Tecnológico de Monterrey - Campus Monterrey

Tarea 2

Desarrollo de aplicaciones avanzadas de ciencias computacionales

Profa. Elda Guadalupe Quiroga González

Daniela Ramos García A01174259

08 de abril del 2025

Generación automática de Analizadores de Léxico y Sintaxis.

Lex & Bison

Son herramientas diseñadas para desarrolladores de compiladores e intérpretes, son ideales para aplicaciones que buscan patrones dentro de sus inputs o que usan un lenguaje. Permiten la creación del prototipo rápidamente, modificación fácil y un mantenimiento simple de la aplicación. Lex es un analizador léxico o scanner mientras que Bison es un analizador sintáctico o parser.

Plataforma y lenguaje base.

Ambas herramientas se ejecutan mediante una interfaz de línea de comandos o CLI y son multiplataforma, funcionando en sistemas operativos como Linux, macOS y Windows. Generalmente los sistemas que usan Linux ya tienen instalado Flex y Bison dentro de su sistema base. El lenguaje base para la generación del código es C, aunque también pueden adaptarse a C++ y recientemente se agregó la adaptación para Java.

Características básicas (formato requerido en las entradas, tipo de ejecución, etc.)

Un programa flex consta de tres secciones, separadas por %% líneas. La primera sección contiene declaraciones y la configuración de opciones. La segunda sección es una lista de patrones y acciones, y la tercera sección es código C que se copia al escáner generado, generalmente pequeñas rutinas relacionadas con el código de las acciones.

En la sección de declaraciones, el código dentro de %{ y %} se copia textualmente cerca del inicio del archivo fuente de C generado. En este caso, solo se configuran variables para líneas, palabras y caracteres.

En la segunda sección, cada patrón se encuentra al inicio de una línea, seguido del código C que se ejecutará cuando coincida. El código C puede ser una sentencia o posiblemente un bloque de varias líneas entre llaves, { }. (Cada patrón debe comenzar al inicio de la línea, ya que flex considera que cualquier línea que comience con un espacio en blanco es código que se copiará en el programa C generado).

Tipo de licenciamiento.

Bison: Bison es software libre; se puede redistribuir y/o modificar según los términos de la Licencia Pública General GNU publicada por la *Free Software Foundatio*n; ya sea la versión 3 de la Licencia o cualquier versión posterior.

Lex: El Gobierno de los Estados Unidos tiene derechos sobre esta obra en virtud del contrato n.º DE-AC03-76SF00098 entre el Departamento de Energía de los Estados Unidos y la Universidad de California.

Se permite la redistribución y el uso en formato fuente y binario, con o sin modificaciones, siempre que se cumplan las siguientes condiciones:

- Las redistribuciones del código fuente deben conservar el aviso de derechos de autor anterior, esta lista de condiciones y la siguiente exención de responsabilidad.

- Las redistribuciones en formato binario deben reproducir el aviso de derechos de autor anterior, esta lista de condiciones y la siguiente exención de responsabilidad en la documentación y/u otros materiales proporcionados con la distribución.

Ni el nombre de la Universidad ni los nombres de sus colaboradores pueden utilizarse para respaldar o promocionar productos derivados de este software sin autorización previa por escrito.

Herramientas teóricas en las que se basan.

Flex funciona a base de expresiones regulares, también conocidas como regex. Son los mismos patrones usados por herramientas como ed, vi y egrep para buscar texto. Un programa en Flex contiene una lista de expresiones regulares con acciones asociadas que se ejecutan al encontrar coincidencias. Flex convierte estas expresiones en un formato interno eficiente, basado en autómatas finitos deterministas (DFA). Primero crea un autómata no determinista (NFA) y luego lo transforma en un DFA optimizado, permitiendo analizar múltiples patrones al mismo tiempo con alta velocidad.

Bison, por su parte, genera analizadores sintácticos a partir de gramáticas libres de contexto, usando técnicas como el análisis LALR(1). Este método construye un analizador basado en una pila que procesa los tokens generados por Flex. Cada regla gramatical puede tener una acción escrita en C, que se ejecuta al reconocer la estructura correspondiente. Bison también permite añadir código personalizado, facilitando así la integración con el resto del programa.

Tipo de interfaz.

Ambas herramientas trabajan con la interfaz de la línea de comandos del sistema operativo.

Facilidad para añadir código propio y cómo se haría

El usuario puede incluir acciones en lenguaje C directamente asociadas a cada expresión regular dentro de las reglas del archivo .l; en Bison, se pueden insertar acciones semánticas en C dentro de las producciones gramaticales del archivo .y, así como definir funciones auxiliares o código de inicialización en secciones especiales del archivo. Esto permite integrar fácilmente la lógica personalizada del programador en el flujo del análisis léxico y sintáctico.

ANTLR

ANTLR (Another Tool for Language Recognition) es una herramienta moderna para la construcción de compiladores, intérpretes, analizadores de datos y generadores de código. Está diseñada para facilitar la definición de gramáticas completas que incluyan tanto el análisis léxico como el sintáctico en un solo archivo, permitiendo el desarrollo rápido de analizadores robustos. ANTLR es muy útil en aplicaciones que requieren interpretar lenguajes personalizados, DSLs o realizar procesamiento complejo de texto estructurado.

Plataforma y lenguaje base.

ANTLR es multiplataforma y su generador está escrito en Java, aunque puede producir analizadores en diversos lenguajes como Java, C#, Python, JavaScript y Go. Para su funcionamiento se requiere tener instalado Java, mientras que la integración con otros lenguajes depende del uso de runtimes específicos que permiten ejecutar el parser generado. El ejecutable principal (antlr4) se utiliza desde la línea de comandos.

Características básicas.

Las gramáticas se escriben en archivos con extensión .g4, que integran tanto las reglas léxicas como sintácticas. Una vez escrita la gramática, el usuario genera el código fuente ejecutando antlr4 archivo.g4, lo cual produce múltiples archivos de código en el lenguaje de destino. ANTLR organiza las reglas léxicas (tokens) y sintácticas en el mismo archivo, permitiendo un diseño modular. La ejecución del parser se realiza a través del runtime correspondiente, que debe instalarse como dependencia en el lenguaje objetivo. Por ejemplo, para Go, se utiliza el runtime oficial disponible en GitHub.

Tipo de licenciamiento.

ANTLR es software libre y se distribuye bajo la licencia BSD de 3 cláusulas, lo que permite su uso tanto en proyectos personales como comerciales sin restricciones de pago. Esta licencia es permisiva, permitiendo modificaciones y redistribución con pocas condiciones.

Herramientas teóricas en las que se basan.

ANTLR utiliza expresiones regulares para la definición del análisis léxico y se basa en gramáticas libres de contexto para el análisis sintáctico. Emplea un algoritmo predictivo conocido como LL(*), que es una extensión más potente del clásico LL(k), permitiendo una mayor flexibilidad y capacidad de predicción sin sacrificar eficiencia. Además, internamente construye árboles de análisis abstracto (AST) y estructuras auxiliares como parse trees y token streams para facilitar el procesamiento de los datos.

Tipo de interfaz.

ANTLR funciona desde una interfaz de línea de comandos, aunque puede integrarse fácilmente a entornos de desarrollo como Visual Studio Code, IntelliJ IDEA, Eclipse y Sublime Text, mediante plugins que ofrecen resaltado de sintaxis, visualización de árboles de análisis y navegación entre reglas gramaticales.

Facilidad para añadir código propio y cómo se haría

ANTLR permite insertar acciones semánticas en el lenguaje objetivo directamente dentro de las reglas gramaticales. Sin embargo, se recomienda utilizar el patrón de diseño basado en listeners o visitors, generados automáticamente por ANTLR. Estos permiten recorrer el árbol de análisis e implementar

funciones personalizadas para cada nodo, manteniendo una separación clara entre gramática y lógica. Esto facilita el mantenimiento del código y su reutilización en diferentes contextos.

Goyacc + lexer (text/scanner built in package)

Goyacc es una herramienta para generar analizadores sintácticos en Go, inspirada en Yacc. Debido a que no incluye un analizador léxico, se utiliza junto con un lexer personalizado, comúnmente escrito usando el paquete estándar text/scanner. Esta combinación ofrece una solución robusta para crear compiladores y analizadores completamente en Go, manteniendo la simplicidad y velocidad del lenguaje.

Plataforma y lenguaje base.

Goyacc y el paquete text/scanner están diseñados específicamente para el lenguaje de programación Go. Son multiplataforma y funcionan en sistemas que soportan Go, como Linux, macOS y Windows. Ambos forman parte del ecosistema oficial de herramientas del lenguaje Go, por lo que no requieren instalaciones adicionales.

Características básicas (formato requerido en las entradas, tipo de ejecución, etc.)

La gramática se define en un archivo .y, que contiene las reglas de producción con acciones escritas en Go. Estas reglas siguen el estilo clásico de Yacc. Por otro lado, el lexer se escribe en un archivo .go, y debe implementar una función Lex(lval *yySymType) int que escanee la entrada y devuelva los tokens correspondientes. Se puede usar text/scanner.Scanner para construir este lexer.

Tipo de licenciamiento (costo, si lo tuviera)

Goyacc y text/scanner están licenciados bajo la Licencia BSD, lo que significa que son de uso libre y pueden incluirse en proyectos personales y comerciales sin costo alguno.

Herramientas teóricas en las que se basan.

El analizador léxico se basa en el modelo de autómatas finitos. En este caso, la lógica de reconocimiento de tokens puede construirse manualmente o asistida por expresiones regulares simples. El parser generado por goyacc se basa en gramáticas libres de contexto y utiliza el método LALR(1), que emplea tablas sintácticas y una pila para procesar los símbolos de entrada de manera eficiente.

Tipo de interfaz.

Ambas herramientas se ejecutan desde la línea de comandos. Para generar el parser se ejecuta *goyacc archivo.y* Y luego se compila junto con el lexer y un archivo main.go que orquesta la ejecución del parser.

Facilidad para añadir código propio y cómo se haría

El programador puede añadir lógica personalizada directamente en las reglas de producción o en el lexer (Lex). También puede definir estructuras y funciones auxiliares dentro del mismo archivo .go o en archivos separados. Esta libertad de implementación hace que Goyacc y su lexer sean altamente flexibles, aunque requieren mayor conocimiento y esfuerzo en comparación con otras herramientas más automáticas como ANTLR.

Link a creación de ejemplo simple de parser y scanner con Goyacc y built in lexer: https://github.com/danielaramosgarcia/Desarrollo-de-aplicaciones-avanzadas---TTC3002B/tree/main/tarea2

Referencias.

1. Levine, J. (2009). *Flex & Bison: Text Processing Tools*. " O'Reilly Media, Inc.". https://books.google.es/books?hl=es&lr=&id=nYUkAAAAQBAJ&oi=fnd&pg=PR3&dq=flex+and+bison+&ots=VY6Alc0B4r&sig=oRGl55fBIK38rClofdnMOf8bMWY#v=onepage&q=flex%20and%20bison&f=false
2. Free Software Foundation. (s.f.). *GNU Bison*. GNU Operating System. Recuperado el 8 de abril de 2025, de https://www.gnu.org/software/bison/
3. Parr, T. (s.f.). *ANTLR (Another Tool for Language Recognition)*. Recuperado de https://www.antlr.org/
4. Golang.org. (s.f.). *Package text/scanner*. Recuperado de https://pkg.go.dev/text/scanner
5. Golang.org/x/tools. (s.f.). *goyacc command*. Recuperado de https://pkg.go.dev/golang.org/x/tools/cmd/goyacc
6. ANTLR GitHub. (s.f.). *ANTLR Runtime for Go*. Recuperado de https://github.com/antlr/antlr4