Área Académica de Ingeniería en Computadores

Programa de Licenciatura en Ingeniería en Computadores

CE3104 Lenguajes, Compiladores e Intérpretes

Grupo 2

Proyecto de Compiladores e Intérpretes
Documentación Externa

Realizado por:

Juan Antonio Solis Argüello #2018151673

Leonardo Guillen Fernandez #2019031688

Brayan Alfaro González #2019380074

José Julián Camacho #2019201459

Profesor:

Marco Hernández Vázquez

Fecha: 18 Enero 2020

Índice

1. Diagrama de Arquitectura. **2**

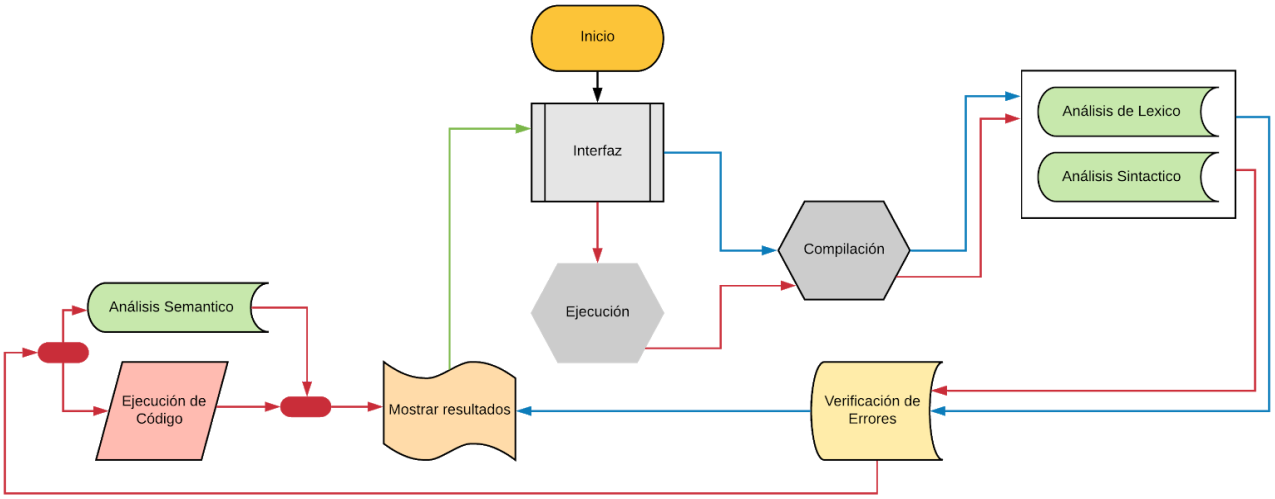
2. Problemas Conocidos. **2**

3. Problemas Encontrados. **3**

4. Conclusiones y Recomendaciones. **8**

5. Bibliografía **9**

1. Diagrama de Arquitectura.



2. Problemas Conocidos.

I. Uno de los problemas conocidos que no se pudieron arreglar debido al hecho de que la ventaja de la aplicación pierde lo dibujado cuando se mueve la ventana o cambia su tamaño, suponemos que se basa en no tener lo dibujado guardado, sin embargo no logramos encontrar una solución a este problema.

II. Otro problema no solucionado fue el no poder dibujar la tortuga sobre el punto actual donde se encuentra el punto de dibujo, este problema se basó en la utilización de la función paint que genera un dibujo automático en un ciclo, sin embargo se decidió implementar el llamado a cada función específica de dibujo y para como parámetro el Graphics del panel en el cual nos encontraríamos para dibujar, a su vez la opción del paint utiliza un Graphics general, el cual no solo dibuja en el panel deseado sino que en toda la ventana deformando lo dibujado con el Graphics del panel.

III. Se presentó un problema en relación con el cálculo de ciertas funciones aritméticas. Esto se produjo debido a que algunas de esas funciones, como la división de números enteros, tiene como resultado un número real y no necesariamente entero. Debido a que, el lenguaje LogoTec no soporta tipos de datos numéricos de punto flotante, cuando se realiza el cálculo de estas

funciones, debe aplicarse al resultado un redondeo para que el dato coincida con el tipo entero. Principalmente en dibujos con gran número de repeticiones, este problema de aproximación puede causar imperfecciones en las figuras. Esto debido a que, al utilizar valores redondeados, el error puede ser acumulativo y puede causar que lo se presente en pantalla difiera de lo que el usuario solicite.

3. Problemas Encontrados.

- **Problema 1**:

    a) Descripción detallada:

    La gramática del lenguaje inicialmente no permitía expresiones numéricas de la forma (a*b+c)*d, esto ya que el parser reconocía los elementos dentro del paréntesis más no reconocía la multiplicación de dicha expresión numérica por un elemento adicional. Es decir, el lenguaje se limitaba a reconocer expresiones numéricas de la forma a*b+c*d + …

    b) Intentos de solución sin éxito:

    No hubo intentos de solución sin éxito.

    c) Soluciones encontradas con su descripción detallada:

    La solución encontrada para el problema fue modificar la gramática para que esta se definiera de forma recursiva. De manera que el primer insumo de las expresiones numéricas, el cual correspondía a un número literal, tuviera una definición alterna, la cual corresponde a la expresión numérica final encerrada entre paréntesis. De esta manera se logró que el lenguaje reconociera expresiones más complejas.

    d) Recomendaciones:

    - Se recomienda prestar especial atención a la definición de la gramática, ya que los errores encontrados en esta son más

complejos de corregir una vez que se ha empezado a trabajar en otras etapas del análisis del compilador/intérprete.

- Se recomienda también realizar una amplia investigación bibliográfica antes de implementar una solución particular en cuanto al ámbito de compiladores/intérpretes. Esto ya que existen muchas soluciones conocidas a necesidades recurrentes, las cuales pueden ahorrar mucho tiempo en la implementación de la solución final.

e) Conclusiones:

- Se concluye que en cuanto al diseño de las gramáticas existen muchas posibles implementaciones, y lo mejor es buscar aquella que se adecue de mejor manera al lenguaje particular que se busca desarrollar.

- Se concluye así mismo que es importante realizar chequeos de calidad entre las diferentes etapas del desarrollo del compilador/intérprete.

f) Bibliografía:

antlr/grammars-v4. (2021). Recuperado el 1 de Enero 2021, from https://github.com/antlr/grammars-v4

- **Problema 2**:

a) *Descripción detallada:*

Inicialmente en el *visitor* del programa cada función de *visitor* retornaba un *Dato*, el cual contiene el objeto que se desea retornar y un valor numérico que indica el tipo del dato. Sin embargo, esta solución no funcionaba por dos razones: la primera es que en el mismo lenguaje el uso de listas implica que al visitarlas se deben de devolver múltiples valores, lo cual no era posible, y la segunda tiene que ver con la implementación propia de antlr en el método de *visitChildren*, el cual tiene una estructura para retornar los elementos retornados por los hijos de un nodo en formato de lista.

b) Intentos de solución sin éxito:

No hubo intentos de solución sin éxito.

c) Soluciones encontradas con su descripción detallada:

La solución encontrada a dicho problema consiste en cambiar el tipo de dato retornado por todas las funciones del *visitor*, de manera que retornen una lista de *Datos*, en lugar de retornar un solo *Dato*. De esta manera se pueden retornar todos los elementos de una lista, y tambíen puede funcionar de manera adecuada el método de *visitChildren*, el cual ahora se encarga de unir las listas retornadas por cada hijo.

d) Recomendaciones:

- Se recomienda tomar en cuenta todos los posibles casos al implementar una solución particular, de manera que no haya que replantear la solución a futuro.

e) Conclusiones:

- Se concluye que en las implementaciones tipo “visitor”, una solución muy útil para el retorno de valores, y el envío de valores entre funciones, corresponde al uso de listas que almacenen dichos valores. Esto puesto que sirven para retornar múltiples datos y pueden unirse varias de ellas.

f) Bibliografía:

Parr, T. (2011). *The definitive ANTLR reference*. Raleigh, NC: Pragmatic Bookshelf.

- **Problema 3**:

a) *Descripción detallada:*

Originalmente se había implementado una nueva estrategia de errores, denominada *StrictErrorStrategySpanish*, la cual sobreescribe el comportamiento de recuperación de errores de parseo que por

defecto hace que el parser de ANTLR trate de recuperarse de errores sencillos de sintaxis, de manera que la nueva estrategia evita la recuperación de errores. Sin embargo, por motivos desconocidos, al introducir un error sintáctico en la función de “inc” el parser entraba en un ciclo infinito.

b) Intentos de solución sin éxito:

Debido a que se desconocía la causa del problema, lo primero que se intentó fue volver a la estrategia de error que usa el parser por defecto. Sin embargo, esto no logró cambiar dicho comportamiento.

c) Soluciones encontradas con su descripción detallada:

La solución encontrada fue la de implementar la generación de una excepción del tipo *RuntimeError* cuando se encuentre cualquier error en la gramática. Esto se hizo modificando nuevamente la clase *StrictErrorStrategySpanish,* y en particular los métodos de recuperación de errores. La razón para implementar esta excepción es que se buscaba una que el parser no pudiera atrapar, de manera que interrumpiera completamente el proceso de parseo. Finalmente, se usó una estructura de try/catch al llamar al parseo, y así atrapar esta clase de excepciones.

d) Recomendaciones:

- Se recomienda realizar múltiples pruebas en los programas que se desarrollen para poder encontrar errores no triviales como el presentado en el actual ejemplo.
- Se recomienda evitar el uso de la estrategia de errores por defecto de ANTLR, ya que puede obviar el manejo de errores como el presentado actualmente.

e) Conclusiones:

- Se concluye que el compilador desarrollado es capaz de encontrar y manejar adecuadamente cualquier error de sintaxis, ya que la excepción agregada debido al presente error da

tranquilidad en cuanto al manejo de cualquier otro error desconocido.

f) Bibliografía:

Why is ANTLR omitting the final token *and* not producing an error?. (2017). Recuperado el 20 de Diciembre del 2020, de https://stackoverflow.com/questions/45025556/why-is-antlr-omitting-the-final-token-and-not-producing-an-error

- **Problema 4**:

a) Descripción detallada:

Al utilizar un valores numérico como strings estos presentaban un fallo y tomaban valores de entero aunque estos estuvieran dentro de comillas por lo tanto fue necesario hacer una revisión detallada a la gramática para lograr solucionar dicho error

b) Intentos de solución sin éxito:

Por la naturaleza del error revisamos sin éxito la implementación de valores numéricos como strings en la gramática dándonos cuenta que estaba declarada de forma incorrecta puesto que aceptaba números si contenían letras por lo tanto intentamos que recibiera números o letras este intento fue erróneo.

c) Soluciones encontradas con su descripción detallada:

La solución del problema fue más sencilla de lo que pensamos puesto que decidimos declarar en la gramática que todo carácter dentro de de las comilla fuera tomados como string

d) Recomendaciones:

Para problemas complejos a veces las soluciones son más simple de lo que pensamos por lo tanto se recomienda siempre buscar la solución más sencilla al problema para evitar sumar complejidad extra al programa

e) Conclusiones:

Se logró solucionar el problema para la sentencias de valores numéricos como strings de forma exitosa

f) Bibliografía:

https://github.com/antlr/antlr4/blob/master/doc/index.md

4. Conclusiones y Recomendaciones.

4.1 Conclusiones:

- Se logró diseñar e implementar satisfactoriamente un compilador capaz de realizar validaciones léxicas, sintácticas y semánticas, y generar errores; con el fin de graficar figuras en una interfaz gráfica a partir de código en el lenguaje de programación LogoTEC.

- Se concluye que ANTLR corresponde a una herramienta efectiva y útil para la implementación de un intérprete, ya que facilitó el desarrollo de las etapas de análisis léxico y análisis sintáctico de las instrucciones para el diseño de figuras.

- Los diversos errores léxicos, sintácticos y semánticos lograron ser desarrollados y pueden ser mostrados al usuario. A partir de ello, se concluye que ANTLR también resulta muy útil para la detección de errores léxicos y sintácticos. Además, se lograron detectar los distintos errores semánticos que pueden presentarse.

- La interfaz fue desarrollada en el lenguaje de programación Java mediante las bibliotecas gráficas AWT y Swing. Debido a ello, se concluye que Java es una muy buena opción para implementar un compilador o intérprete con ANTLR, e integrarlo con una interfaz gráfica amigable con el usuario.

- Se logró generar y mostrar en pantalla el árbol de parseo AST del código fuente, por lo que se reafirma que ANTLR es una herramienta muy buena para ayudar a la comprensión y al desarrollo de la etapa de análisis sintáctico de un compilador.

- Se concluye que las gramáticas libres de contexto (BNF) son de gran utilidad para establecer las reglas sintácticas de un lenguaje de programación, ya que la gramática para LogoTEC fue diseñada de esa manera y logra soportar la funcionalidad requerida.

4.2 Recomendaciones:

- Se recomienda el uso de ANTLR para futuros trabajos puesto que es una herramienta de alta calidad la cual opera sobre lenguajes, proporcionando un marco para construir reconocedores (parsers), intérpretes, compiladores y traductores de lenguajes a partir de las descripciones gramaticales de los mismos lo cual permitió el desarrollo e implementación de un intérprete o compilador mediante una gramática establecida

- Se recomienda una buena organización y comunicación grupal para que la implementación del trabajo ya que una buena organización permite que el trabajo realizado sea más eficiente y una buena comunicación permite que el trabajo sea mas organizado permitiendo así que la implementación del proyecto fuese satisfactoria.

- Se recomienda para la implementación de programas similares el uso de JAVA puesto que este lenguaje posee librerías potentes para diseñar Interfaces gráficas , manejo de objetos y es compatible con la herramienta ANTLR la cual fue muy importante para el desarrollo del programa.

- Se recomienda hacer lectura del libro The Definitive ANTLR 4 Reference en caso de tener alguna duda del manejo de esta herramienta ya que documenta de manera eficaz y permite evacuar dudas con respecto a la implementación del programa.

5. Bibliografía

Parr, T. (2011). *The definitive ANTLR reference*. Raleigh, NC: Pragmatic Bookshelf.

Parr, T. (2020). Antlr/antlr4. Recuperado el 13 de Diciembre de 2020, de https://github.com/antlr/antlr4

antlr/grammars-v4. (2021). Recuperado el 1 de Enero 2021, from https://github.com/antlr/grammars-v4