Teoría de Lenguajes y Autómatas

Proyecto Especial:

Organigraph

Grupo G56:
Lucas Gonzalez Rouco (63366)
Tobias Ves Losada (63342)
Juan Cruz Boullosa Gutierrez (63414)

Profesores:
Arias Roig, Ana Maria
Golmar, Mario Agustin

1. Introducción

Organigraph es un lenguaje de programación diseñado específicamente para la representación y visualización de la jerarquía de empleados dentro de una empresa. Este lenguaje imperativo permite a los usuarios declarar empleados y sus relaciones jerárquicas, facilitando la creación de una estructura organizacional clara y precisa. Cada declaración de empleado en Organigraph puede incluir la información que sea requerida mediante atributos.

Además, Organigraph ofrece una amplia variedad de herramientas para modificar y personalizar la estructura organizacional. Los usuarios pueden agregar o eliminar empleados, cambiar sus posiciones dentro de la jerarquía y ajustar la información de cada empleado de manera sencilla.

2. Modelo Computacional

2.1. Dominio

Organigraph es un lenguaje de programación diseñado para convertir estructuras jerárquicas de empleados en documentos en formato JSON. Luego, dicho archivo puede integrarse con herramientas de visualización para mostrar la jerarquía en forma de grafo.

El lenguaje permite declarar variables del tipo empleado. Estos objetos tienen un nombre, metadata, que son atributos que cuentan con una etiqueta y un contenido que puede ser numérico o de tipo string. Además, los empleados cuentan con dos listas. Una con los empleados que están por encima de sí mismo en la jerarquía y otra con los que están debajo.

También se pueden utilizar listas de empleados, las cuales pueden almacenarse en variables. Pueden utilizarse mientras se declaran o bien pueden obtenerse al buscar empleados que tengan ciertos atributos deseados.

Luego se puede establecer relaciones jerárquicas estableciendo debajo de quienes se encuentra. Y también permitiendo remover a los empleados del grafo al que pertenecen.

2.2. Lenguaje

El lenguaje ofrece las siguientes prestaciones:

(I). Crear uno o más empleados.

(II). Crear empleados con atributos definidos mediante un tag y un valor.

(III). Declarar arreglos de empleados.

(IV). Crear un empleado y directamente indicar quienes son sus superiores en cuanto a jerarquía.

(V). Buscar al conjunto de empleados que contengan atributos particulares y con valores deseados.

(VI). Indicar que los empleados en una lista están por debajo de los empleados que se encuentran en otra lista.

(VII). Remover a un empleado del grafo al que pertenece, removiendo a todos los subordinados que no tengan otro jefe.

(VIII). Reemplazar a un empleado por otro existente.

(IX). Reemplazar a un empleado por uno nuevo.

(X). Obtener una lista de los empleados que cumplan con ciertos criterios y utilizar dicha lista para establecer jerarquía. Los criterios son:

- (X.1). Contienen ciertos atributos con un valor específico.
- (X.2). Son subordinados directos de un empleado.
- (X.3). Son subordinados directos de un empleado y dicho empleado.
- (X.4). Son empleados que tienen a los mismos jefes directos que un empleado particular.

3. Implementación

3.1. Frontend

El desarrollo frontend comienza con el procesamiento de la entrada utilizando Flex para la creación de tokens. Se define un alfabeto y establecen diversos contextos como, por ejemplo **MULTILINE_COMMENT** y **ONELINE_COMMENT**, para el manejo de comentarios multilínea y de una línea respectivamente.

En esta etapa, se definen tokens como **search**, **remove**, entre otros, y se realizan acciones específicas para cada token. Para evitar el uso de switch, en ciertos casos se envía el token a la función que realiza la acción. Como por ejemplo en el caso de llaves, corchetes y paréntesis ya que se utiliza la misma acción para ambos tokens.

El desarrollo frontend continua con el análisis sintáctico utilizando Bison, donde se establecen las gramáticas aceptadas por el lenguaje. Luego utiliza dichas reglas para armar el árbol de sintaxis abstracta (AST) a partir de los tokens generados por FLEX.

3.2. Backend

El desarrollo de backend consiste en generar la estructura que permita obtener la salida del compilador. En este caso, dicha estructura es el GeneratorState que es una tabla de símbolos con ciertas peculiaridades.

Como alternativa a una tabla de símbolos única, se utilizan dos tablas, una con las variables del tipo empleado y otra con las variables del tipo vector de empleados. De esta forma al buscar un elemento de cierto tipo de dato basta con recorrer la tabla correspondiente al tipo y comparar identificadores.

Esta estructura se encuentra en el archivo generator.c que es donde se procesan los datos del frontend y se modifica el estado para generar la salida.

Por último, para la generación del código se recorren las estructuras de estado que contiene todos los empleados y por cada uno de los empleados que tengan la lista de jefes vacía, se comienza a imprimir las relaciones en el archivo JSON.

3.3. Adicionales

Organigraph incluye una funcionalidad adicional para facilitar la visualización de estas estructuras. Al ejecutar el lenguaje, se genera automáticamente un JSON que describe la jerarquía organizacional en un formato estructurado y legible.

Para mejorar la comprensión de esta jerarquía, se ha desarrollado una página web que procesa el JSON y presenta la estructura organizacional en forma de grafo dinámico. Esta representación gráfica muestra claramente cómo están estructurados y relacionados los diferentes empleados dentro de la organización. Cada nodo en el grafo representa a un empleado, y las conexiones entre nodos reflejan las relaciones jerárquicas entre ellos.

La visualización en forma de grafo proporciona una representación visual intuitiva que facilita la comprensión de la organización y permite a los usuarios analizar la estructura de manera efectiva. Cabe destacar que si un empleado no forma parte de ninguna relación, su nodo que lo representa podría haberse generado más lejos que el resto. Por lo tanto, para poder visualizarlo, es necesario mover el contenedor donde se encuentra el grafo.

Para acceder a esta funcionalidad ingresar a: D3 Hierarchical Graph Visualization

3.4. Dificultades Encontradas

Durante el desarrollo del lenguaje, se encontró una problemática significativa relacionada con la gestión de jerarquías múltiples en distintos proyectos. Inicialmente, el lenguaje permitía la creación y manejo de varias jerarquías de proyectos simultáneamente, permitiendo que un empleado pudiera estar asignado a múltiples proyectos, cada uno con su propia estructura jerárquica y relaciones de subordinación.

Esta flexibilidad, sin embargo, introdujo complejidades innecesarias y complicaciones en el manejo de las relaciones entre los empleados. Por ejemplo, un empleado asignado a varios proyectos podía tener diferentes roles y subordinados en cada uno, lo que generaba confusión al intentar eliminarlo de un proyecto específico sin afectar las jerarquías en los demás. Además, la coherencia y mantenimiento de las jerarquías se volvieron tareas difíciles y propensas a errores.

Para simplificar el uso y asegurar una representación más clara y coherente de la estructura organizacional y evitar que un único empleado tuviera variables por cada proyecto, se decidió eliminar del lenguaje el tipo de dato project. Este enfoque unificado eliminó las ambigüedades y facilitó la manipulación de la jerarquía, asegurando que cada empleado tuviera una posición y un conjunto de relaciones claramente definidas.

Otro problema surgió con una instrucción que permitía utilizar un empleado individual en un array de empleados como parámetro. Esta situación generó conflictos significativos durante el procesamiento de la entrada de datos por la ambigüedad del tipo.

La dificultad principal radica en la necesidad de distinguir claramente entre un empleado único y un conjunto de empleados cuando se procesa la entrada. Finalmente se determinó que se debe utilizar una lista de empleados y en caso de querer que sea un único empleado hacer que la lista solamente contenga a un empleado.

También surgió un problema con FLEX que había pasado desapercibido en la segunda entrega, ya que en el pattern para reconocer string había un [punct]+ y punct tenía incluido el símbolo “ por ende el programa no detectaba de forma correcta cuando finaliza el string en caso de haber varios strings diferentes.

Por último, se encontraron varios segmentation faults ocasionados por accesos a memoria no permitidos. Que se resolvieron principalmente con validaciones.

4. Futuras Extensiones

Una de las posibles extensiones futuras para el lenguaje es mejorar la funcionalidad de eliminación de empleados. Actualmente, al remover un empleado, todos los empleados a su cargo que no tienen otro jefe también son eliminados. Aunque permanecen como entes desconectados. Podría ser útil que se implemente una función para eliminar por completo a un empleado, liberando la memoria.

Sin embargo, en algunas situaciones, podría ser útil asignar los empleados subordinados del empleado removido a otros empleados en lugar de desarmar la jerarquía de los hijos del empleado removido. Esta funcionalidad adicional ofrecería mayor flexibilidad y control sobre

la estructura organizacional, permitiendo mantener la jerarquía y redistribuir las responsabilidades de manera más efectiva.

La implementación de esta característica podría incluir comandos adicionales para especificar a qué empleados se deben asignar los subordinados. Por ejemplo, al eliminar un empleado, se podría indicar un nuevo supervisor para los empleados que estaban bajo su cargo.

5. Conclusiones

Durante el desarrollo de este proyecto, experimentamos un proceso dinámico y desafiante que implicó ajustes continuos desde la primera hasta la segunda entrega. Estos ajustes fueron necesarios para mejorar la usabilidad y la experiencia del usuario, y en algunos casos estuvieron motivados por limitaciones técnicas específicas. El trabajo práctico nos brindó la oportunidad de aplicar los conceptos teóricos aprendidos en clase de manera práctica, aprovechando tecnologías como Flex y Bison. A través de esta aplicación directa, consolidamos nuestro entendimiento sobre lenguajes de programación y compiladores.

En resumen, este proyecto fue una experiencia formativa que nos permitió aplicar nuestros conocimientos académicos en un entorno práctico, enfrentando desafíos tecnológicos y consolidando nuestro aprendizaje sobre el desarrollo de software.