Rodrigo Merino de la Parra
A00836396

1. **Realiza una investigación relacionada a: *Generación automática de Analizadores de Léxico y Sintaxis (Scanners and Parsers)*.**
2. **Deberás documentar al menos 3 herramientas distintas, donde una de ellas será Lex&Yacc ó Flex&Bison (que son las más clásicas) y las otras 2 serán a tu elección y dependerá (principalmente) del lenguaje en el que estés planeando desarrollar nuestro mini-proyecto. Tu análisis deberá incluir cosas como:**
    a. **Plataforma y lenguaje base.**
    b. **Características básicas (formato requerido en las entradas, tipo de ejecución, etc.)**
    c. **Tipo de licenciamiento (costo, si lo tuviera)**
    d. **Herramientas teóricas en las que se basan: exp. regulares, métodos sintácticos empleados, etc.**
    e. **Tipo de interfaz.**
    f. **Facilidad para añadir código propio y cómo se haría.**
3. **¿Qué entregar? Un documento que describa las principales características de las herramientas analizadas (incluir referencias). Además, de preferencia, desarrolla un ejercicio en alguna de las herramientas (puede ser alguno de los ejemplos de la documentación).**

Flex & Bison

Flex (Fast Lexical Analyzer Generator) y Bison (una reimplementación de Yacc - Yet Another Compiler Compiler) son dos herramientas clásicas y potentes que suelen usarse en conjunto para crear compiladores y otros programas que interpretan lenguajes. Estas herramientas generan analizadores léxicos y sintácticos a partir de especificaciones para manejar entradas estructuradas y son comúnmente utilizadas en el desarrollo de compiladores. Flex se encarga del análisis léxico, convierte el texto de entrada en tokens. Bison se encarga del análisis sintáctico, toma los tokens que fueron generados y construye una estructura de árbol basada en unas reglas de gramática. Estas herramientas juntas simplifican la parte del “frontend” del compilador que requieren manejar entradas complejas.

Estas herramientas fueron diseñadas principalmente para sistemas operativos tipo Unix, pero están disponibles para todas las plataformas incluyendo MacOS y Windows a través de herramientas como MinGW o Cygwin.

Para utilizar estas herramientas el programador escribe las definiciones de tokens y reglas gramaticales en archivos con terminación .l para Flex y .y para bison, con lo que después estas herramientas convierten en código fuente de C/C++.

El proceso de ejecución es de dos fases. Primero, se compilan los archivos de Flex y Bison para poder generar código fuente .c y .h. Luego esto puede ser compilado utilizando el compilador de C/C++ para poder tener el ejecutable final.

Flex se basa en la teoría de los autómatas finitos. Convierte las expresiones regulares que definió el usuario en un Autómata Finito No Determinista (AFND), que luego transforma en un Autómata Finito Determinista (AFD) optimizado para el reconocimiento eficiente de tokens.

Bison utiliza un método de análisis sintáctico ascendente que es conocido como LALR(1) (Look-Ahead LR). Un método capaz de analizar una amplia clase de gramáticas libres de contexto.

La interacción principal de la interfaz es a través de la línea de comandos. Estas herramientas no cuentan con una GUI integrada. El desarrollador edita los archivos de estas herramientas en un editor de texto y utiliza los comandos flex y bison en la terminal.

Estas herramientas son muy flexibles, el código C/C++ puede ser añadido directamente en diferentes secciones de los archivos de entrada de Flex y Bison. En bison se pueden definir funciones, estructuras y la función main directamente en el archivo .y.

Flex y Bison son software libre y de código abierto. Flex se distribuye con la licencia BSD y Bison con la Licencia Pública General de GNU (GPL). Básicamente significa que son gratuitas y pueden ser utilizados sin costo en todo tipo de proyectos con tal de que se respeten las reglas de sus licencias.

ANTLR (ANother Tool for Language Recognition)

ANTLR es un generador de parsers moderno y popular que es utilizado en muchos proyectos academicos y comerciales para construir lenguajes, frameworks o herramientas.

ANTLR está escrito en Java, por lo que puede correr en cualquier plataforma que tenga la máquina virtual de java (JVM), esto incluye Windows, MacOS y Linux. Una de las mayores ventajas de generar analizadores en múltiples lenguajes de destino es que el programador elige el lenguaje de salida al utilizar esta herramienta. Puede ser Java, C#, Python, JavaScript, Go y otros.

El formato de entrada requiere de un único archivo de gramática con extensión .g4 que combina las reglas léxicas (para el scanner) y reglas sintácticas (para el parser). La sintaxis para definir la gramática de ANTLR es clara y expresiva.

El tipo de ejecución de ANTLR es un proceso también de dos pasos. Primero se ejecuta la herramienta de ANTLR sobre el archivo de gramática .g4, esto genera un conjunto de archivos fuente en el lenguaje de destino elegido. Después el segundo paso es que estos archivos se compliquen junto al resto del código del proyecto para crear la aplicación final.

ANTLR utiliza un metodo de analisis sintactico descendente llamado ALL(*) o All-Star. Este es un algoritmo potente ya que puede ver un número ilimitado de tokens hacia adelante para tomar decisiones de análisis, esto le permite manejar gramáticas mucho más complejas y ambiguas de forma más intuitiva. ANTLR también puede generar automáticamente árboles de analisis sintáctico (Parse Trees) que pueden ser recorridos utilizando patrones como Listener o Visitor.

La herramienta se utiliza a través de la línea de comandos, sin embargo sí existen algunos plugins para IDEs que proporcionan coloreado de sintaxis para las gramáticas, visualizaciones de árboles sintácticos y otros tipos de ayuda para hacer más sencillo el desarrollo.

La “filosofía” que sigue ANTLR es mantener la gramática limpia, separándose en acciones semánticas. La lógica de la aplicación se implementa utilizando patrones Listener o Visitor. Se crea una clase en el lenguaje de destino que hereda de una clase base generada por ANTLR y se implementan métodos que son invocados al entrar o salir de cada regla de la gramática. Esto resulta en un código más limpio y modular.

ANTLR es de código abierto y se distribuye bajo la Licencia BSD de 3 cláusulas. Esto significa que se permite su uso gratuito en proyectos de cualquier tipo con tal de respetar las reglas de la licencia.

PLY (Python Lex-Yacc)

PLY es una implementación de Lex y Yacc en Python. Una opción muy popular para personas que desarrollan proyectos de compiladores o intérpretes en Python.

PLY al ser una librería de python funciona en cualquier plataforma donde se pueda ejecutar Python. Está diseñado y escrito exclusivamente en Python.

PLY a diferencia de otras herramientas no utiliza archivos de especificación separados. Las reglas léxicas y sintácticas se definen dentro del mismo código Python. Los tokens se definen como variables o funciones en el script de Python cuyos nombres si tienen que seguir una convención específica como t_PLUS = r’\+’. Las reglas gramaticales se definen como funciones con docstrings que describen la regla como def p_expression_plus(p): ‘expression : expression PLUS term’.

PLY no tiene un paso de generación de código como tal, utiliza la introspección de python en el tiempo de ejecución para leer las instrucciones, leer las definiciones de los tokens y las docstrings de las funciones de la gramática, construyendo una tabla de análisis internamente en memoria.

PLY es una implementación de los algoritmos originales Lex y Yacc. Para el análisis léxico se utilizan expresiones regulares que generan un autómata finito. Para el análisis sintáctico se utiliza el método LALR(1).

No tiene una interfaz, es una librería de python. Se importa y se utiliza como cualquier otra librería de python. Pip install ply, import ply.lex as lex import ply.yacc as yacc.

Para añadir código es super sencillo ya que todo es código Python. Las acciones semánticas y reglas gramaticales son el cuerpo de las funciones de Python.

PLY es un software de código abierto que es distribuido bajo la licencia de BSD, permite su uso en todo tipo de proyectos con tal de que se respete la licencia.

Referencias

compilers:flex_bison [Skenz - How To Wiki]. (n.d.). https://www.skenz.it/compilers/flex_bison
ANTLR. (n.d.). https://www.antlr.org/
PLY (Python Lex-Yacc) — ply 4.0 documentation. (n.d.).https://ply.readthedocs.io/en/latest/ply.html#introduction%20https://www.antlr.org/%20https://webdiis.unizar.es/asignaturas/LGA/material_2004_2005/Intro_Flex_Bison.pdf%20https://www.cse.scu.edu/~m1wang/compiler/TutorialFlexBison.pdf