72.39 Autómatas, Teoría de Lenguajes y Compiladores - 1er Cuatrimestre 2024

DISEÑO E IMPLEMENTACIÓN DE UN LENGUAJE

The Tree Calculator (TTC)

Fecha de entrega: 20 de Junio de 2024

Integrantes

- Axel Castro Benza - 62358 - acastrobenza@itba.edu.ar
- Rocío D'Intino - 62341 - rdintino@itba.edu.ar
- Florencia Carrica - 62040 - fcarrica@itba.edu.ar
- Santos Galarraga - 62185 - sgalarraga@itba.edu.ar

Tabla de Contenidos

1. Introducción

El presente trabajo detalla el desarrollo de un compilador especializado destinado a la manipulación y análisis de árboles, una estructura de datos esencial en el ámbito de la informática. Los árboles, conceptualizados como grafos conexos, desempeñan un papel fundamental en numerosas aplicaciones, abarcando desde la organización jerárquica de datos hasta la representación eficiente de expresiones matemáticas y la implementación de algoritmos de búsqueda avanzados.

El objetivo primordial de este proyecto consiste en diseñar un lenguaje de programación que facilite la creación y manipulación de árboles de manera tanto intuitiva como eficiente. Este lenguaje no solo ofrecerá funcionalidades básicas como la inserción y eliminación de nodos, sino que también integrará características avanzadas diseñadas para operaciones sofisticadas sobre estructuras arbóreas.

Entre las capacidades previstas se incluyen operaciones de manipulación como la suma y resta de árboles, al igual que la visualización de su estructura. Además, el lenguaje estará equipado para realizar recorridos específicos y evaluar árboles de expresiones (EXPTrees), permitiendo así un análisis detallado y sistemático de la información contenida en estos árboles.

En resumen, este proyecto se orienta a desarrollar un entorno de programación especializado que no solo satisfaga las necesidades fundamentales de manipulación y análisis de árboles, sino que también proporcione herramientas avanzadas para explorar y trabajar con estructuras de datos complejas de manera eficiente y efectiva.

El desarrollo de este trabajo práctico se efectuó en el siguiente repositorio de github: https://github.com/AxelCastroo/TLA.git

2. Modelo Computacional

2.1. Dominio

Nuestro proyecto se enfoca en la manipulación y análisis de árboles, una estructura de datos fundamental en informática. Los árboles, en nuestro contexto, son grafos conexos acíclicos que se emplean ampliamente en diversas aplicaciones.

En nuestro entorno de desarrollo, un árbol se define por nodos que pueden tener cero o más hijos. Nuestro lenguaje facilita una variedad de operaciones sobre estos árboles, tales como la inserción y eliminación de nodos, así como recorridos in-order, pre-order y post-order. Además, ofrece capacidades para transformaciones estructurales y técnicas de balanceo mediante métodos como AVL y Red-Black.

Además de las operaciones básicas, nuestro lenguaje permite calcular métricas esenciales como la raíz del árbol, su altura, la profundidad o nivel de un nodo específico, el ancho del árbol y la cantidad total de nodos, entre otras métricas relevantes para análisis y manipulación avanzada de estructuras de árbol.

2.2. Lenguaje

Nuestro lenguaje se caracteriza por una sintaxis clara y accesible, pero optimizada para las operaciones sobre árboles. El lenguaje incluye construcciones para realizar recorridos en pre-order, in-order y post-order. Estos recorridos están diseñados para ser intuitivos y flexibles. Además de soportar BSTree y AVLTree para estructuras de datos eficientes, nuestro lenguaje también incluye soporte para Red-Black Trees (RBTree) y Expression Trees (EXPTree), proporcionando a los desarrolladores una gama completa de opciones para sus necesidades de programación sobre estructuras de árbol.

El lenguaje no solo proporciona capacidades avanzadas para la manipulación de estructuras de datos como árboles y la evaluación de expresiones matemáticas, sino que también permite una gestión flexible y dinámica de variables.

Además, permite la utilización de funciones bastante comunes en los distintos lenguajes mediante keywords (por ejemplo if, if-else, for, entre otros) con el fin de aumentar el alcance del lenguaje de programación.

En resumen, nuestro lenguaje proporciona una sintaxis eficiente para operaciones sobre árboles, soporte completo para diversas estructuras de datos eficientes como BSTree, AVLTree, RBTree y EXPTree, además de facilitar la gestión dinámica de variables y la

implementación de construcciones comunes de control de flujo, todo diseñado para maximizar la flexibilidad y la eficiencia en la programación.

3. Implementación

3.1. Frontend

Para facilitar la comprensión y análisis de las estructuras, nuestro lenguaje incluye la capacidad de representar visualmente los árboles utilizando Graphviz. Esta herramienta permite una descripción gráfica en lenguaje DOT, lo que es crucial para validar las operaciones realizadas y entender la estructura resultante de los árboles.

Respecto al análisis léxico, se utiliza Flex como generador donde los lexemas aceptados se encuentran en FlexPatterns.l. Luego, los tokens generados se obtienen del archivo FlexActions.c y en caso de ser necesario, se almacena el valor del lexema (caso de ser entero o booleano).

En cuanto al análisis sintáctico, se utiliza Bison para establecer las reglas de gramática y se puede encontrar en BisonGrammar.y. A su vez, las funciones generadas de este archivo se encuentran en BisonActions.c. Este genera structs que dependen de la regla gramatical en la que se encuentra que luego serán impresas por Generate.c. Además de esto, en este archivo se maneja la tabla de símbolos junto con type-checking.

3.2. Backend

El backend de nuestro proyecto está diseñado para proporcionar una generación de código eficiente y precisa, manejando la lógica compleja de la traducción del lenguaje de alto nivel a código ejecutable. Los logs y registros de errores son una parte integral del backend, permitiendo un desarrollo y mantenimiento efectivos del compilador.

Respecto a la tabla de símbolos, esta guarda el tipo de la variable y si fue o no instanciada. Esta implementación se realizó utilizando una tabla de hash localizada en el archivo hashMap.c. Con el fin de acceder correctamente a esta tabla, se deben utilizar los structs key y value cuya utilidad está implícita en los nombres.

En cuanto al type-checking, se crearon funciones (getExpressionType y getFuncitionCallType) que permiten obtener el tipo de dato. Luego, se realiza una validación en todas las asignaciones, expresiones y llamados de funciones con el fin de no generar conflictos de tipos de datos.

En caso de que se pueda generar el árbol de sintaxis sin ningún error, el trabajo práctico finaliza generando un archivo en lenguaje java llamado Main.java. En este archivo se encuentran todas las transformaciones de código desde el lenguaje creado a Java que puede observarse en el fichero informático denominado Generator.c. A su vez, este está acompañado de otros archivos que son necesarios a la hora de ejecutar ya que utilizan o implementan funciones heredadas.

3.3. Dificultades Encontradas

En cuanto al desarrollo del frontend, se encontraron algunos inconvenientes al iniciar el desarrollo del trabajo práctico. No se tenía una idea clara de qué elementos eran necesarios para crear un lenguaje propio por lo que la primera estructura presentada se asemejaba mucho al lenguaje Java. Tras obtener una devolución adecuada, se pudo adaptar la gramática más simple de entender y de compilar. A su vez, en esta etapa, hubo problemas en cuanto a la gramática ya que esta presentaba, entre otras cosas, la anomalía del shift-reduce la cual fue resuelta tras entender la causa.

Con respecto al backend, la principal dificultad fue la implementación de la librería Graphviz junto con su correcta utilización. Para esto se tuvo que investigar exhaustivamente, lo que llevó una gran cantidad de tiempo ya que ningún integrante poseía conocimientos sobre esta misma. Por otra parte, si bien no fue un inconveniente muy difícil de resolver, la validación de tipos de datos junto con la implementación de la tabla de símbolos requirieron considerable tiempo.

No obstante, la gran problemática encontrada fue la incorrecta definición de un tipo de dato (factor), la cual utilizaba la función “union{...}” y produjo un retraso relevante a la hora de entregar el trabajo práctico. Finalmente, se notó que al utilizar esta función se almacenaba incorrectamente el struct, provocando que, por ejemplo, no se pueda asignar un valor a una variable declarada, entre otros. Tras arreglar este inconveniente, se pudo avanzar a una mayor velocidad.

Un problema que se encontró fue el tiempo límite de entrega. Por falta de testeos durante el desarrollo del trabajo hubo una demora de entrega que no fue anticipada. Una vez solucionados estos problemas el proyecto fue entregado, pero fuera del tiempo límite.

4. Casos de Uso

A continuación se mostrará el diseño de código válido en el lenguaje.

- Variables

Las variables permitidas en el lenguaje de programación son las de tipo booleanas, enteras y de árbol. Estas últimas son las únicas que no pueden instanciarse con una asignación. A continuación, se mostrarán algunos ejemplos válidos:

```
Int integer = 44
Boolean bool
bool = true
AVLTree tree1
BSTree tree2
EXPTree tree3
(...)
```

- Operadores

Los operadores válidos son los que se encuentran en la mayoría de lenguajes de programación. Algunos de estos siendo:

```
exp1 | exp2
exp1 & exp2
!exp1
exp1 > exp2
exp1 <= exp2
(...)
```

- Estructuras de control

Las estructuras de control válidas son las que también pueden encontrarse en la mayoría de los lenguajes. Siendo estos:

```
for n in (1 to 10){
     (...)
}

if (exp1 >= exp2){
     (...)
}

if (exp1 >= exp2){
     (...)
} else {
```

```
    (...)
}
```

- Funciones de árbol

  A continuación se presentarán todas las funciones de árboles permitidas junto con su breve descripción:

```
Int x = 4
tree1 insert x                  /* Inserta el valor 4 */
tree1 remove x                  /* Remueve el valor 4 */
tree1 includes x                /* Pregunta si el valor 4
                                está en el árbol, retorna
                                true en caso afirmativo,
                                false sino */
tree1 height                    /* Obtiene la altura máxima
                                del árbol */
tree1 depth x                   /* Obtiene la altura del
                                nodo con valor entero 4 */
tree1 visualize                 /* Imprime el árbol en un
                                archivo .dot */
tree1 iterate in-order          /* Itera el árbol en
                                in-order */
tree1 iterate pre-order         /* Itera el árbol en
                                pre-order */
tree1 iterate post-order        /* Itera el árbol en
                                post-order */
expTree.calculate(x)            /* Calcula el valor final
                                de la expresión. Esta
                                función está reservada
                                únicamente para variable de
                                tipos EXPTree */
```

5. Futuras Extensiones

Una posible dirección futura para el desarrollo de nuestro proyecto podría ser la incorporación de estructuras que soporten grafos, dado que estos presentan similitudes significativas con las estructuras de árboles binarios. Implementar soporte para grafos abriría la puerta a la inclusión de algoritmos específicos de esta estructura, como el cálculo de caminos más cortos (Shortest Path), el algoritmo de Dijkstra, entre otros, lo que ampliaría considerablemente la funcionalidad y el alcance de nuestro lenguaje.

Asimismo, podría resultar beneficioso revisar y mejorar la definición de los árboles de expresión (Expression Trees) para permitir que acepten múltiples fórmulas, o incluso para que sean reutilizables en distintos contextos. Esta mejora incrementaría la flexibilidad y la utilidad del lenguaje en aplicaciones más complejas y variadas. A su vez se podrían añadir nuevas funciones avanzadas al lenguaje, tales como el cálculo de valores máximos y mínimos, entre otras operaciones matemáticas y algorítmicas. Estas adiciones no solo aumentarían las capacidades del lenguaje, sino que también proporcionarían a los desarrolladores herramientas más potentes y versátiles para sus proyectos.

En términos generales, estas mejoras no solo fortalecerían la robustez y la funcionalidad de nuestro lenguaje, sino que también lo harían más adaptable y útil para una gama más amplia de aplicaciones en el ámbito de la informática y la ingeniería de software.

6. Conclusiones

El desarrollo de este trabajo práctico especial representó una exploración profunda y multidimensional dentro del campo de la informática, abarcando diversas áreas que enriquecieron nuestro aprendizaje en distintas disciplinas de nuestra carrera. Como equipo, este proyecto nos permitió adentrarnos de manera significativa en el complejo ámbito del diseño de lenguajes de programación y compiladores, aspectos fundamentales para la creación de software avanzado y eficiente. Particularmente, nos enfocamos en el estudio detallado de las estructuras de árbol y sus operaciones, destacando su relevancia en la organización y manipulación de datos en múltiples contextos informáticos. A través de este proyecto, logramos implementar un lenguaje especializado que simplifica la gestión y análisis de estas estructuras, consolidando así nuestros conocimientos teóricos con aplicaciones prácticas concretas.

Esta experiencia no solo nos proporcionó habilidades técnicas, sino que también nos enseñó valiosas lecciones sobre la importancia de la planificación meticulosa, la resolución de problemas técnicos complejos y la colaboración efectiva en equipo. Enfrentamos desafíos significativos, como la corrección de errores en la definición de tipos de datos y la integración de herramientas externas como la librería Graphviz, lo cual demandó investigación exhaustiva y un enfoque disciplinado para superarlos.

En resumen, este proyecto ha fortalecido nuestra comprensión integral de la creación de compiladores y el diseño de lenguajes. Las bases sólidas establecidas aquí no solo son aplicables en contextos actuales, sino que también nos preparan para enfrentar desafíos futuros y explorar nuevas oportunidades en el dinámico campo de la informática y la ingeniería de software.

7. Referencias y Bibliografía