Proyecto Especial: Diseño e implementación de un lenguaje

Informe Backend: SrQuix

Miembros del grupo

Francisco Marcos Ferrutti, fferrutti@itba.edu.ar
Ignacio Bruzone, ibruzone@itba.edu.ar
Tomás Martinez, tomartinez@itba.edu.ar
Luca Seggiaro, lseggiaro@itba.edu.ar

72.39 - Autómatas, Teoría de Lenguajes y Compiladores

Tabla de contenidos

1. Introducción

En este informe se analizará el desarrollo del proyecto SrQuix, un compilador que permite escribir circuitos de manera sencilla, adaptandolos a la librería LaTeX “Circuitikz”[1], la que permite a uno dibujarlos a través de vectores. Consideramos que muchos usuarios pueden interesarse por SrQuix, nuestro programa, ya que no es a base de vectores, facilitando su creación.

El objetivo de este trabajo es poder aplicar los conocimientos adquiridos en el transcurso de la materia Autómatas, Teoría de Lenguajes y Compiladores, abarcando las distintas fases del trabajo en paralelo a las clases relacionadas con sus distintas etapas.

2. Consideraciones adicionales

Como se mencionó en la introducción, el factor a considerar específico al proyecto es el uso de la librería Circuitikz para la generación de código, y lo que implica la conversión a un plano 2D. La implementación de ésta y sus dificultades serán abordadas a lo largo del informe.

3. Desarrollo del proyecto

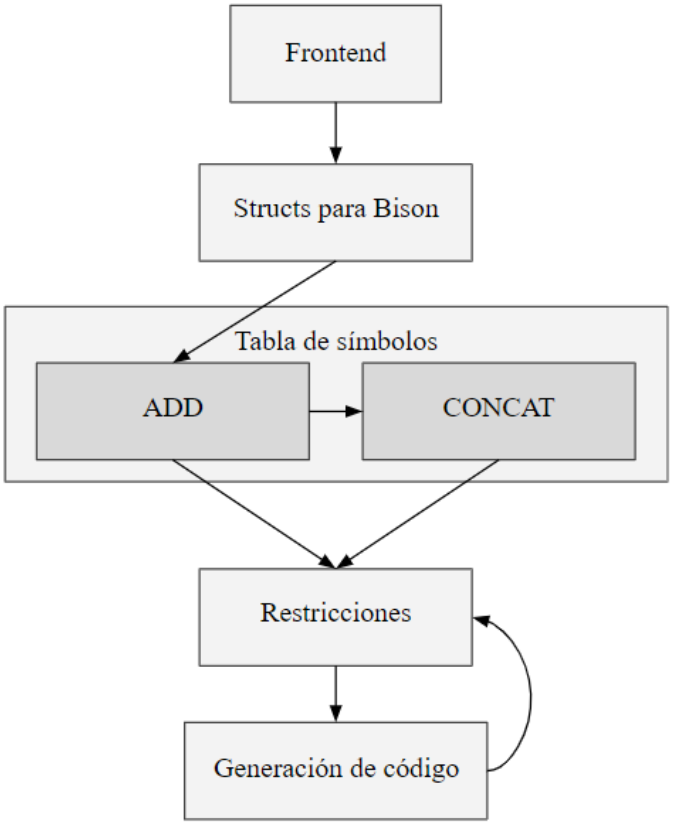

Imagen 1: Diagrama del desarrollo del proyecto

Como se puede observar en la Imagen 1, el proyecto siguió las fases propuestas por la cátedra[3]. En particular, destaca la falta de scopes y type-checking. La primera no es pertinente a nuestro proyecto, ya que se trabaja únicamente en un scope global. En cambio para la segunda, no fue necesario debido a que distinguimos entre valores numéricos y texto, es decir, en ningún momento definimos ambos bajo una categoría como “constant”.

La implementación de la sección de tabla de símbolos fue compuesta de dos instancias, la de asignación de los componentes, y la de concatenación. La primera agrega los componentes a la tabla en base a las declaraciones de objetos. Para el segundo caso, se encuentran los componentes presentes, y los concatena en base a las reglas del lenguaje.

En base a estas, se generan las restricciones, que almacenan en una lista de mensajes todos los errores que ocurren en esta etapa del trabajo.

Una vez implementada la tabla de símbolos y un conjunto de restricciones iniciales, desarrollamos la generación de código, en donde transformamos el código generado por nuestro lenguaje a LaTeX. La generación nos permitió encontrar restricciones no consideradas, y al implementarlas pudimos mejorar la generación de código iteradas veces.

4. Dificultades

Respecto a las dificultades del proyecto, consideramos valioso mencionar nuestro principal objetivo a lo largo del desarrollo: facilitar el uso de los usuarios evitando la necesidad de tener experiencia con vectores para construir circuitos. Si bien nuestros usuarios pueden tener conocimientos acerca del manejo de vectores, a la hora de concatenar múltiples componentes a través de coordenadas puede resultar confuso, y puede tomar tiempo antes de llegar a un circuito cerrado que representa lo que uno tiene en mente. Teniendo esto en cuenta, pasamos por distintas iteraciones de construcción del lenguaje, tanto que la entrega de front end se vió modificada durante esta segunda instancia.

```
source bat1 {
   value: 12;
}

source bat2 {
   value: 24;
}

// Listado de dispositivos rápido
resistances r1, r2, r3, r4 {
   values: 4, 2, 3, 2.5;
}

series miserie {
   bat1;
   r2;
   r4;
}

series serie2 {
   miserie.startAfter(r2).endBefore(bat 1);
   r3;
   bat2.reversed;
}

series serie3 {
   serie2.startAfter(bat1).endAfter(r3);
   r4;
}
```

```
resistance r1, r2, r3, r4 = [4, 2, 3, 2.5];

source bat1, bat2 =[12, 24];

node nodo1, nodo2, nodo3, nodo4;

nodo1>r1>nodo3;
nodo4>bat2>nodo3;
nodo4>bat1>nodo1;
nodo1>r2>nodo2>r3>nodo3;
nodo2>r4>nodo4;
```

Tabla 1: Comparación entre la 1ra entrega y modelo actual, circuito equivalente

Como se puede observar, se redujo considerablemente la cantidad de código para un mismo circuito. Esto es con el objetivo mencionado previamente, facilitar la experiencia de

uso. Adicionalmente, se cambió a un diseño centrado en nodos, haciendo hincapié en las intersecciones de distintos caminos en un circuito.

Durante el desarrollo del trabajo, surgieron distintas ideas de cómo cambiar la gramática original para alcanzar nuestro objetivo, pero debimos conscientemente mantenernos la idea propuesta (ya distinta de la original), para poder seguir avanzando. Sino, hasta el día de hoy estaríamos perfeccionándola. El hecho de aprender los conceptos en simultáneo al trabajo catalizó éste fenómeno, ya que fuimos siendo más conscientes de los pasos a seguir.

Hubo otra parte de la implementación que, en retrospectiva, nos gustaría refactorizar, más relacionada al código C. Debido a la estructura de SrQuix, es natural permitir que se puedan concatenar componentes y nodos de manera casi libre. Por ello, al definir sus estructuras en la tabla de símbolos en C:

```c
struct component_t {
    char *component_name;
    int constant;

    ComponentType * component_type;

    object_type prev_type;
    void *prev;

    object_type next_type;
    void * next;
};
```

```c
struct node_t {
    char *name;

    object_type dir_type[4];
    void *dir[4];
};
```

Tabla 2: Estructuras de componentes y nodos en la tabla de símbolos

Permitimos libertad respecto al objeto a concatenar, especificando a través del "`object_type`" el tipo de objeto a concatenar. Si bien esto puede resultar intuitivo, resultó en mucho overhead de código, ya que se debía considerar los distintos casos, especialmente a la hora de concatenar dos objetos (4 casos distintos).

Otra dificultad particular de nuestro proyecto fue la transformación a vectores a partir del árbol de sintaxis abstracta y la tabla de símbolos. Nos propusimos distintas alternativas de cómo implementarlo, y terminamos aplicando una metodología fundamental para la programación, ir paso por paso. Comenzamos con imprimir un componente, después una serie de componentes, después series de componentes, y finalmente, las conectamos. La concatenación de nodos se toma como una serie vacía, en caso contrario, deberíamos hacer diversas iteraciones sobre el diagrama para poder incluir las conecciones. Pasamos mucho

tiempo planteando distintas maneras de rotar las series y evitar superposición, cuando no era el enfoque principal del trabajo.

Por último, no consideramos que la librería de LaTeX[2] no soportara los componentes que deseamos implementar. Por ello, debimos reemplazar el voltímetro monofásico y el óhmetro por “lámpara” y un componente genérico.

5. Futuras extensiones

Agrupamos las extensiones del trabajo en las siguientes 4 categorías:

➔ Heurísticas

Respecto a la experiencia del usuario, nos gustaría profundizar la generación del circuito. Si bien mostrar las distintas cadenas unidas por los nodos puede ser el primer paso, nos gustaría generarlas en distintas direcciones, rotando cuando sea necesario, para que se parezca más a un circuito tradicional. Investigar más en profundidad acerca de la teoría de grafos, planaridad, y cómo éstas se implementan en código puede resultar útil para el desarrollo de futuras heurísticas.

➔ Más componentes

También nos gustaría permitir que el usuario tenga acceso a una mayor variedad de componentes, sea de la física tradicional como de la electrónica. La librería soporta una variedad de componentes que no implementamos. Otros componentes como los transistores o compuertas lógicas requieren expandir el funcionamiento del programa, ya que dejan de lado nuestra implementación de objeto previo y siguiente.

➔ Personalización del circuito

Continuando con la mejora de experiencia para los usuarios, nos gustaría que éstos puedan tener mayor control sobre el circuito. Creemos que permitir que le den un nombre, rotar componentes y cambiar la unidad de estos también sería valioso. Adicionalmente, consideraríamos agregar otros parámetros, para delimitar componentes, marcarlos como variables (flecha en diagonal), etc.

➔ Cálculos sobre el circuito

Otra feature que se nos ocurrió es que el usuario pueda poder ver los resultados de la ecuaciones de mallas y poder calcular resistencia, intensidad, potencial, etc, a partir del circuito y los valores que se le ingresan. Pensamos que esto nos hubiera servido cuando cursamos física 3 a modo de corrección de ejercicios y trabajos de laboratorio que al estar aprendiendo puede ser confuso.

➔ Refactorización

En cuanto al código, como fue mencionado en la sección anterior, nos gustaría refactorizar la estructura de la tabla de símbolos. Consideramos que la implementación en generación de código, donde creamos líneas de componentes con un nodo de comienzo, fin, y el tamaño de la cadena, facilitarían el overhead de verificación de tipos (no los habría).

➔ Mejorar errores

Finalmente, nos gustaría expandir la estructura de impresión de errores. Si bien mostramos el componente dónde ocurren y el mensaje de error adecuado, creemos que es sumamente valioso imprimir errores claros y concisos, en donde el usuario sepa casi instantáneamente dónde ocurrieron. Mejoras que consideramos: explicitar la línea del error, utilizar colores y ejemplificar el caso de error.

6. Conclusión

Como grupo, sentimos que este trabajo puede haber sido una introducción a un campo muy interesante. Creemos haber simplemente abarcado un sector pequeño de la punta del iceberg, y que las distintas extensiones mencionadas comprueban que este proyecto puede ser continuado.

Destacamos principalmente la conexión con las distintas áreas de la informática. Si bien fue un desafío, fue sumamente interesante implementar la generación del código, ya que pasamos de parsear y ordenar la información obtenida a realmente producir un resultado.

7. Referencias y bibliografía

[1] CircuiTikz package, Overleaf
https://www.overleaf.com/learn/latex/CircuiTikz_package

[2] LaTeX Graphics using TikZ, Josh Cassidy (Agosto 2013)
https://www.overleaf.com/learn/latex/LaTeX_Graphics_using_TikZ%3A_A_Tutorial_for_Beginners_(Part_4)%E2%80%94Circuit_Diagrams_Using_Circuitikz

[3] Backend de un compilador simple, ITBA, cátedra 72.39 - Autómatas, Teoría de Lenguajes y Compiladores (v0.2.0)
https://docs.google.com/document/d/1KF-qylH4ciL5WjRQRuRSulp2xFb38X6a4THut2GDcfQ/edit