Universidad Zaragoza
1542

Trabajo Fin de Grado

ANAGRA 3.0: Herramienta para el estudio de gramáticas libres de contexto y técnicas de análisis sintáctico

ANAGRA 3.0: A tool for the study of context-free grammars and parsing techniques

Autora

Laura González Pizarro

Director

Joaquín Ezpeleta Mateo

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2023-24

RESUMEN

Un compilador es un programa informático que traduce un código fuente escrito en un lenguaje de programación de alto nivel a un código objeto, generalmente en lenguaje máquina o en un lenguaje intermedio. Esta traducción permite que el programa escrito por un programador sea ejecutado por un computador. El proceso de compilación consta de varias fases, y cada fase cumple una función específica para convertir el código fuente en un programa ejecutable.

El análisis sintáctico o proceso de *parsing* es una etapa fundamental de un compilador. Se encarga de analizar la estructura gramatical del código fuente y construir un árbol sintáctico que representa la jerarquía de las construcciones sintácticas. Su importancia radica en garantizar que el código fuente cumpla con las reglas sintácticas del lenguaje de programación.

El proyecto ANAGRA surgió debido a que no existía una herramienta completa para el estudio de gramáticas libres de contexto y técnicas de análisis sintáctico. Las versiones anteriores, ANAGRA 1.0 y ANAGRA 2.0, ofrecieron la posibilidad de realizar las operaciones de transformación más habituales sobre gramáticas y diferentes tipos de análisis sintáctico: descendentes como LL(1), y ascendentes como SLR, LR y LALR. Además de proporcionar un proceso interactivo de análisis sintáctico con visualización del árbol de sintaxis. Con el tiempo, la última versión, ANAGRA 2.0, basada en Java 1.3, perdió ciertas funcionalidades y soporte.

Este trabajo presenta ANAGRA 3.0, una reimplementación de la herramienta, siguiendo el estilo y características de las versiones anteriores, con el objetivo de agregar funcionalidades y mejorar la experiencia del usuario. ANAGRA 3.0 no solo preserva la funcionalidad de versiones anteriores, sino que también presenta nuevas características. Ahora, es capaz de transformar gramáticas a las formas normales de Chomsky y Greibach. Además, la herramienta puede guardar tablas de análisis sintáctico para generar analizadores por tabla y ofrece una simulación interactiva con la capacidad de poder avanzar y retroceder en el proceso de análisis, mostrando la evolución en la entrada, la pila y con su correspondiente árbol de sintaxis.

La interfaz se modernizó, haciéndola más intuitiva, y se solucionaron problemas de la versión anterior, como la representación de autómatas correspondientes al análisis ascendente donde ahora se pueden visualizar los arcos de un nodo a sí mismo y las etiquetas de dos nodos que se apuntan entre sí. Además, se implementaron ciertas operaciones concurrentes para evitar que la interfaz se quede congelada durante el análisis sintáctico en casos de gramáticas complejas. Se realizaron cambios en la presentación de resultados como la reubicación del log y el formato de presentación del autómata en texto, mejorando la legibilidad y usabilidad.

Estas mejoras, junto con la ampliación de funcionalidades, consolidan ANAGRA 3.0 como una herramienta robusta y versátil. De este modo, se cumplieron todos los objetivos propuestos, posicionando a ANAGRA como una valiosa herramienta de apoyo para el estudio de gramáticas libres de contexto y técnicas de análisis sintáctico.

Índice general

Capítulo 1

Introducción

1.1. Motivación y contexto

Un compilador es un programa informático que traduce un código fuente escrito en un lenguaje de programación de alto nivel a un código objeto, generalmente en lenguaje máquina o en un lenguaje intermedio. Esta traducción permite que el programa escrito por un programador sea ejecutado por un computador. El proceso de compilación consta de varias fases, y cada fase cumple una función específica para convertir el código fuente en un programa ejecutable. A continuación, se presenta un diagrama que abarca las fases esenciales de un compilador.

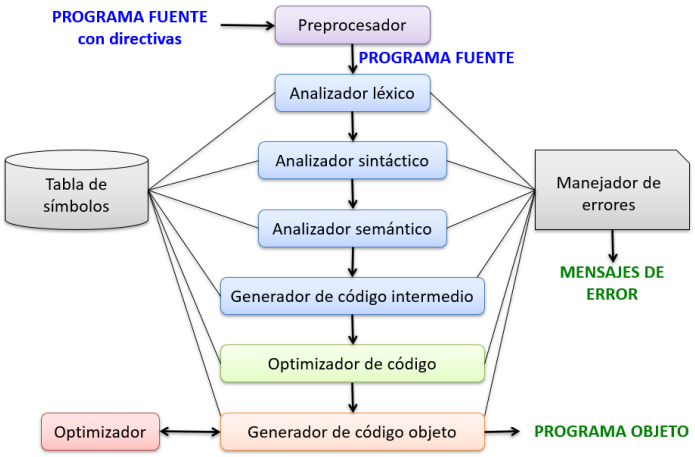


Figura 1.1: Diferentes partes para la creación de un compilador

En la Figura 1.1, se pueden observar las diferentes partes que conforman un compilador. A continuación, se realiza una descripción detallada de sus tres partes fundamentales: el análisis léxico, el análisis sintáctico y el análisis semántico.

El análisis léxico, también conocido como *scanning*, se encarga de examinar el código fuente, descomponiéndolo en tokens. Los tokens representan elementos léxicos, tales como palabras clave, identificadores, operadores y otros símbolos que conforman el lenguaje de programación. Este proceso es esencial para detectar errores léxicos en el código fuente y proporciona una base estructurada para la siguiente fase del proceso de compilación: el análisis sintáctico.

La segunda fase del compilador es el análisis sintáctico [1] , también conocido como *par-*

sing, este agrupa los tokens suministrados por el analizador léxico para reconocer frases. Se encarga de analizar la estructura gramatical del código fuente y construir un árbol sintáctico que representa la jerarquía de las construcciones sintácticas. Su importancia radica en garantizar que el código fuente cumpla con las reglas sintácticas del lenguaje de programación.

El análisis semántico examina el significado del código fuente para asegurar que cumpla con las reglas semánticas del lenguaje. El analizador semántico identifica posibles errores como incoherencias en el uso de variables, operaciones y funciones, así como la verificación de la conformidad del código con las reglas semánticas específicas del lenguaje.

Debido a que no existía una herramienta completa para el estudio de gramáticas y análisis sintáctico surgió el proyecto de "ANAGRA 1.0. Un entorno para el estudio de las fases de análisis en el desarrollo de traductores" y posteriormente evolucionó "ANAGRA 2.0 Diseño e implementación de un entorno multiplataforma de ayuda para el estudio de asignaturas de compiladores"[2]. Estas herramientas ofrecían más tipos de analizadores sintácticos, incorporando además de los analizadores LL(1)[3] y SLR[4], los analizadores LR y LALR [5], además de construir el proceso de análisis sintáctico interactivo siguiendo la evolución en la entrada y con su correspondiente árbol de sintaxis. Asimismo, se añadió la posibilidad de realizar operaciones de transformación sobre gramáticas y obtener información adicional de estas mediante el cálculo de los conjuntos Primero y Siguiente.

La ultima versión de ANAGRA fue implementada en el año 2000 utilizando la versión de Java 1.3, versión que ya no recibe ningún tipo de actualizaciones o soporte. Debido a esto la herramienta ha perdido ciertas funcionalidades, por esto, se decidió hacer una reimplementación de la herramienta "ANAGRA 3.0: Herramienta para el estudio de gramáticas libres de contexto y técnicas de análisis sintáctico", una nueva versión que mejore la experiencia de usuario y añada más funcionalidades.

1.2. Estado del arte

En el ámbito de las herramientas para el estudio de gramáticas y análisis sintáctico, cabe destacar que la disponibilidad de opciones exhaustivas es limitada. Se realizó un análisis detallado de algunas de las herramientas más destacadas, resaltando sus características y limitaciones. A continuación se describen las principales herramientas.

En primer lugar, JFLAP[6] es una de las herramientas para el estudio de conceptos relacionados con los lenguajes formales más populares. Implementada en Java, permite construir autómatas finitos no deterministas y autómatas de pila no deterministas, máquinas de Turing de cintas múltiples y sistemas L. Asimismo, permite realizar el análisis sintáctico de diferentes tipos de gramáticas, así como herramientas para la simulación de gramáticas y autómatas. Sin embargo, es importante tener en cuenta que en lo que respecta al análisis de gramáticas y al análisis sintáctico, JFLAP se limita a los analizadores LL(1) y SLR. Esto implica que, solo es posible realizar el análisis de una entrada, la herramienta no admite otro tipo de analizadores ni permite realizar operaciones de transformación sobre gramáticas.

Por otro lado, ANTLRWorks[7] es una herramienta para el estudio de gramáticas y su análisis. La funcionalidad de ANTLRWorks es parecida a la de la herramienta anterior debido a que solo permite realizar el análisis de una entrada para determinar si una entrada pertenece a la gramática, con la limitación de que únicamente permite realizar el análisis LL(1). Además, ANTLRWorks tampoco permite realizar operaciones de transformación sobre gramáticas.

Cabe resaltar la existencia de otras herramientas que abordan distintos enfoques sobre los analizadores sintácticos. Por ejemplo, CTPG [8], es una librería que toma una descripción de lenguaje en forma de código C++ y la convierte en un analizador LR(1) con una tabla y un analizador léxico de autómata finito determinista, que realiza el análisis sintáctico durante el tiempo de compilación. Por otro lado, EllErre [9] es un generador de autómatas LR, que permite la construcción de los automatas SLR, LR y LALR para una gramática dada. Asimismo, Parglare [10] representa un analizador diseñado para gramáticas LR y GLR. Sin embargo, es importante señalar que estas librerías ofrecen limitadas funcionalidades para el análisis sintáctico, además carecen de una interfaz gráfica.

1.3. Objetivos y alcance del proyecto

El proyecto ANAGRA surge de la necesidad de crear una herramienta con dos claros objetivos muy diferenciados. El primero siendo facilitar el estudio de los lenguajes y gramáticas libres de contexto. Con este propósito, la herramienta deberá ser capaz de reconocer gramáticas, realizar las operaciones más de transformación sobre gramáticas (ejemplos ...).

El segundo objetivo se centra en la implementación de una herramienta para el estudio de las técnicas del análisis sintáctico. De este modo, es necesario poder generar las tablas para el análisis asociadas a diferentes tipos de análisis sintáctico, además en caso que sea necesario, generar autómatas correspondientes. Asimismo, la herramienta debe poder simular entradas paso a paso, mostrando la evolución del análisis sintáctico incluyendo la evolución de la entrada y la pila, la secuencia de producciones, la construcción del árbol de sintaxis, y determinar si pertenecen o no al lenguaje.

En la siguiente lista se describen las tareas concretas que se han realizado durante la totalidad del trabajo.

1. **Análisis de requisitos del trabajo**: fase inicial del trabajo en la que se realiza un análisis de la herramienta anterior, se fija el alcance del proyecto y se realiza una lista de requisitos.

2. **Definición de gramática y funcionalidades**: durante esta etapa se define el concepto de una gramática libre de contexto. Por otra parte, se ha llevado a cabo la implementación de operaciones de transformación de gramáticas, además de herramientas auxiliares.

3. **Implementación de diferentes analizadores sintácticos**: en esta tarea se desarrollan los diferentes tipos de análisis sintáctico para una gramática. Estos generan tablas para el análisis que se utilizan para simular entradas de texto, y determinar si pertenecen al lenguaje o no.

4. **Integración de la interfaz**: a lo largo de esta etapa, se ha diseñado la interfaz gráfica de la herramienta, además, al mismo tiempo que se han integrado los módulos que componen su funcionalidad.

5. **Documentación y Manual de uso**: tarea en la que se genera la documentación del trabajo realizado y se proporciona un manual de uso a los usuarios futuros.

1.4. Metodología y herramientas

Respecto a las herramientas empleadas en este proyecto, cabe destacar que la base del desarrollo se ha llevado a cabo íntegramente en Python. Se ha llevado a cabo una reimplementación del proyecto anterior, ANAGRA, que estaba inicialmente implementado en Java. La elección de Python proporciona a los usuarios la capacidad de comparar los algoritmos a alto nivel con la implementación realizada, este aspecto no estaba tan accesible en la versión anterior debido a que Java es un lenguaje con mucha verbosidad.

Respecto al lenguaje a utilizar para la especificación de la herramienta se decidió que el fuera Bison, a diferencia de Yacc[11] que se utilizó en las versiones anteriores. El cambio de lenguaje se determinó debido a que Bison es una alternativa de código abierto más avanzada que mantiene compatibilidad con Yacc pero agrega funcionalidades adicionales y mejoras, además de ser el lenguaje utilizado en asignaturas anteriores.

En cuanto a la las librerías para el desarrollo de una interfaz gráfica en Python, existen diversas opciones, cada una con sus características. Entre las bibliotecas más destacadas se encuentran Tkinter [12], wxPython [13], PyQt [14], PySide [15]. Se optó por utilizar PyQt, para el desarrollo de la interfaz fundamentalmente debido a que es la biblioteca con mayor robustez y madurez. Además, la elección se ve respaldada por su exhaustiva documentación donde se encuentran una abundancia de ejemplos y recursos disponibles, lo que ha facilitado el desarrollo de la herramienta.

Existen diversas librerías para el análisis gramatical, entre las más destacadas se encuentran Yapps, PLY, Lrparsing, PlyPlus, APG, ANTLR. Se decididó por el uso de la librería PLY (Python Lex-Yacc) [16] para el análisis gramatical de la aplicación. Esto se debe, principalmente, a que está implementada también en Python, lo que facilita la integración con otros componentes del sistema.

El código implementado para este proyecto se encuentra gestionado en su totalidad bajo control de versiones mediante el uso de la herramienta Git. Además, la memoria del proyecto, también gestionado mediante control de versiones, fue elaborado con el sistema de composición de textos LaTeX.

El resumen de principales tecnologías y herramientas incluye las siguientes:

- Python (`https://www.python.org/`)
- Bison `https://www.gnu.org/software/bison/`)
- PyQt (`https://pypi.org/project/PyQt5/`)
- PLY (`https://www.dabeaz.com/ply/`)
- Git (`https://git-scm.com/`)
- LaTeX(`https://www.latex-project.org/`)

1.5. Estructura de la memoria

La memoria de este Trabajo de Fin de Grado se estructura en los siguientes Capítulos. En el Capítulo 1 se introduce y detalla la motivación subyacente que ha impulsado la ejecución de este proyecto, así como describir los objetivos y el alcance de la herramienta. Además de

exponer los diferentes enfoques metodológicos y herramientas consideradas.

El Capítulo 2 proporciona una explicación general sobre los conceptos de la informática vinculados a la herramienta desarrollada. Se presenta una breve introducción a las gramáticas libres de contexto y al análisis sintáctico, detallado en el análisis ascendente y descendente, así como de ejemplos concretos de analizadores como LL(1), SLR, LALR y LR.

Por otro lado, en el Capítulo 3 se describe la fase inicial del proyecto, en la que se realiza un detallado análisis de la versión anterior de la herramienta, las nuevas funcionalidades a añadir en el proyecto y un análisis de requisitos.

El Capítulo 4 muestra la implementación y organización de la herramienta. Donde se realiza un análisis detallado de cada una de las etapas del desarrollo junto a los módulos implementados.

En el Capítulo 5 se describen las principales características generales de la herramienta implementada, cada uno de los elementos que la componen junto a las funcionalidades que incorporan y las diferentes acciones que se pueden realizar.

Finalmente, en el Capítulo 6 se comentan las conclusiones obtenidas tras el diseño e implementación de la herramienta. Por último, se muestra un apartado con el cronograma de trabajo del proyecto y otro con posibles líneas futuras de trabajo, donde además se proponen mejoras a realizar con el fin de mejorar el funcionamiento de la herramienta.

Capítulo 2

Fundamentos previos

En este Capítulo, se establecen los fundamentos de los temas vinculados a la herramienta desarrollada, con el objetivo de simplificar la comprensión de los apartados siguientes. Se presenta una introducción a las gramáticas libres de contexto y al análisis sintáctico. Dentro de este último, se llevará a cabo un enfoque más detallado en el análisis ascendente y descendente, destacando sus principales tipos, como los métodos LL(1), SLR, LALR y LR.

2.1. Gramática

Los cuatro componentes que forman una gramática libre de contexto[17] o simplemente gramática son:

1. **Terminales**: conjunto finito de símbolos que forman las cadenas del lenguaje definido.
2. **No-Terminales**: conjunto finito de variables, donde cada una representa un lenguaje.
3. **Símbolo inicial**: representa el lenguaje definido. El símbolo inicial es el único no terminal que se utiliza para generar todos las cadenas del lenguaje.
4. **Producciones**: representan la definición recursiva de un lenguaje. Cada producción consiste en:
   a) Un no terminal que es definido por la producción. A este no terminal se le llama *cabeza* o *parte izquierda* de la producción.
   b) El símbolo $\rightarrow$
   c) Una cadena de cero o más terminales y no terminales. A esta lista se le denomina *cuerpo* o *parte derecha* de la producción, representa una manera de formar cadenas en el lenguaje de la parte izquierda. Al hacerlo, se dejan los terminales sin cambios y se sustituye cada no terminal por cualquier cadena que este en su lenguaje.

Una gramática se representa por sus cuatro componentes G = (N, T, S, P), donde N es el conjunto de no terminales, T los terminales, S el símbolo inicial y P el conjunto de producciones.

Las gramáticas libres de contexto tienen asociadas dos conjuntos, que son utilizados a la hora de realizar el análisis sintáctico. Antes de introducir los conjuntos, es crucial entender el concepto de **forma de frase**, que se define como cualquier cadena de símbolos terminales y no terminales.

- **Conjunto Primero**: PRI(a), donde "a" es cualquier forma de frase, es el conjunto de terminales que comienzan las cadenas derivadas de "a".

- **Conjunto Siguiente**: SIG(A), donde "A" es un no-terminal, es el conjunto de terminales que pueden aparecer inmediatamente a la derecha de A en alguna forma de frase.

2.2. Análisis sintáctico

El análisis sintáctico es el proceso para determinar cómo se puede generar una cadena de terminales por una gramática. Para esto, en cada paso se realiza una derivación, esto es, se reemplaza repetidamente cada no terminal por una de las partes derecha de sus producciones.

Existen dos tipos de analizadores que se distinguen por la forma en que se deriva el no terminal que se remplazará en cada paso. La elección se realiza de la siguiente manera, si la derivación es por la izquierda (*leftmost*) siempre se elige el no terminal más a la izquierda en cada oración. Por otro lado, si la derivación es por la derecha (*rightmost*), se elige el no terminal más a la derecha.

Un analizador sintáctico construye un árbol de análisis, en el que la raíz está etiquetada con el símbolo inicial, cada nodo interior corresponde a un no terminal, y cada hoja está etiquetada con un terminal o la cadena vacía (ϵ). Los hijos de un nodo son los símbolos (de izquierda a derecha) de una de las partes derechas de las producciones que tiene dicho nodo como parte izquierda. Otra manera de clasificar analizadores es según su manera de construir el árbol de sintaxis, existen dos formas: si se generan del nodo raíz a las hojas es un analizador descendente(*top-down*), mientras que si el árbol se genera desde las hojas hasta el nodo inicial, el analizador es ascendente (*bottom-up*). Una gramática es ambigua cuando existe más de un árbol de sintaxis asociado a una misma cadena de entrada.

Los analizadores sintácticos pueden tener k símbolos de *lookahead*, estos son los siguientes k símbolos de la entrada y se utilizan para decidir qué producción aplicar en el análisis. Su funcionamiento es el siguiente: cuando el nodo del árbol que se está analizando es un terminal, y este coincide con el símbolo de *lookahead*, se avanza en el análisis árbol y la entrada. El siguiente terminal en la entrada se convierte en el nuevo símbolo de *lookahead*, y el siguiente hijo en el árbol de análisis se considera.

2.2.1. Análisis sintáctico descendente (*top-down*)

El análisis descendente corresponde a la construcción del árbol de sintaxis comenzando por la raíz y creando los nodos hasta llegar a las hojas. Equivalentemente, el análisis descendente puede verse como derivar el símbolo más a la izquierda para una cadena de entrada.

En cada paso del análisis descendente, se determina la producción que se aplicará para un no terminal. Una vez elegida la producción del no terminal, el resto del proceso de análisis consiste en "hacer coincidir" los terminales en el cuerpo de la producción con los de la cadena de entrada.

Existen dos tipos de analizadores descendentes, los analizadores recursivos y los analizadores por tabla. Los analizadores recursivos descendentes traducen la gramática en procedimientos o funciones en el código fuente. Cada no terminal tiene asociada una función que intenta reconocer y derivar la cadena de entrada correspondiente. Mientras que los analizadores descendentes por tabla, se basan en el uso de tablas de análisis precalculadas. Estas tablas ayudan a determinar las acciones a realizar en función del símbolo de entrada y el estado actual del análisis.

Análisis LL(1)

Los analizadores LL(1) son analizadores descendentes por tabla, son también conocidos como analizadores sintácticos predicativos. La primera "L" en LL(1) implica escanear la entrada de izquierda a derecha, la segunda "L" para derivar el símbolo más a la izquierda de la entrada, y el "1" significa que se utiliza un símbolo de *lookahead* en cada paso para la toma decisiones en el análisis.

La tabla para el análisis LL(1) se construye a partir de los conjuntos Primero y Siguiente de la gramática. Las filas se indexan por los símbolos no terminales, mientras que las columnas corresponden a los terminales y al símbolo **$**, que indica fin de cadena. Cada elemento [A, t] de la tabla contiene la producción que se debe aplicar cuando se encuentra el no terminal A y t como símbolo de entrada.

Una gramática es LL(1) cuando en la tabla del análisis cada elemento es un error o un conjunto con una única producción. Algunas gramáticas pueden tener entradas con más de una producción, esto puede ser debido a que la gramática tenga recursión a izquierda o que sea ambigua. La implementación del algoritmo de construcción de la tabla de análisis LL(1) se encuentra en el Anexo B.

2.2.2. Análisis sintáctico ascendente(*bottom-up*)

Un análisis ascendente corresponde a la construcción de un árbol que comienza en las hojas y avanza hacia la raíz. Equivalentemente el análisis ascendente puede verse como derivar el símbolo más a la derecha para una cadena de entrada.

El análisis ascendente utiliza una pila que estará inicialmente vacía e irá apilando símbolos según vayan siendo analizados. En cada paso del análisis ascendente se puede realizar una de las siguientes funciones:

Desplazamiento(*shift*): operación de mover un símbolo de la entrada a la pila. Esto corresponde a crear un nodo hoja en el árbol de sintaxis.

Reducción(*reduce*): encontrar en la parte superior de la pila una parte derecha de una producción y sustituirla por su parte izquierda. Corresponde a que la parte izquierda de la producción sea la raíz y agrupe todos sus subárboles que son su parte derecha de la producción.

El analizador reconoce la cadena cuando la entrada ha sido consumida por completo y la pila contiene únicamente el símbolo inicial. El uso de estas funciones hace que los analizadores ascendentes sean también conocidos como analizadores desplazamiento/reducción. Existen tres técnicas para la construcción de analizadores desplazamiento/reducción, analizadores SLR, LR y LR, estos dependen de la complejidad en su desarrollo, del tamaño del analizador y de su potencia.

Los analizadores ascendentes construyen un autómata finito determinista donde cada estado almacena la información de los posibles **prefijos**, que son fragmentos de las partes derechas de las producciones gramaticales, recorridos hasta alcanzar dicho estado. Los estados representan conjuntos de configuraciones, una **configuración** de una gramática G es una producción de G con un punto en alguna posición de su parte derecha. A partir de los estados en la pila y del símbolo de entrada se puede determinar las acciones a ejecutar.

Las tablas para analizadores ascendentes están conformadas por dos partes, la tabla Acción y la tabla Ir A. Estas tablas cambian dependiendo de qué tipo de analizador desplazamiento/reducción se escoja debido a que se basan en el autómata del análisis.

La tabla Acción indexa las filas por los estados del autómata, mientras que las columnas corresponden a los terminales y el símbolo **$** que indica fin de cadena. Cada entrada en la tabla Acción indica indica qué acción llevar a cabo, esta puede ser: un desplazamiento, una reducción, la aceptación de la cadena o la señalización de un error en la entrada.

En cuanto a la tabla Ir A, se indexan las filas con los estados del autómata y las columnas con los no terminales. El elemento [i, A] indica a que estado conduce cuando el análisis se encuentra en el estado i y el no terminal A conducen al estado j.

Una gramática admite un análisis ascendente cuando en la tabla Acción cada elemento es una error o un conjunto con una única producción. La presencia de más de una producción en un elemento de la tabla indica la existencia de conflictos en el análisis, existen dos tipos de conflictos:

- **Conflicto desplazamiento/reducción**: se puede tanto realizar un desplazamiento como una reducción.
- **Conflicto reducción/reducción**: el analizador detecta en la cima de la pila partes derechas de más de una producción.

Análisis SLR

Los analizadores LR(0) o más bien conocidos como analizadores SLR(*simple-LR*) son el tipo de analizadores desplazamiento/reducción más sencillos de construir, los más pequeños y los menos potentes de los tres. La "L" de LR(0) implica escanear la entrada de izquierda a derecha, la "R" para derivar el símbolo más la derecha y el "0" indica que no se utilizan símbolos de *lookahead* en el análisis.

La colección canónica LR(0) proporciona la base para construir el autómata que se utiliza para tomar decisiones de análisis. Cada estado del autómata LR(0) representa un conjunto de elementos en la colección canónica LR(0). Para construir la colección canónica LR(0) de una gramática es necesario definir gramática aumentada y dos funciones, Clausura y Sucesor.

Si G es una gramática que tiene como símbolo inicial S, entonces G', la gramática aumentada para G, es G con un nuevo símbolo inicial S' y la producción S' $\rightarrow$ S. El propósito de esta nueva producción inicial es indicar al analizador cuándo debe detener el análisis y determinar si la entrada ha sido aceptada, es decir, la cadena se acepta cuando se reduce S' $\rightarrow$ S.

La función Clausura determina todos los elementos que pueden ser alcanzados directa o indirectamente desde un conjunto dado de elementos. Mientras que, la función Sucesor para un conjunto de elementos I y un símbolo de la gramática X, se define como la clausura del conjunto de todos los elementos A $\rightarrow \alpha.X\beta$ tal que A $\rightarrow \alpha X.\beta$ esté en I. La función Sucesor se utiliza para definir las transiciones en el autómata LR(0) para una gramática.

Los estados del autómata LR(0) corresponden a conjuntos canónicos LR(0) y la transición del estado I para el símbolo X las especifica la función Sucesor(I,X). Los autómatas ayudan con las decisiones de desplazamiento/reducción. Supongamos que la cadena X de símbolos gramaticales lleva el autómata LR(0) desde el estado inicial 0 a algún estado j. Entonces, se

realiza una operación de desplazamiento del siguiente símbolo de entrada "a" si el estado j tiene una transición con el símbolo "a". De lo contrario, se realiza una reducción, los elementos en el estado j dirán qué producción utilizar.

Análisis LR

El análisis LR canónico o simplemente LR es un tipo de análisis desplazamiento/reducción que toma un símbolo como *lookahead*, a diferencia del SLR que no utilizaba ninguno. Para la construcción del autómata, cada estado contiene una configuración y de un conjunto de símbolos de *lookahead*. En un estado con la configuración A $\rightarrow X_1...X_i.X_{i+1}...X_n$ y el conjunto de *lookahead* "a", implica que se han reconocido $X_1...X_i$, y se espera reconocer $X_{i+1}...X_n$. Además, después de haber reconocido $X_1...X_iX_{i+1}...X_n$ es posible encontrar "a".

Debido a este cambio en la creación del autómata, el análisis LR es más potente que el SLR, de hecho, los analizadores LR son los más potentes de todos los analizadores ascendentes. Sin embargo, es importante señalar que esta mayor capacidad conlleva asociado un aumento en el tamaño del autómata, ya que puede haber varias configuraciones LR para cada configuración SLR posible.

Análisis LALR

Los analizadores LALR(*lookahead-LR*) son una solución intermedia entre el analizador SLR y LR. Su autómata se construye como el analizador LR, pero agrupa aquellos estados que únicamente se diferencian en el símbolo *lookahead*. Por ello, se obtiene un autómata con el mismo número de estados que el del analizador SLR aún más potente debido al uso del conjunto de símbolos de *lookahead*.

Capítulo 3

Análisis de la herramienta

En este Capítulo se presenta el análisis realizado en la fase inicial del proyecto, este implica un análisis del estado de la versión anterior de la herramienta junto con sus limitaciones, así como de las nuevas funcionalidades a incorporar. Por último, se realiza un análisis de requisitos del proyecto.

3.1. Análisis de funcionalidades de la versión anterior

ANAGRA fue una herramienta muy completa y útil en su época, sin embargo, con el transcurso del tiempo y el avance en los lenguajes de programación junto con el desarrollo de nuevas librerías, y también reconociendo aspectos que no fueron abordados en su momento, han surgido dos inconvenientes notables:

El primero de ellos es que debido a que fue desarrollada con Java 1.3, versión que a día de hoy está obsoleta y ha perdido soporte. Esta situación impide una correcta visualización de ciertas partes de la interfaz, como son los autómatas asociados al análisis ascendente de una gramática.

El segundo inconveniente radica en la implementación secuencial de la herramienta, por ello, cuando se realiza el análisis ascendente de una gramática compleja la interfaz se congela, impidiendo al usuario realizar otras operaciones hasta que se complete el cálculo de las tablas del análisis, el cual puede prolongarse durante varios minutos. Esto genera la impresión de que la aplicación se ha quedado bloqueada, afectando la experiencia del usuario.

Adicionalmente, el tercer inconveniente se presenta en la representación gráfica de los autómatas asociados al análisis ascendente, se observa que no se mostraban los arcos de un nodo a sí mismo, por otra parte, cuando dos nodos se apuntaban entre ellos no se podía leer la etiqueta debido a que los textos de ambos arcos se sobreponían uno encima de otro. Del mismo modo, no se verificaban las precondiciones de las operaciones de transformación de gramáticas. Por lo tanto, si se intentaba realizar una operación en una gramática que no cumplía con las condiciones necesarias, la aplicación podía quedarse bloqueada en algunos casos, o mostrar resultados incorrectos en otros.

3.2. Análisis de nuevas funcionalidades y cambios respecto a la versión anterior

Este proyecto representa una implementación completamente renovada de la herramienta, manteniendo las líneas de diseño y funcionalidades de las versiones anteriores. Sin embargo,

se ha llevado a cabo una serie de acciones clave para mejorar y ampliar significativamente la herramienta. En primer lugar, se ha trabajado en la expansión de la funcionalidad de la herramienta, incorporando características adicionales que enriquecen su utilidad. Asimismo, se ha dedicado especial atención a mejorar la experiencia del usuario, implementando ajustes que promueven una interfaz más moderna e intuitiva. Además, se han abordado y corregido los problemas identificados en la versión anterior, asegurando así una mejora sustancial en el rendimiento de la herramienta. Estos esfuerzos combinados constituyen una evolución significativa en comparación con la versión previa.

La primera funcionalidad incorporará las operaciones de transformación de gramáticas para convertirlas en la forma normal de Chomsky[18] y de Greibach[19]. Que una gramática esté en una forma normal implica que las reglas de producción adoptan una estructura estandarizada y más simple, lo que hace que la gramática sea más fácil de analizar.

Por otro lado, la segunda permitirá guardar las tablas de los diferentes tipos de análisis sintáctico, siempre que la gramática no tenga conflictos durante el análisis. Estas tablas son de utilidad en caso de que el usuario desee construir un analizador sintáctico ascendente o descendente mediante tablas.

La tercera funcionalidad a desarrollar permitirá la simulación interactiva de una entrada para un analizador sintáctico pudiendo avanzar o retroceder en el proceso. Esto permitirá al usuario volver a estados anteriores del análisis sin tener que realizar una nueva ejecución de la simulación, como ocurría en la versión anterior debido a que solo se podía avanzar en el análisis.

Con el objetivo de mejorar la experiencia de usuario se ha modernizado la interfaz y se han corregido los problemas detectados en la versión anterior. Esta mejora incluye, hacer la interfaz más intuitiva y el uso de hilos[20] para así impedir que la interfaz se congele cuando se esta realizando el análisis de una gramática. Además, en esta versión se muestran los arcos de un nodo a sí mismo, también, modificar la localización de la etiqueta del arco cuando dos nodos se apuntaban entre ellos para que sea legible para el usuario. Asimismo, se hace una comprobación exhaustiva de las precondiciones de cada operación de transformación y en caso de que la gramática no cumpla con alguna de las condiciones, se le notifca al usuario cual.

Sumado a ello, en esta versión actualizada se han propuesto algunos cambios con respecto a la versión anterior. El primero de ellos consiste en la reestructuración del formato de presentación del autómata del análisis ascendente en texto siguiendo un estilo similar al utilizado por Bison, haciéndolo más legible y comprensible para el usuario, ya que muestra más información.

Otra modificación de esta versión afecta al log, ventana que muestra las operaciones que se han realizado en la aplicación, que ahora se encuentra ubicado en un Menú dentro de la pestaña Ayuda. Este cambio ha contribuido a una interfaz más limpia y estéticamente agradable, además, el concentrar la visualización del log en una acción específica del menú permite que la interfaz sea más intuitiva y centrada en las funciones de la aplicación.

El uso de Python como lenguaje para la implementación de este proyecto proporciona a los usuarios la capacidad de comparar los algoritmos a alto nivel con la implementación realizada, este aspecto no estaba tan accesible en la versión anterior debido a que Java es un lenguaje con mucha verbosidad. En la Figura 3.3 se muestra una comparativa de del pseudocódigo del algoritmo de eliminación de símbolos no alcanzables y la implementación

realizada en Python.

```
Algoritmo removalUnreachableTerminals:
Require: G=(N, T, P, S) t.q L(G) != ø

  old := {}
  new := {X in (N union T) | S-> aYb in P} union {S}

  While old != new
    old := new
    new := old union {Y in (N union T) | A-> aYb in P, A in old}
  End While

  N' := new intersection N
  T' := new intersection T
  P' := {A->w in P | A in N', w in (N' union T')*}
```

Figura 3.1: Pseudocódigo del algoritmo

```
def removal_unreachable_terminals(grammar):
  gr = grammar.copy()
  old = set()
  new = set(smb for prod in gr.prods[gr.initial_tok]
          if prod is not None for smb in prod) | set(gr.initial_tok)

  while old != new:
      new_smbs = new.difference(old)
      old = new

      for smb in new_smbs & gr.non_terminals:
          new |= set(smb for prod in gr.prods[smb]
                     if prod is not None for smb in prod)

  gr.terminals &= new
  unreachable_smbs = gr.non_terminals.difference(new)
  gr.non_terminals &= new

  gr.prods = {key: value for key, value in gr.prods.items()
                                    if key not in unreachable_smbs}

  return gr
```

Figura 3.2: Algoritmo en Python

Figura 3.3: Comparativa Pseudocódigo e implementación del algoritmo eliminación de símbolos no alcanzables

3.3. Análisis de requisitos

Tras el análisis de la herramienta anterior junto a su funcionalidad y problemas asociados, y la definición de nuevas características, se realizó una lista con los requisitos de la aplicación. Es importante señalar que esta lista experimentó modificaciones a lo largo del desarrollo del proyecto. Inicialmente, se contempló únicamente la implementación del analizador LL(1). Sin embargo, tras su implementación, se consideró extender la funcionalidad para incluir el analizador SLR, y finalmente se añadieron los analizadores LR y LALR.

Los requisitos definidos tras el proceso se resumen en la Tabla 3.1. Para cada requisito se muestra un código y una pequeña descripción.

Código	Descripción
	Usuario
RF-01	El usuario deberá poder editar fuentes Bison con gramáticas en un editor amigable, con las características básicas de búsqueda (buscar, reemplazar), edición(copiar, cortar, pegar, borrar, seleccionar todo) y formato(cambiar fuente, color, espacios de tabulación).
RF-02	El usuario deberá poder tanto abrir como guardar archivos con gramáticas, seleccionándolas a lo largo del sistema de ficheros de la máquina donde se ejecute.
RF-03	El usuario deberá poder cambiar el idioma de la aplicación cuando lo desee.
RF-04	El usuario deberá poder calcular el conjunto Primero de una gramática.
RF-05	El usuario deberá poder calcular el conjunto Siguiente de una gramática.
RF-06	El usuario deberá poder calcular el conjunto Primero de una forma frase de una gramática.
RF-07	El usuario deberá poder eliminar los símbolos no derivables de una gramática.
RF-08	El usuario deberá poder eliminar los terminales no accesibles de una gramática.
RF-09	El usuario deberá poder eliminar las producciones ϵ de una gramática.
RF-10	El usuario deberá poder eliminar la recursividad a izquierda de una gramática.
RF-11	El usuario deberá poder eliminar los ciclos de una gramática.
RF-12	El usuario deberá poder factorizar a izquierda la gramática.
RF-13	El usuario deberá poder transformar la gramática a forma normal de Chomsky.
RF-14	El usuario deberá poder transformar la gramática a forma normal de Greibach.
RF-15	El usuario deberá poder calcular las tablas de análisis LL(1), SLR, LALR y LR.
RF-16	El usuario deberá poder guardar las tablas del análisis (LL(1), SLR, LALR, LR) en un fichero JSON.
RF-17	El usuario deberá poder calcular el autómata del análisis ascendente (SLR, LALR y LR) y acceder a la información de cada uno de los nodos. También, podrá visualizar la información del autómata en texto plano.
RF-18	El usuario deberá poder simular una entrada LL(1), SLR, LALR o LR interactivamente (pudiendo avanzar y retroceder) donde se mostrará la evolución de la pila entrada, la secuencia de producciones aplicadas, y el árbol de sintaxis.
	Software
RF-19	La herramienta ha de ser multiplataforma
RF-20	La herramienta ha de ser multilenguaje, es decir, todos sus textos estarán disponibles en varios idiomas. El idioma se podrá cambiar fácilmente desde un menú dedicado a ello.
RF-21	La herramienta ha de poder compilar gramáticas descritas en el lenguaje Bison, con la salvedad de que no contengan bloques de código.
RF-22	La herramienta dispondrá de un editor, donde se mostrará la gramática. Este contendrá una barra de estado donde se indicará el estado de la fuente activa, si está en modo de lectura o escritura, y la línea y columna donde está posicionado el cursor. Contendrá una serie de menús mediante los cuales el usuario podrá acceder a todas las funcionalidades de la aplicación descritas anteriormente.
RF-23	La herramienta comprobará las precondiciones de las operaciones de transformación antes de realizarlas, en caso de no cumplir las condiciones se indicará al usuario cuáles.

Cuadro 3.1: Análisis de requisitos.

Capítulo 4

Implementación de la herramienta

4.1. Estructura general del sistema

El nuevo ANAGRA cuenta con una estructura modular que desacopla la funcionalidad de la interfaz de la herramienta, siguiendo la estructura del patrón de arquitectura Modelo-Vista-Controlador(MVC) [21]. Por esto, es posible utilizar ANAGRA como ejecutable o como librería, permitiendo así que pueda ser integrada en otros sistemas.

Cada módulo implementa una etapa en la que se realizan tareas independientes del resto, partiendo de la definición de la gramática hasta la simulación sintáctica de esta. Cada módulo funcional dentro de ANAGRA cuenta con su contraparte específica en la capa de interfaz. Las etapas definidas son las siguientes:

- **Etapa de implementación de gramáticas**: en esta etapa se define la estructura de datos de una gramática, se implementan todas las operaciones de transformación sobre gramáticas. A su vez, también se desarrollan herramientas auxiliares como el cálculo del conjunto Primero y Siguiente sobre una gramática.
- **Etapa implementación análisis sintáctico**: a partir de una gramática se han desarrollado los analizadores sintácticos. En el caso del análisis descendente solo se calcula la tabla de análisis, mientras que para el análisis ascendente, se calculan las tablas Acción e Ir A, junto con la construcción del autómata asociado. Asimismo, se ha implementado la simulación de entradas a partir de las tablas del análisis de una gramática.
- **Etapa integración de la interfaz**: en esta etapa se implementa toda la interfaz gráfica y funcionalidad de la herramienta junto a la integración de los módulos el gramáticas, análisis y la simulación.

4.2. Etapa de implementación de gramáticas

En esta primera etapa se define el concepto de gramática libre de contexto, esto incluye sus estructuras de datos y las funciones asociadas como cargar una gramática a partir de un fichero Bison o el cálculo del conjunto Primero y Siguiente. Igualmente, de los algoritmos para las diferentes operaciones de transformación, así como, de formas normales. A continuación se muestran los módulos implementados en esta etapa:

- **Gramática**: este módulo representa el concepto de gramática. Esta compuesto por un conjunto de símbolos terminales, otro de no terminales, un símbolo inicial y por un conjunto de producciones. Ofrece las operaciones de transformación y formas normales descritas en el Capítulo 3. Así como del cálculo del conjunto Primero y Siguiente.

- **Conjuntos**: este módulo representa gráficamente él concepto de gramática, en el se implementan las ventanas emergentes cuando se calcula el conjunto Primero, conjunto Siguiente y conjunto Primero de una forma de frase.

- **bisonlex y bisonparse**: archivos implementados para la abstracción de una gramática en un fichero Bison. Este proceso se explica detalladamente en la siguiente sección.

4.2.1. Abstracción de una gramática en un fichero Bison

El proceso consiste en la abstracción de los diferentes parámetros de la gramática de un fichero Bison a partir de la generalización de la gramática de este lenguaje, para ello se utilizó la librería PLY(Python Lex-Yacc), una herramienta para la construcción de compiladores lex y yacc.

PLY consta de dos módulos separados: lex.py y yacc.py. El módulo lex.py se utiliza para dividir el texto de entrada en una colección de tokens especificados por un conjunto de reglas de expresiones regulares. Mientras que yacc.py se utiliza para reconocer la sintaxis del lenguaje que ha sido especificada en forma de una gramática libre de contexto.

La entrada se compone por dos ficheros, el primero, se definen los tokens del lenguaje mediante expresiones regulares, además de las acciones a realizar cuando se encuentra cada token. En el segundo, se definen las reglas gramaticales utilizando la notación BNF(Backus-Naur Form)[22], además de las acciones a realizar cuando se aplica cada regla gramatical. Por ello, en el primero bisonlex, se indicaron los diferentes símbolos de Bison. Mientras que en el segundo, bisonparse, se definieron las reglas de Bison y las acciones para ir completando los diferentes parámetros de una gramática. Así, una vez finalizada la compilación se podrán abstraer, en una estructura de datos, los diferentes parámetros de cualquier gramática escrita en Bison.

4.3. Etapa del análisis sintáctico

En esta etapa se han implementado los algoritmos necesarios para realizar el análisis sintáctico de una gramática. Esto incluye los algoritmos para el cálculo las tablas de análisis para cada tipo de analizador, así como el cálculo del autómata asociado en el caso del análisis ascendente. Además de los algoritmos utilizados para la simulación de cadenas a partir de las tablas del análisis. A continuación se muestran los módulos implementados en esta etapa:

- **Análisis LL(1)**: módulo que implementa las funciones del análisis LL(1), estas son, el cálculo de la tabla de análisis LL(1), la comprobación de que la gramática es LL(1), y la simulación de una entrada.

- **Analizador ascendente**: este módulo implementa las operaciones comunes en todos los tipos de analizador ascendente, estas son, expandir gramática y comprobar que no tiene ningún conflicto, y el algoritmo de simulación de una entrada.

- **Análisis SLR**: módulo que implementa las funciones Clausura, Sucesor del análisis SLR. Además del cálculo de configuraciones canónicas SLR, tabla Acción e Ir A, y el autómata correspondiente al análisis SLR.

- **Análisis LR**: módulo que implementa las funciones Clausura, Sucesor del análisis LR. Además del cálculo de configuraciones canónicas LR, tabla Acción e Ir A, y el autómata correspondiente al análisis LR.

- **Análisis LALR**: este módulo implementa las funciones Clausura, Sucesor del análisis LALR. Además del cálculo de configuraciones canónicas LALR, tabla Acción e Ir A, y el autómata correspondiente al análisis LALR.
- **Tablas**: este módulo implementa las ventanas emergentes de la tabla del análisis LL(1), las tablas Acción e Ir A, la gramática expandida y el autómata en formato texto correspondiente al análisis ascendente.
- **Autómata**: módulo que representa gráficamente el autómata correspondiente al análisis ascendente.
- **Simular**: este módulo implementa la ventana que se muestra al simular una entrada, además de toda la lógica detrás del análisis interactivo.
- **Árbol**: módulo que representa gráficamente el árbol de sintaxis generado al realizar el análisis sintáctico.

4.4. Etapa integración de la interfaz

En esta última etapa se desarrolla todo el proceso de integración de la interfaz gráfica de la herramienta junto a los módulos descritos en las secciones anteriores. Este paso abarca desde la implementación de las funcionalidades de la aplicación hasta el tratamiento de posibles errores.

El tratamiento de errores en la herramienta consiste desde la gestión de errores tanto léxicos como sintácticos de gramática, como de entradas de texto introducidas por el usuario no válidas, entre otras. Además, se incorporan mensajes auxiliares informativos para orientar al usuario a lo largo del proceso, como notificar si la gramática introducida no produce ningún lenguaje o, si una cadena es reconocida por la gramática o no. Los módulos implementados en esta fase:

- **Utils**: este módulo implementa las ventanas relacionadas con la funcionalidad de la aplicación que no están directamente relacionadas con el análisis de una gramática, estas son, la ventana de búsqueda de una palabra en la gramática, el remplazamiento de una palabra por otra en la gramática, la ventana de log de la herramienta, la ventana para introducir la entrada a simular y la barra de progreso que se muestra cuando se esta realizando el análisis sintáctico de una gramática.
- **Main**: modulo principal de la aplicación que integra toda la funcionaliad de la herramienta junto a la interfaz gráfica.

Capítulo 5

Resultados

En este Capítulo se describen las principales características generales de la herramienta implementada, así como cada uno de los elementos que la componen junto a las funcionalidades que se incorporan. Junto con las diferentes acciones que se pueden realizar.

5.1. Descripción general de la herramienta

La interfaz de esta nueva versión de ANAGRA se ha diseñado siguiendo el mismo estilo que la de su versión anterior. La interfaz esta compuesta por un menú situado en la parte superior de la ventana con los submenus Archivo, Editar, Buscar, Texto y Ayuda, que se detallarán en las siguientes subsecciones. Una barra de estado situada en la parte inferior de la ventana que muestra el estado de la fuente activa, si está en modo de lectura o escritura, la línea y columna donde está posicionado el cursor. También, un editor en el que se encuentra la gramática que se podrá editar solo si se esta en modo escritura. La Figura 5.1 muestra la interfaz de la herramienta, mientras que la Figura 5.2 muestra las principales opciones.

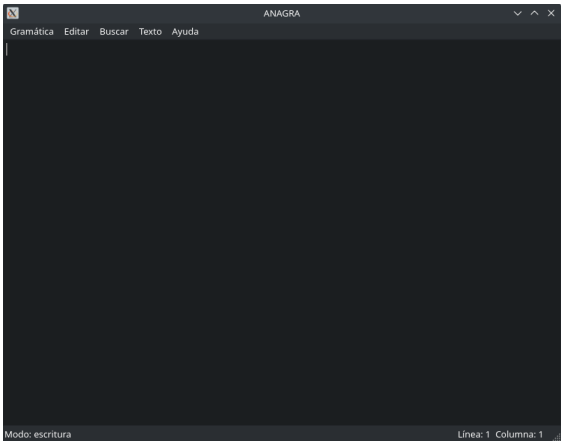


Figura 5.1: Interfaz de usuario.

Figura 5.2: Opciones de la aplicación.

ANAGRA acepta gramáticas escritas con la sintaxis de Bison, detecta errores en la misma, informando de ello mostrando un mensaje señalando el carácter, la línea y la columna con el error, y colocando el cursor sobre la línea en la que ha sido detectado. A su vez, cuando una transformación es aplicada sobre una gramática, la gramática resultante se abre en una nueva ventana, de manera que el usuario pueda compararlas y trabajar independientemente con ambas. Dispone, además, de otras facilidades de edición, como son la configuración en varios idiomas o el formateado automático de la fuente de la gramática

5.2. Menú Archivo

Las acciones recogidas en el menú Archivo son las permiten al usuario abrir una nueva ventana, abrir un fichero con una gramática, editar la gramática previamente aceptada, aceptar la gramática introducida, guardar la gramática en un fichero y cerrar la aplicación. Tanto si se acepta la gramática como si se abre de un fichero se comprueba que tenga la sintaxis de una gramática de Bison, en caso de que haya algún error sintáctico o léxico se informa de ello, mostrando un mensaje que señala el carácter, la línea y la columna, y colocando el cursor sobre la línea en la que ha sido detectado. En caso contrario, el editor cambiará a modo lectura, impidiendo que se pueda editar, y se añadirán las opciones Herramientas, Transformaciones, Análisis y Simular al menú. La Figura 5.3 muestra las ventanas tras aceptar una gramática sin errores.

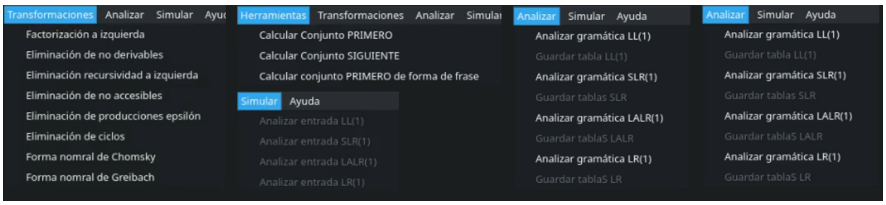


Figura 5.3: Opciones específicas de la aplicación para gramáticas.

5.3. Menú Editar

Las acciones del menú Editar permiten copiar, cortar, pegar y borrar el texto seleccionado, además de seleccionar todo el texto en el editor. Cabe destacar que estas acciones pueden realizarse mediante las combinaciones de teclas habituales.

5.4. Menú Buscar

En el menú Buscar se permite al usuario buscar una palabra en el editor o remplazarla otra. La búsqueda que hace sobre la gramática diferencia entre mayúsculas y minúsculas. Para buscar una palabra en la gramática, primero, se abrirá una ventana donde el usuario podrá introducir la palabra a buscar, cuando pulse el botón de aceptar se subrayarán en verde las coincidencias, mientras, que en caso de no haber ninguna coincidencia, se mostrará un mensaje indicándolo. La Figura 5.4 muestra el proceso de buscar una palabra en el texto.

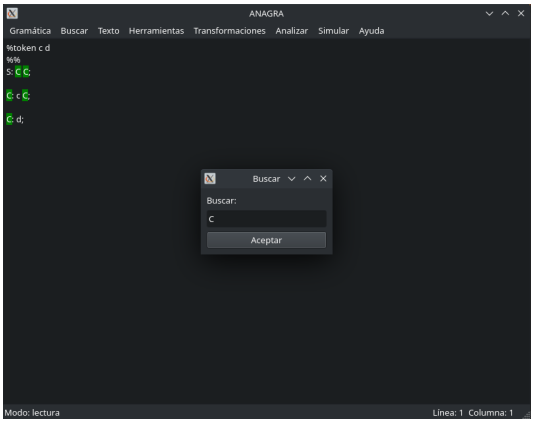


Figura 5.4: Resultados tras la búsqueda

Para remplazar una palabra por otra en la gramática, primero, se abrirá una ventana donde el usuario podrá introducir la palabra a buscar y aquella por la que la quiera cambiar, cuando pulse el botón de aceptar se remplazarán todas las coincidencias encontradas, mientras, en caso de no haber ninguna coincidencia se mostrará un mensaje indicándolo.

5.5. Menú Texto

Dentro del menú Texto se realizar las acciones de cambiar el color y fuente de la letra, el espacio de tabulación, el idioma, el formato en el que se muestra la gramática y guardar las preferencias. Cuando se guardan las preferencias, la siguiente ejecución de la aplicación aparece con las preferencias indicadas. En la Figura B.11 se muestran las ventanas emergentes para cambiar el color o fuente de la letra, o los espacios de tabulación.

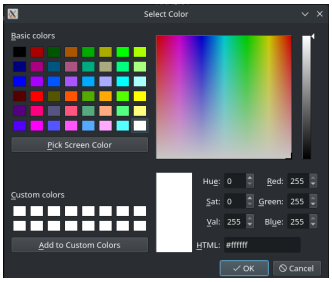


Figura 5.5: Cambiar color

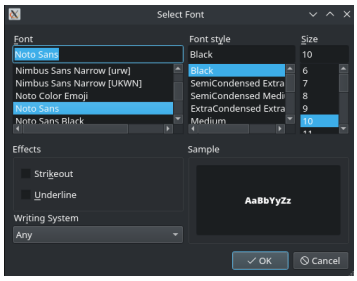


Figura 5.6: Cambiar fuente

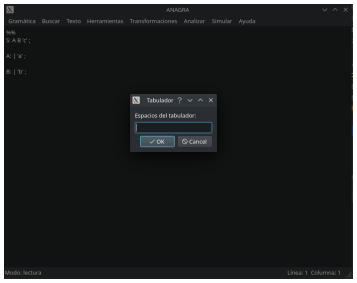


Figura 5.7: Cambiar espacios

Figura 5.8: Menús

5.6. Menú Ayuda

En el menú Ayuda ofrece dos opciones, la primera de ellas es mostrar el log de la herramienta que indica las acciones que se han ido realizando en la aplicación, mientras que la segunda muestra una ventana con información relativa sobre el TFG (autora, director, enlace al repositorio...).

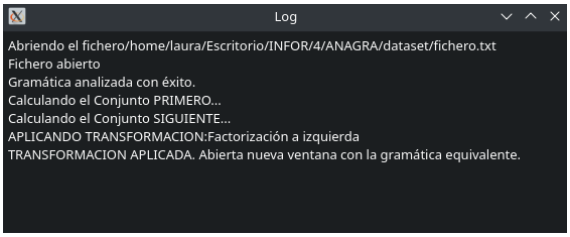


Figura 5.9: Log de la aplicación.

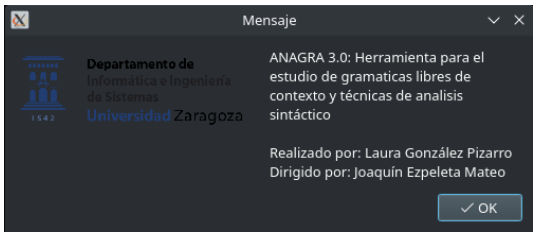


Figura 5.10: Información sobre ANAGRA.

5.7. Menú Herramientas

Dentro del menú de Herramientas se permite al usuario calcular el conjunto Primero, Siguiente o Primero de una forma de frase que introduzca el usuario, de una gramática. Cada conjunto se mostrará en una nueva ventana emergente, como indica la Figura 5.11. Para calcular el conjunto Primero de una forma de frase primero se abrirá una ventana con dos entradas de texto, en la de arriba es donde el usuario podrá introducir la forma de frase a calcular y una vez pulse el botón de aceptar se mostrará el conjunto en la de abajo. Este proceso se muestra en la Figura 5.12.

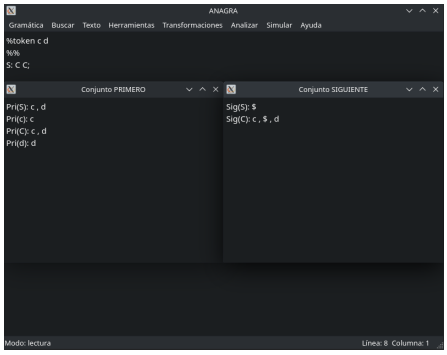


Figura 5.11: Conjunto Primero y Siguiente

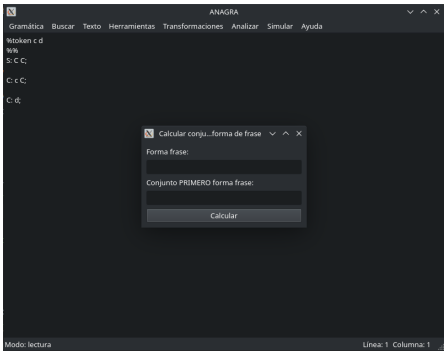


Figura 5.12: Conjunto Primero forma frase

5.8. Menú Transformaciones

Dentro del menú Transformaciones puede realizar las diferentes operaciones de transformación y formas normales sobre la gramática detalladas en la Sección 3.2. Si la gramática no cumple con las precondiciones necesarias para realizar la operación deseada se mostrará un mensaje al usuario cual son las condiciones que no cumple, sino se realizara la operación y gramática resultante se abrirá en una nueva ventana emergente. El psuedocodigo de las operaciones de transformación se puede consultar en el Anexo A.

5.9. Menú Análisis

Dentro del menú Análisis se permite al usuario realizar el análisis sintáctico LL(1), SLR, LALR o LR de la gramática. Para el análisis descendente se mostrará solo la tabla de análisis, mientras que para el análisis ascendente se mostrarán: la gramática expandida, el autómata en ventana y otra en texto plano, y las tablas Acción e Ir A. Cada componente del análisis se mostrará en una nueva ventana emergente. En el caso de que no haya conflictos en el análisis habilitará al usuario poder guardar las tablas calculadas en un fichero JSON.

La Figura 5.13 muestra las tablas del análisis (SLR en este caso) correspondientes a la misma gramática, mientras que en la Figura 5.14 se muestra el autómata asociado a la gramática. En caso de conflictos en las tablas, por no tratarse de una gramática de la clase, las celdas con conflictos se destacan en rojo.

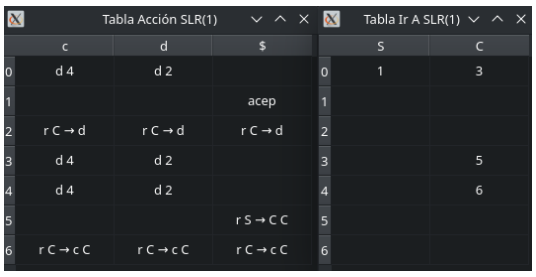


Figura 5.13: Tablas análisis SLR

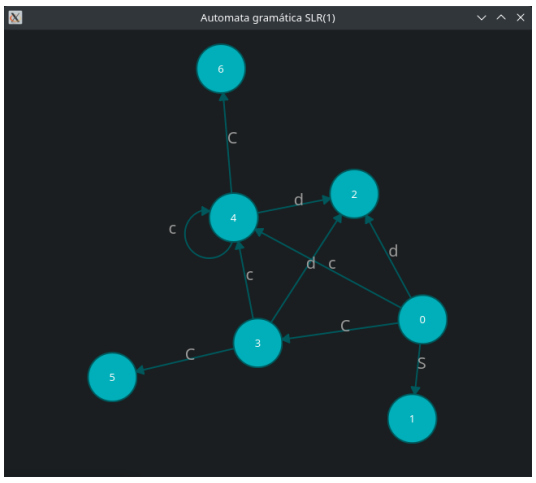


Figura 5.14: Autómata análisis SLR

En la representación gráfica, se puede mover un nodo por la pantalla simplemente haciendo clic sobre él y arrastrándolo. Mientras que con dos clic sobre un nodo, se mostraría el conjunto de configuraciones asociadas al mismo, pudiendo organizarse de la manera como en la Figura 5.15. ANAGRA lleva a cabo una distribución automática de los nodos en la ventana, de manera que se facilita la interpretación del mismo.

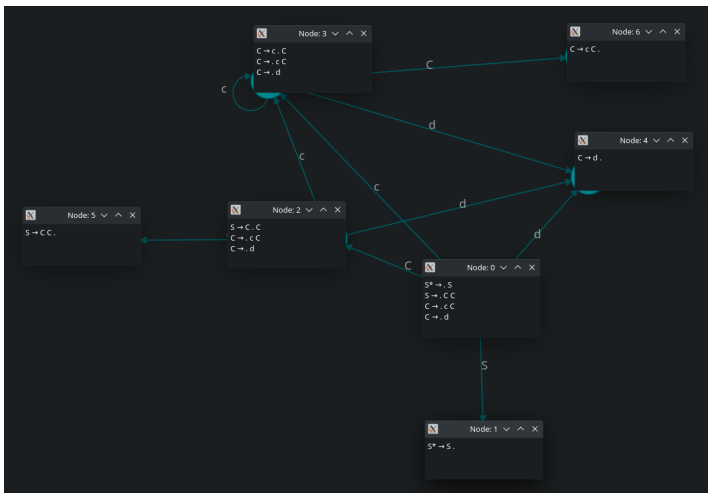


Figura 5.15: Autómata mostrando la información de cada nodo

5.10. Menú Simular

Dentro del menú Simular se permite al usuario realizar la simulación interactiva de los diferentes analizadores para una entrada de texto, siempre que se haya realizado el análisis previamente y no haya encontrado ningún conflicto. Para ello, primero se mostrará un menú donde el usuario podrá introducir una entrada de texto. Una vez el usuario pulse el botón de aceptar se cerrará la ventana anterior y se abrirán dos ventanas, la primera mostrará la simulación de la pila, la entrada, la secuencia de producciones aplicadas y los botones de avanzar y retroceder, mientras la segunda mostrará la construcción dinámica del árbol de sintaxis. Una vez acabada la simulación se le indicará al usuario si la frase es reconocida por la gramática o no. Este proceso se muestra en las Figuras 5.16, mientras que la Figura 5.17 muestra paso a paso la construcción del árbol de sintaxis.

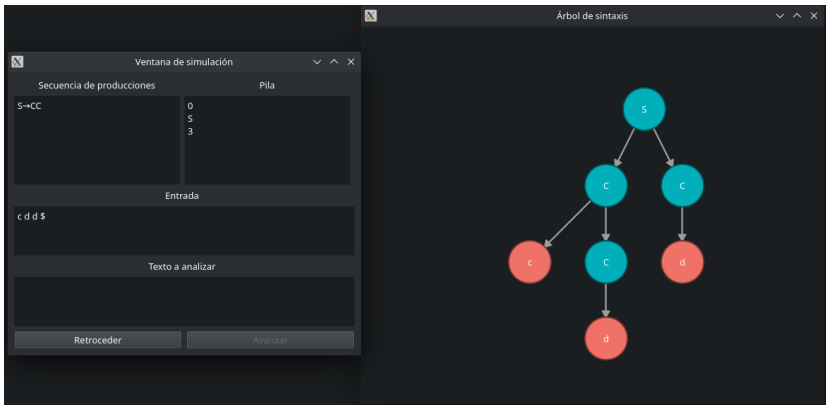


Figura 5.16: Estado del analizador durante el proceso de análisis

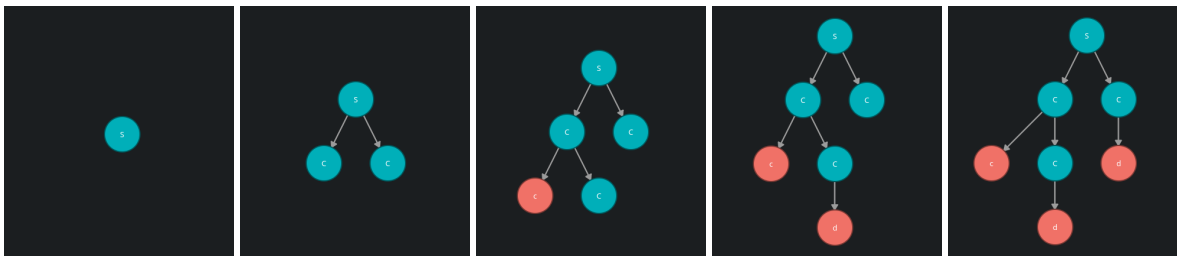


Figura 5.17: Construcción del árbol de sintaxis paso a paso

En la representación gráfica del árbol de sintaxis, los nodos azules se utilizan para denotar símbolos no terminales, mientras que los nodos rojos se reservan para los símbolos terminales. Además, los colores y estilos de línea han sido cuidadosamente seleccionados para garantizar una legibilidad óptima en ambos modos de visualización, ya sea en entornos de fondo claro u oscuro.

Capítulo 6

Conclusiones

Este trabajo tenía como objetivo principal implementar una herramienta para el estudio de gramáticas libres de contexto y análisis sintáctico. A modo de conclusión se destacan los principales hitos logrados. En primer lugar, con la herramienta ANAGRA se permite el estudio de gramáticas libres de contexto ya que se pueden reconocer gramáticas a partir de una especificada en Bison. También, se puede obtener información relativa a gramáticas como el conjunto Primero, el conjunto Siguiente y el conjunto Primero de una forma de frase. Además de realizar operaciones de transformación sobre gramáticas, estas son: factorización a izquierda, eliminación de no terminales no derivables, eliminación de recursividad a izquierda, eliminación de símbolos no alcanzables, eliminación de producciones ϵ, eliminación de ciclos y transformación a forma normal de Chomsky y de Greibach.

Del mismo modo, ANAGRA también sirve para el estudio del análisis sintáctico ya que permite el análisis ascendente LL(1) donde se calcula la tabla para el análisis, mientras que para análisis descendente de tipo SLR, LR canónico y LALR se puede calcular la gramática ampliada, la tabla Acción e Ir A y el autómata correspondiente al análisis tanto en formato gráfico como en texto. También, se puede simular interactivamente una entrada de texto mostrando la evolución de la entrada, la pila, la secuencia de producciones y el árbol de sintaxis, y determinar si la entrada pertenecen o no al lenguaje.

Asimismo, se ha ampliado la funcionalidad de la herramienta respecto a la versión anterior, incorporando características adicionales que enriquecen su utilidad. Las mejoras incluyen la implementación de operaciones de transformación de gramáticas a las formas normales de Chomsky y Greibach, facilitando su análisis. También se ha habilitado la capacidad de guardar tablas de análisis sintáctico, proporcionando utilidad para la generación de analizadores por tabla. Una tercera funcionalidad permite la simulación interactiva del análisis sintáctico, con la posibilidad de avanzar o retroceder en el proceso.

Para mejorar la experiencia del usuario, se modernizó la interfaz y se resolvieron problemas de la versión anterior, como la representación de autómatas correspondientes al análisis ascendente donde ahora se pueden visualizar los arcos de un nodo a sí mismo y las etiquetas de dos nodos que se apuntan entre sí. Además, se implementaron ciertas operaciones concurrentes para evitar que la interfaz se quede congelada durante el análisis sintáctico en casos de gramáticas complejas. Además, se propusieron cambios como la reubicación del log de la herramienta y reestructuración del formato de presentación del autómata del análisis ascendente en texto siguiendo un estilo similar al utilizado por Bison.

Estas mejoras, junto con la ampliación de funcionalidades, consolidan ANAGRA 3.0 como una herramienta robusta y versátil. De este modo, se cumplieron todos los objetivos propues-

tos, posicionando a ANAGRA como una valiosa herramienta de apoyo para el estudio de gramáticas libres de contexto y técnicas de análisis sintáctico.

6.1. Cronograma

Durante los 10 meses aproximados de duración de este Trabajo Fin de Grado, el proyecto se ha divido en las tareas mostradas en la la Cuadro 6.1, la cual expresa la evolución temporal de este proyecto desde que se inicio en febrero hasta su finalización en enero. El Cuadro 6.2 expone la dedicación en horas a cada uno de las tareas de este proyecto.

Tarea	Feb	Mar	Abr	May	Ago	Sep	Dic	Ene
Revisar documentación de librerías								
Implementar las gramáticas								
Implementación de la interfaz de usuario								
Implementar el analizador LL(1)								
Implementar el analizador SLR								
Implementar el analizador LR								
Implementar el analizador LALR								
Redactar la memoria								

Cuadro 6.1: Diagrama de Gantt.

Tarea	**Tiempo (horas)**
Revisión del estado del arte	10
Implementación de la funcionalidad	150
Implementación de la interfaz	125
Reuniones	20
Redacción de la memoria	75
Redacción manual de usuario	10
Total	390

Cuadro 6.2: Horas dedicadas a cada tarea del proyecto.

Finalmente, el código del trabajo realizado es accesible públicamente desde el repositorio mostrado a continuación bajo la licencia GNU GPL-3.0:

`https://github.com/llauragonzalezz/ANAGRA`

6.2. Trabajo futuro

Finalizado el trabajo, se plantea su incorporación como material de apoyo para la asignatura de Procesadores de Lenguajes de la Universidad de Zaragoza. Como ayuda, se ha redactado un manual de usuario de la herramienta, que se puede consultar en el Apéndice B.

Para futuras investigaciones y mejoras en este proyecto, se pueden considerar diversas líneas de trabajo con el objetivo de ampliar y perfeccionar las capacidades del sistema. En primer lugar, se podría explorar la implementación de analizadores sintácticos alternativos, incluyendo aquellos con un *lookahead* diferente de 1 o analizadores recursivos como el GLR (Generalized Left-to-right Rightmost derivation)[23]. Esto permitiría evaluar y comparar el

rendimiento de distintos enfoques en la fase de análisis sintáctico, contribuyendo así a la robustez y versatilidad del sistema.

Adicionalmente, también se podría considerar la implementación de ciertas funcionalidades de la herramienta para ser concurrente o distribuida. Esto agilizaría el cálculo de operaciones costosas, como el cálculo de configuraciones canónicas en gramáticas muy grandes.

Por ultimo, otra línea de trabajo prometedora sería la implementación de herramientas adicionales para el análisis de lenguajes. Esto podría abarcar desde el análisis léxico hasta la el análisis semántico. La expansión de estas capacidades proporcionaría una mayor profundidad y amplitud en la comprensión del código fuente, permitiendo abordar aspectos de los programas analizados.

Bibliografía

[1] Alfred V Hoe, Ravi Sethi y Jeffrey D Ullman. "Compilers—principles, techniques, and tools". En: (1986).

[2] Joaquín Ezpeleta. "Material de apoyo". En: (2009). URL: https://webdiis.unizar.es/~ezpeleta/doku.php?id=material_de_apoyo.

[3] Terence Parr y Kathleen Fisher. "LL (*) the foundation of the ANTLR parser generator". En: *ACM Sigplan Notices* 46.6 (2011), págs. 425-436.

[4] Alfred V. Aho y Stephen C. Johnson. "LR parsing". En: *ACM Computing Surveys (CSUR)* 6.2 (1974), págs. 99-124.

[5] Frank DeRemer y Thomas Pennello. "Efficient computation of LALR (1) look-ahead sets". En: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.4 (1982), págs. 615-649.

[6] Thomas Finley Susan Rodger. *FLAP - An Interactive Formal Languages and Automata Package*. .ºReilly Media, Inc.", 2006.

[7] Jean Bovet y Terence Parr. "ANTLRWorks: an ANTLR grammar development environment". En: *Software: Practice and Experience* 38.12 (2008), págs. 1305-1332.

[8] Peter winter. "Proyecto ctpg". En: (2021). URL: https://github.com/peter-winter/ctpg.

[9] schnorr. "Proyecto Ellerre". En: (2021). URL: https://github.com/schnorr/ellerre.

[10] Igor Dejanović. "Parglare: A LR/GLR parser for Python". En: *Science of Computer Programming* (2021), pág. 102734. ISSN: 0167-6423. DOI: 10.1016/j.scico.2021.102734. URL: https://www.sciencedirect.com/science/article/pii/S0167642321001271.

[11] Stephen C Johnson et al. *Yacc: Yet another compiler-compiler*. Vol. 32. Bell Laboratories Murray Hill, NJ, 1975.

[12] Fredrik Lundh. "An introduction to tkinter". En: *URL: www. pythonware. com/library-/tkinter/introduction/index. htm* (1999).

[13] Noel Rappin. *wxPython in Action*. 2006.

[14] Mark Summerfield. *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming (paperback)*. Pearson Education, 2007.

[15] Gopinath Jaganmohan y Venkateshwaran Loganathan. *PySide GUI Application Development*. Packt Publishing Ltd, 2016.

[16] Shannon Behrens. "Prototyping Interpreters using Python Lex-Yacc". En: *Dr. Dobb's Journal*. (2004).

[17] Armin Cremers y Seymour Ginsburg. "Context-free grammar forms". En: *Journal of Computer and System Sciences* 11.1 (1975), págs. 86-117.

[18] Noam Chomsky. "On the representation of form and function". En: (1981).

[19] Sheila A Greibach. “A new normal-form theorem for context-free phrase structure grammars”. En: *Journal of the ACM (JACM)* 12.1 (1965), págs. 42-52.

[20] Leodanis Pozo Ramos. “Use PyQt’s QThread to Prevent Freezing GUIs”. En: (2019). URL: `https://realpython.com/python-pyqt-qthread/`.

[21] Glenn E Krasner, Stephen T Pope et al. “A description of the model-view-controller user interface paradigm in the smalltalk-80 system”. En: *Journal of object oriented programming* 1.3 (1988), págs. 26-49.

[22] Dave Crocker y Paul Overell. *Augmented BNF for syntax specifications: ABNF*. Inf. téc. 2008.

[23] Scott McPeak y George C Necula. “Elkhound: A fast, practical GLR parser generator”. En: *International Conference on Compiler Construction*. Springer. 2004, págs. 73-88.

Apéndice A

Métodos de transformación de una gramática

A.1. Cálculo del conjunto Primero

Algorithm 1 Calcular Conjunto Primero

Require: $X \in N \cup T$
Ensure: calcula $PRI(X)$
  repeat
    if $X \in T$ **then**
      añadir X a $PRI(X)$
    end if
    if $X \to \varepsilon$ es una producción **then**
      añadir ε a $PRI(X)$
    end if
    if $X \to Y_1 \dots Y_k$ es una producción **then**
      for $j \leftarrow 1$ to k **do**
        if $a \in PRI(Y_j) \wedge \varepsilon \in PRI(Y_1) \cap \dots \cap PRI(Y_{j-1})$ **then**
          añadir a a $PRI(X)$
        end if
      end for
      if $\varepsilon \in PRI(Y_1) \cap \dots \cap PRI(Y_k)$ **then**
        añadir ε a $PRI(X)$
      end if
    end if
  until no se añada nada a ningún PRI

A.2. Cálculo del conjunto Siguiente

Algorithm 2 Calcular Conjunto Siguiente

Require: $True$
Ensure: calcula $SIG(A)$ para todo $A \in N$
  añadir \$ a $SIG(S)$
  repeat
    for cada producción $A \to \alpha B \beta$ **do**
      añadir $PRI(\beta) \backslash \{\varepsilon\}$ a $SIG(B)$
    end for
    for cada producción $A \to \alpha B$ o $A \to \alpha B \beta$ con $\varepsilon \in PRI(\beta)$ **do**
      añadir $SIG(A)$ a $SIG(B)$
    end for
  until no se añada nada a ningún SIG

A.3. Eliminación símbolos no terminables

Algorithm 3 EliminaNoTerminables

Require: $G = (N, T, P, S)$ t.q. $L(G) \neq \emptyset$
Ensure: $G' = (N', T, P', S) \wedge G' \approx G \wedge \forall X' \in N'. \exists w \in T*. X' \Rightarrow^* w$
  $viejo := \{\}$
  $nuevo := \{A \in N \mid A \to w \in P, w \in T^*\}$
  while $viejo \neq nuevo$ **do**
    $viejo := nuevo$
    $nuevo := viejo \cup \{B \in N \mid B \to \alpha \in P, \alpha \in (T \cup viejo)^*\}$
  end while
  $N' := nuevo$
  $P' := \{A \to w \in P \mid A \in N', w \in (N' \cup T)^*\}$

A.4. Eliminación de recursividad a izquierda

Algorithm 4 Algoritmo eliminaRecIzda

Require: $G = (N, T, P, S)$ sin "ciclos" $(A \Rightarrow^+ A)$ ni producciones ε $(A \to \varepsilon)$ y $N = \{A_1, \ldots, A_n\}$
Ensure: $G' \approx G$, sin rec. a izda.
  $G' \leftarrow G$
  for $i \leftarrow 1$ to n **do**
    for $j \leftarrow 1$ to $i - 1$ **do**
      $A_j \to \delta_1 | \ldots | \delta_k$ son las producciones actuales de A_j sustituir en P'
      $A_i \to A_j \alpha$ por $A_i \to \delta_1 \alpha | \ldots | \delta_k \alpha$
    end for
    eliminar rec. inmediata de A_i
  end for

A.5. Factorización a izquierda

Algorithm 5 Factorizar Gramática a la Izquierda

Require: $G = (N, T, P, S)$
Ensure: Gramática G factorizada a la izquierda
  $P' \leftarrow P$
  for cada no terminal $A \in N$ **do**
    for cada par de producciones $A \to \alpha\beta$ y $A \to \alpha\gamma$ en P' **do**
      if α es un prefijo común de β y γ **then**
        Crear un nuevo no terminal A' y actualizar N, P':
          $N \leftarrow N \cup \{A'\}$
          Reemplazar $A \to \alpha\beta$ y $A \to \alpha\gamma$ con:
            $A \to \alpha A'$
            $A' \to \beta \mid \gamma$
      end if
    end for
  end for

A.6. Eliminación de ciclos

Algorithm 6 Eliminación de ciclos

Require: $G = (N, T, P, S)$
Ensure: Elimina los ciclos de una gramática
  Eliminar las producciones ϵ
  for cada regla unitaria $A \to B$ **do**
    for cada regla $B \to X_1 \dots X_n$ **do**
      Añadir la regla $A \to X_1 \dots X_n$, a menos que sea una regla unitaria ya eliminada
    end for
  end for

A.7. Eliminación de producciones épsilon

Algorithm 7 Eliminación de producciones épsilon en una Gramática

Require: $G = (N, T, P, S)$
Ensure:
  for cada regla $A \to X_1 \dots X_n$ **do**
    Si todos los X_i son nulos, marcar A como nulo
  end for
  for cada regla $A \to X_1 \dots X_n$ **do**
    Crear todas las combinaciones omitiendo los X_i nulos
    Eliminar todas las reglas ϵ, excepto si la regla es $S_0 \to \varepsilon$
  end for

A.8. Eliminación de no terminales no derivables

Algorithm 8 EliminaNoDerivables

Require: $G = (N, T, P, S)$ t.q. $L(G) \neq \emptyset$
Ensure: $G' = (N', T, P', S) \wedge G' \approx G \wedge \forall X' \in N'.\exists w \in T*.X' \Rightarrow^* w$
  $viejo := \{\}$
  $nuevo := \{A \in N \mid S \rightarrow w \in P, w \in N\}$
  while $viejo \neq nuevo$ **do**
    $viejo := nuevo$
    $nuevo := viejo \cup \{B \in N \mid A \rightarrow B \in P, A \in viejo\}$
  end while
  $N' := nuevo$
  $P' := \{A \rightarrow w \in P \mid A \in N', w \in (N' \cup T)^*\}$

A.9. Eliminación de no accesibles

Algorithm 9 EliminaNoAccesibles

Require: $G = (N, T, P, S)$ t.q. $L(G) \neq \emptyset \wedge \forall X \in N.X$ es terminable
Ensure: $G' = (N', T', P', S) \wedge G' \approx G \wedge \forall X \in (N' \cup T').\exists \alpha, \beta \in (N' \cup T').S \Rightarrow^* \alpha X \beta$
  $viejo := \{S\}$
  $nuevo := \{X \in (N \cup T) \mid S \rightarrow \alpha X \beta \in P\} \cup \{S\}$
  while $viejo \neq nuevo$ **do**
    $viejo := nuevo$
    $nuevo := viejo \cup \{Y \in (N \cup T) \mid A \rightarrow \alpha Y \beta \in P, A \in viejo\}$
  end while
  $N' := nuevo \cap N$
  $T' := nuevo \cap T$
  $P' := \{A \rightarrow w \in P \mid A \in N', w \in (N' \cup T')^*\}$

Apéndice B

Manual de usuario

B.1. Instalación de la herramienta

Una vez clonado el repositorio de Github es necesario la instalacion de los paquetes PyQt5, networkx y pygraphviz.

B.1.1. Instalación de la herramienta en Linux

Para instalar los paquetes necesarios para la ejecución de ANAGRA en Linux es necesario realizar los siguientes pasos:

```
pip install PyQt5
pip install networkx
pip install pygraphviz
```

B.1.2. Instalación de la herramienta en Windows

Para instalar los paquetes necesarios para la ejecución de ANAGRA en Windows es necesario realizar los siguientes pasos:

```
pip install PyQt5
pip install networkx
```

Para instalar pygraphviz es necesario seguir el tutorial en su web [1].

B.2. Tutorial del sistema

B.2.1. Menú Gramática

Para trabajar con ANAGRA lo primero de todo es disponer de una gramática sobre la que poder operar. Para ello existen dos posibilidades, se puede obtener de un fichero o la pode escribir el usuario en el editor.

Nueva (CTRL+N)

Esta opción crea una ventana nueva ventana independiente de la actual donde se puede trabajar con otra gramática distinta.

[1] Tutorial instalacion pygraphviz.

Abrir (CTRL+O)

Esta opción permite abrir el fichero la gramática con la que se va trabajar. Para ello, el usuario debe seleccionar del un cuadro de diálogo mostrado en la Figura B.1 el fichero. El formato del fichero aparece en el anexo B.2.11. Una vez seleccionada la gramática y esta no contenga errores, se mostrará en el editor.

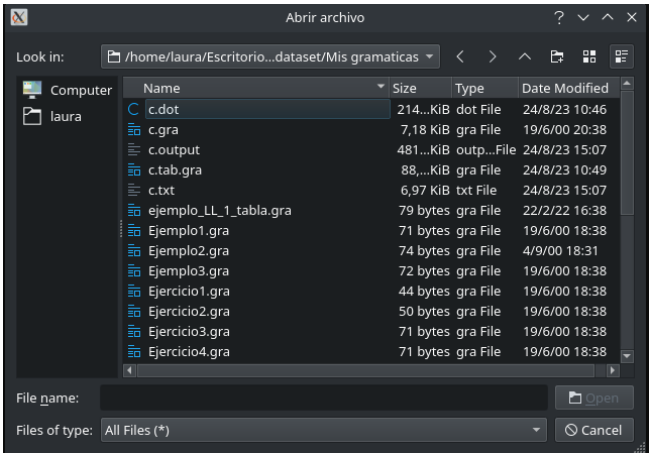


Figura B.1: Menú abrir fichero.

Editar

Esta opción se utiliza para modificar una gramática abierta anteriormente o editada anteriormente y que ya haya sido comprobadas. Esto hace que el editor vuelva a modo edición y el usuario pueda realizar los cambios deseados en la gramática.

Guardar (CTRL+G)

Con esta opción se guarda la gramática actual de trabajo en fichero. Si la gramática ya había sido leída de un fichero, se guardará automáticamente. En caso contrario de comportará como la opción Guardar como explicada en el apartado siguiente.

Guardar como

Esta opción permite lo mismo que la anterior pero dando la opción de que el fichero donde se guarde la gramática sea introducido. El dialogo es el mismo que el de la Figura B.1

Salir

Provoca el cierre de la aplicación y la pérdida de toda aquella información que no haya sido guardada. Aparece un diálogo como el de la Figura B.2 recordando al usuario guardar las preferencias y dándole la posibilidad de continuar la operación o cancelarla.

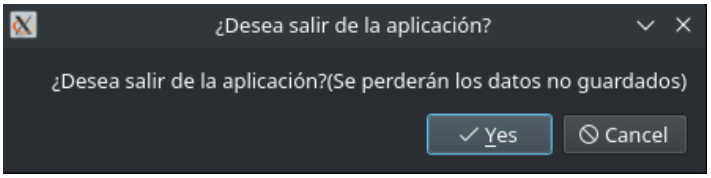


Figura B.2: Opciones especificas de la aplicación para gramáticas.

B.2.2. Menú Edición

Cortar (CTRL+X)

Borra el texto seleccionado del área de edición de la gramática y lo copia en el portapapeles del sistema.

Copiar (CTRL+C)

Copia el texto seleccionado del área de edición de la gramática al portapapeles del sistema.

Pegar (CTRL+V)

Pega el contenido del portapapeles en el área de edición de la gramática.

Borrar

Borra el de texto seleccionado por el cursor.

Seleccionar todo (CTRL+A)

Selecciona todo el texto introducido en el área de edición de gramáticas.

Aceptar gramática

Se ha de presionar esta opción cuando se haya terminado de editar la gramática y se quiera poder utilizar las opciones de análisis y transformación sobre ella. Una vez presionado se verificará la corrección de la gramática editada. Si es correcta finalizará el modo edición. En caso contrario se mostrará un mensaje como el de la figura B.3 donde se muestra el carácter, la linea y la columna donde esta el error, y se permanecerá en el modo edición.

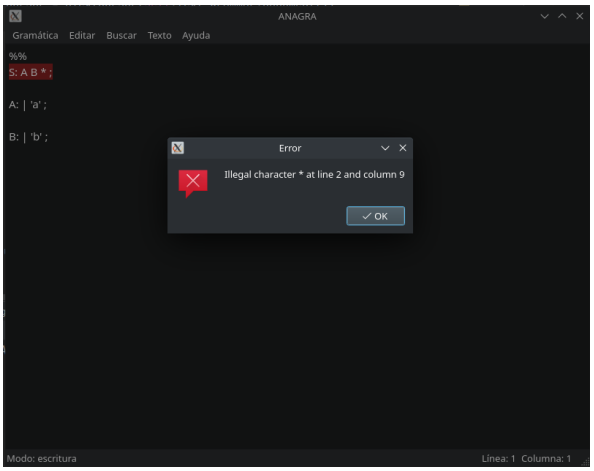


Figura B.3: Mensaje de error

B.2.3. Menú Buscar

Buscar (CTRL+F)

Para buscar una palabra en la gramática, primero, se abrirá una ventana donde el usuario podrá introducir la palabra a buscar, cuando pulse el botón de aceptar se subrayarán en verde las coincidencias, mientras, que en caso de no haber ninguna coincidencia, se mostrará un mensaje indicándolo. La Figura 5.4 muestra el proceso de buscar una palabra en el texto.

Buscar (CTRL+B)

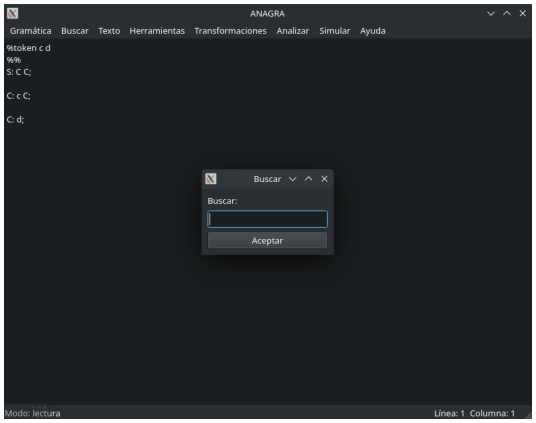


Figura B.4: Tablas análisis SLR

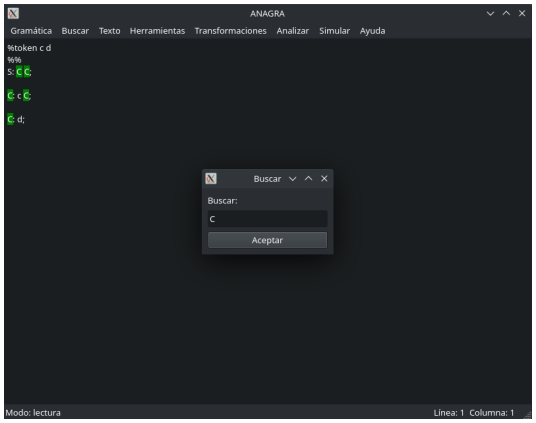


Figura B.5: Autómata análisis SLR

Reemplazar (CTRL+R)

Para remplazar una palabra por otra en la gramática, primero, se abrirá una ventana donde el usuario podrá introducir la palabra a buscar y aquella por la que la quiera cambiar, cuando pulse el botón de aceptar se remplazarán todas las coincidencias encontradas, mientras, en caso de no haber ninguna coincidencia se mostrará un mensaje indicándolo. La Figura 5.4 muestra el proceso de remplazar una palabra por otra en el texto.

B.2.4. Menú Buscar

Buscar (CTRL+B)

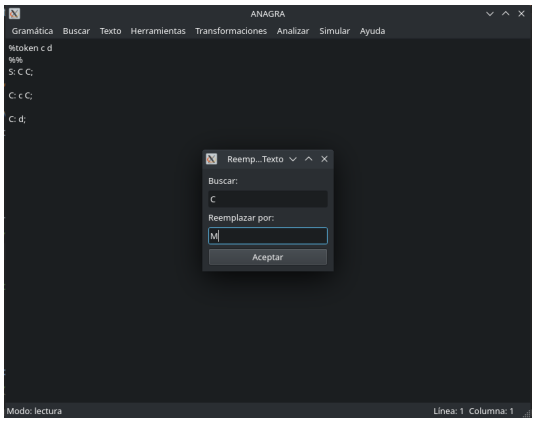


Figura B.6: Tablas análisis SLR

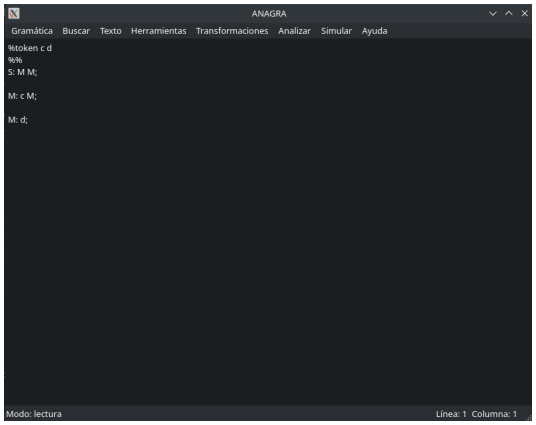


Figura B.7: Autómata análisis SLR

B.2.5. Menú Texto

Fuente

Permite cambiar el tipo de letra con el que estamos editando o mostrando la gramática. En la Figura B.9 aparece el diálogo.

Color

Permite cambiar el color de letra con el que estamos editando o mostrando la gramática. En la Figura B.8 aparece el diálogo.

Tabulador

Permite cambiar el número de espacios que formarán un tabulador por defecto. En la Figura B.10 aparece el diálogo.

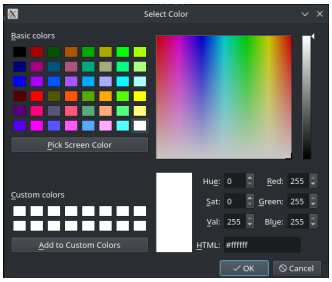


Figura B.8: Cambiar color

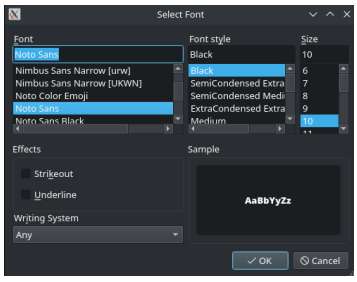


Figura B.9: Cambiar fuente

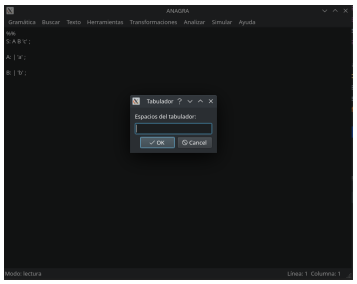


Figura B.10: Cambiar espacios

Figura B.11: Menús

Extendido

Permite cambiar el formato en el que aparece la gramática al formato compacto. En este formato las producciones aparecen de la siguiente manera.

```
Parte Izquierda :
                Parte Derecha1
                | Parte Derecha2
                | ..................
                | Parte Derecha n
;
```

Compacto

Permite cambiar el formato en el que aparece la gramática al formato compacto. En este formato las producciones aparecen de la siguiente manera.

```
Parte Izquierda : Parte Derecha1 | Parte Derecha2 | .... | Parte Derechan;
```

Guardar preferencias

Permite guardar las opciones de tipo de letra, color de letra y espacios de tabulador elegidos, para que la próxima vez que se abrá la aplicación sean tomados como valores por defecto.

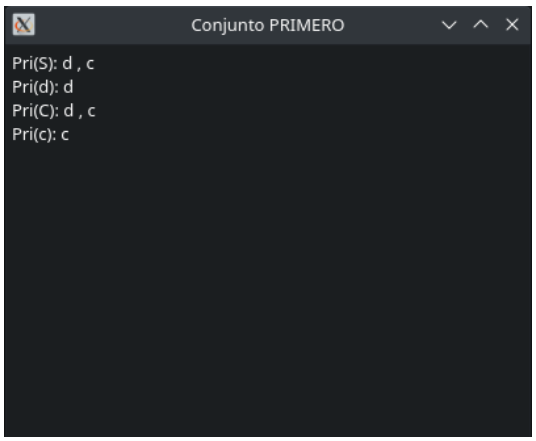
B.2.6. Menú Herramientas

Calcular conjunto Primero

Muestra una nueva ventana como la Figura B.12a donde se muestra el conjunto de Primero de los símbolos de la gramática.

Calcular conjunto Siguiente

Muestra una nueva ventana como la Figura B.12a donde se muestra el conjunto de Siguiente de los símbolos de la gramática.



(a) Conjunto Primero

(b) Conjunto Siguiente

Calcular conjunto Primero de una forma frase

Esta opción permite calcular el conjunto Primero de una forma de frase. Para ello, primero se mostrará una ventana en la que el usuario podrá ingresar la forma de la frase que desea analizar en el campo superior. Luego, al pulsar el botón "Calcula", el conjunto Primero de la forma de frase se mostrará en el campo inferior de la ventana. En la Figura B.13 se muestra la ventana:

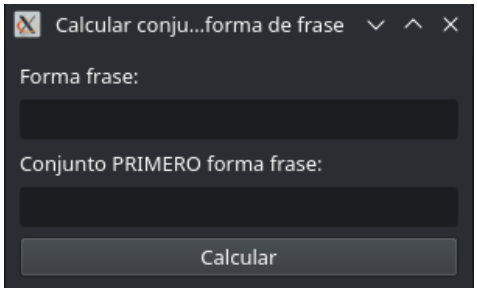


Figura B.13: Opciones especificas de la aplicación para gramáticas.

B.2.7. Menú Transformaciones

En este menú aparece la posibilidad de aplicar distintas transformaciones a la gramática actual para convertirla en otra equivalente con objeto de que se pueda adaptar a uno u otro tipo de gramática. Las posibles transformaciones se enumeran a continuación:

- **Eliminación de no derivables**: eliminar los símbolos que no son capaces de derivar ningún otro símbolo.
- **Factorización a izquierda**: factorizar a izquierda de todas aquellas producciones que pueden provocar *backtracking* en el analizador.
- **Eliminación de ciclos**: eliminar todos aquellos símbolos no terminales de la gramática que derivan en sí mismos en uno o más pasos.

- **Eliminación de no accesibles**: eliminar todos aquellos símbolos de la gramática que no pueden ser accedidos desde el símbolo inicial.
- **Eliminación de anulables**: eliminar todas aquellas producciones cuya parte derecha es lambda
- **Eliminación de la recursividad a izquierda**: eliminar la recursividad a izquierda de las producciones.
- **Forma normal de Greibach**: transforma la gramática en la forma normal de Greibach.
- **Forma normal de Chomsky**: transforma la gramática en la forma normal de Chomsky.

Antes de realizar una transformación se comprueba las precondiciones de la operación, si la gramática no cumple con las condiciones se mostrará un mensaje como el de la Figura B.14 indicando cuales y no se realizará la operación. Si la gramática cumple la precondición se aplicará la transformación y la gramática resultante aparecerá en una ventana nueva para que puedan compararse las diferencias entre la gramática original y la transformada.

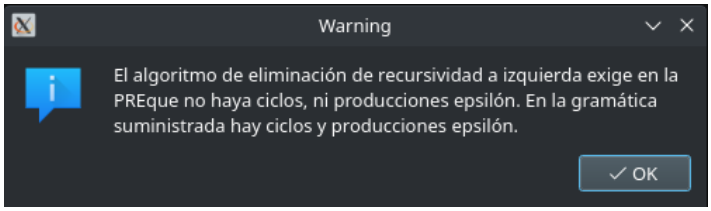


Figura B.14: Opciones especificas de la aplicación para gramáticas.

B.2.8. Menú Analizar

En el menú Analizar se puede realizar el análisis sintáctico de la gramática, mostrando las diferentes partes del análisis e indicando si corresponde o no al tipo específico de análisis llevado a cabo

Analizar gramática LL(1)

Esta opción ejecuta el análisis LL(1) de la gramática, mostrando, en ventanas independientes, la tabla para análisis LL(1). Las casillas de la tabla que aparecen con el fondo rojo son aquellas en las que existe un conflicto. En la Figura B.15 aparece una de estas tablas generadas a partir del análisis LL(1) de la gramática. El análisis solo se realiza si es la primera vez que se pulsa en esta opción, las siguientes veces solo se muestran las ventanas.

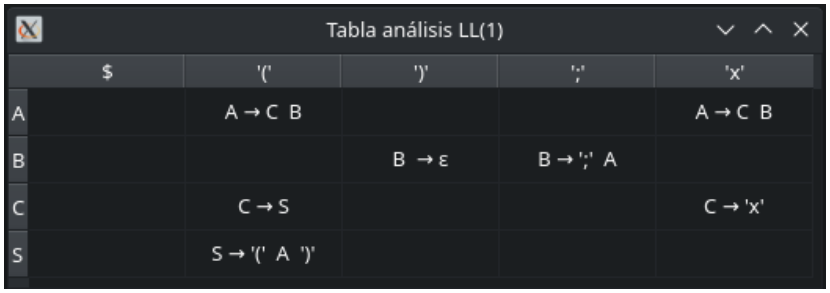


Figura B.15: Opciones especificas de la aplicación para gramáticas.

Guardar tabla análisis LL(1)

Esta opción está solamente activa si se ha ejecutado el análisis LL(1) anteriormente y la gramática no tiene conflictos, y muestra para seleccionar el fichero donde guardar las tablas para el análisis.

Analizar gramática SLR

Esta opción ejecuta el análisis SLR de la gramática, mostrando, en ventanas independientes, las tablas para análisis SLR, el autómata de conjuntos de configuraciones LR(0), la gramática ampliada y una versión del autómata en modo texto. Las casillas de la tabla que aparecen con el fondo rojo son aquellas en las que existe un conflicto. En la Figura B.16 aparece una de estas tablas generadas a partir del análisis SLR de la gramática, en la Figura B.18 se muestra el autómata en ventana y en texto, y en la Figura se muestra la gramática expandida. El análisis solo se realiza si es la primera vez que se pulsa en esta opción, las siguientes veces solo se muestran las ventanas.

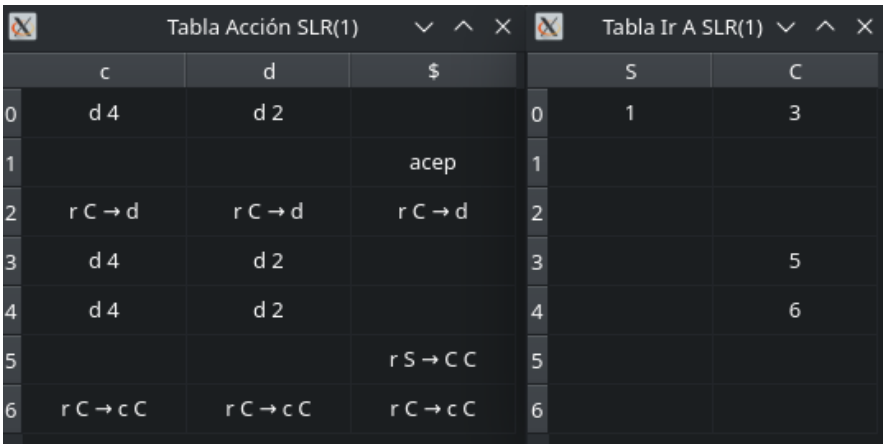


Figura B.16: Tablas análisis SLR.

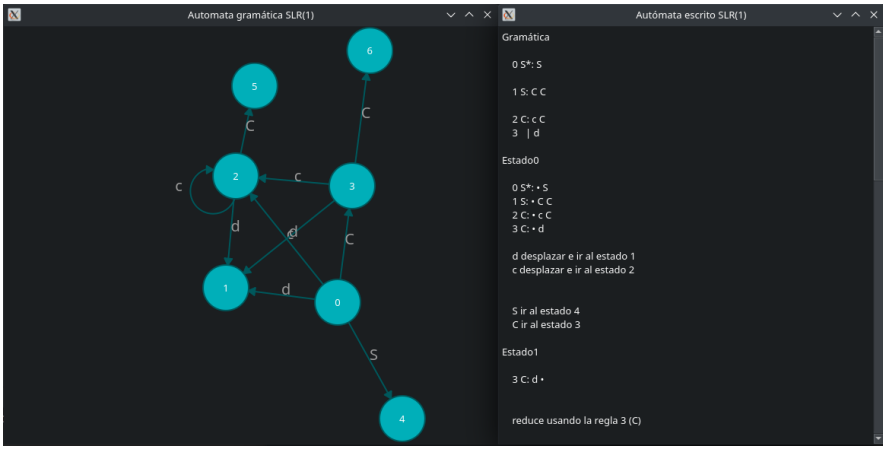


Figura B.17: Autómatas análisis SLR.

Guardar tablas SLR

Esta opción está solamente activa si se ha ejecutado el análisis SLR anteriormente y la gramática no tiene conflictos, y muestra para seleccionar el fichero donde guardar las tablas para el análisis.

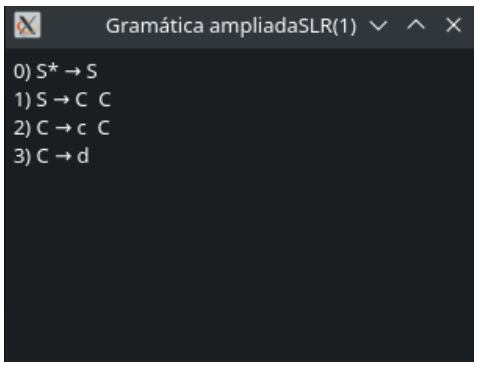


Figura B.18: Gramática ampliada del análisis SLR.

Analizar gramática LALR

El proceso para esta opción es idéntico al que se explica en la Sección B.2.8. La única diferencia es que en este caso cambia el algoritmo de simulación.

Guardar tabla tablas LALR

El proceso para esta opción es idéntico al que se explica en la Sección B.2.8. La única diferencia es que en este caso cambia el algoritmo de simulación.

Analizar gramática LR

El proceso para esta opción es idéntico al que se explica en la Sección B.2.8. La única diferencia es que en este caso cambia el algoritmo de simulación.

Guardar tabla tablas LR

El proceso para esta opción es idéntico al que se explica en la Sección B.2.8. La única diferencia es que en este caso cambia el algoritmo de simulación.

B.2.9. Menú Simular

Simular entrada LL(1)

Esta opción permite, una vez que se ha comprobado que la gramática actual es del tipo LL(1), simular el analizador sintáctico tipo LL(1) asociado a la gramática. Para ello, el programa muestra una ventana como la de la Figura B.19 donde el usuario podrá introducir la entrada a simular.

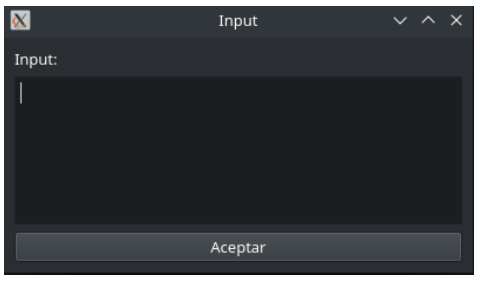


Figura B.19: Opciones especificas de la aplicación para gramáticas.

Una vez el usuario pulse el botón Aceptar, aparecerán dos ventanas, la de la derecha ventana de simulación donde se recogen las estructuras internas del simulador (pila de símbolos, producciones emitidas, entrada que falta por analizar y entrada original del simulador). Esta ventana se muestra en la Figura B.20. En esta ventana, cada vez que se pulse el botón Avanzar, se ejecutará avanzará un paso del proceso de simulación y se actualizarán las estructuras del analizador en pantalla. Mientras que si se pulsa el botón Retroceder, se retorcerá un paso en el análisis.

La ventana de la derecha contiene el árbol de sintaxis que corresponde a la entrada que se está simulando. El árbol ira avanzando o retrocediendo en su construcción a medida que el usuario avance o retroceda en el análisis. Dicha ventana aparece en la B.21.

Figura B.20: Ventana con la entrada, pila y producción disparada.

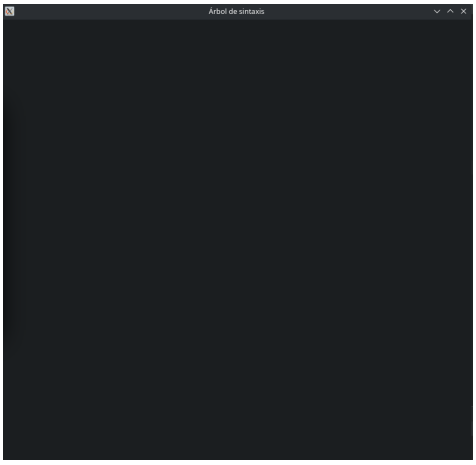
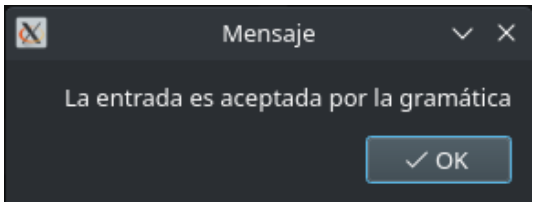
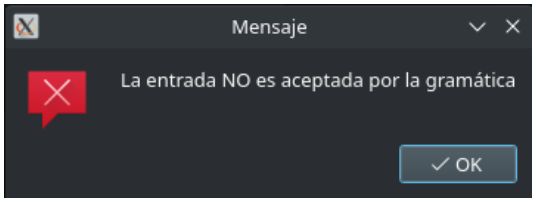


Figura B.21: Árbol de sintaxis.

Al final del análisis, se indicará como en la Figura B.22 si la entrada es aceptada por la gramática es aceptada según el analizador LL(1) o no.



(a) Entrada aceptada.



(b) Entrada No aceptada.

Figura B.22: Mensaje si el analizador acepta la entrada o no.

Simular entrada SLR

El proceso para esta opción es idéntico al que se explica en la sección B.2.9. La única diferencia es que en este caso cambia el algoritmo de simulación.

Simular entrada LALR

El proceso para esta opción es idéntico al que se explica en la sección B.2.9. La única diferencia es que en este caso cambia el algoritmo de simulación.

Simular entrada LR

El proceso para esta opción es idéntico al que se explica en la sección B.2.9. La única diferencia es que en este caso cambia el algoritmo de simulación.

B.2.10. Menú Ayuda

Log

En la ventana de log se muestran la lista de acciones que el usuario ha realizado en la aplica. Esta ventana se muestra en la Figura B.23.

Acerca de

Contiene información acerca del proyecto y las personas que han intervenido en él. Esta ventana se muestra en la Figura B.24.

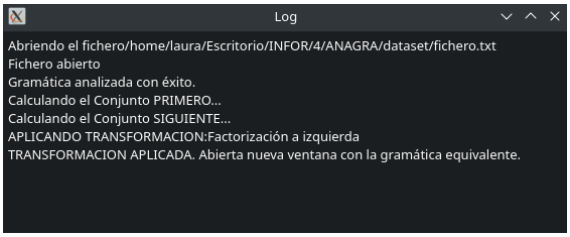


Figura B.23: Log de la aplicación.

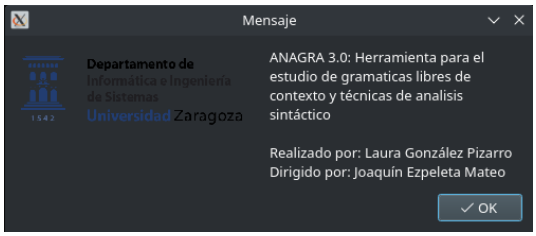


Figura B.24: Información sobre ANAGRA.

B.2.11. Formatos de entrada

Formato de los ficheros de gramáticas

El formato de los ficheros para introducir las gramáticas es el formato utilizado por YACC. Es decir, que podemos pasarle un fichero YACC y el programa lo reconocerá y obtendrá una gramática de él. Hay que advertir que existen varias restricciones a lo indicado en el párrafo anterior.

1. Los tokens terminales de la gramática podrán ser definidos como identificador de texto al inicio del fichero con la declaración %token..., o como carácter entre comillas simples en las producciones de la gramática.

Cuando leamos una gramática desde un fichero, en la pantalla principal aparecerá el contenido íntegro del fichero. Cuando editemos la gramática o apliquemos alguna transformación sobre ella, en la pantalla principal solo aparecerá información acerca de las producciones y definición de tokens. Esta información será la que se almacene en un fichero de disco en caso de que seleccionemos dicha opción del menú.

Formato de las formas de frase que se introducen en la aplicación

- **Forma de frase que se introduce para obtener el conjunto Primero.**: se introducirán los símbolos de la gramática separados por uno o más espacios en blanco o tabuladores. Las mismas palabras reservadas que tenemos para los ficheros donde se almacenan las gramáticas, los tenemos aquí. Dichas palabras se especifican en el punto anterior. Ante cualquier error en la formación de la forma de frase, ANAGRA mostraría un mensaje de error indicándolo.

- **Entrada que simularemos con el analizador sintáctico asociado a las gramáticas**: se introducirán los símbolos terminales de la gramática separados por uno o más espacios en blanco o tabuladores.