          },
          {
            "jugador": "Jugador 13"
          },
          {
            "jugador": "Jugador 14"
          },
          {
            "jugador": "Jugador 15"
          },
          {
            "jugador": "Jugador 16"
          }
        ]
      }
    ]
}
```

Imágenes 4.a y 4.b: *Entrada JSON del test 4.*

Imágen 5: *Salida HTML del test 4*


```
{
    "deporte": "Basquet 5",
    "probabilidades": "60-20-20",
    "equipos": [
        {
            "nombre": "Equipo 1",
            "formacion": "2-2-1",
            "jugadores": [
                {
                    "jugador": "Jugador 1"
                },
                {
                    "jugador": "Jugador 2"
                },
                {
                    "jugador": "Jugador 3"
                },
                {
                    "jugador": "Jugador 4"
                },
                {
                    "jugador": "Jugador 5"
                }
            ]
        },
        {
            "nombre": "Equipo 2",
            "formacion": "2-1-2",
            "jugadores": [
                {
                    "jugador": "Jugador 6"
                },
                {
                    "jugador": "Jugador 7"
                },
                {
                    "jugador": "Jugador 8"
                },
                {
                    "jugador": "Jugador 9"
                },
                {
                    "jugador": "Jugador 10"
                }
            ]
        }
    ]
}
```


Imágenes 6.a y 6.b: *Entrada JSON del test 12.*

Imágen 7: *Salida HTML del test 12.*

6.b. Links a los Repositorios

Repositorio de github del compilador original:

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

Repositorio de este trabajo:

https://github.com/matiluchetti/TP-TLA