yet.anotherTruth.Tree.Tool: Una herramienta didáctica sobre Árboles de Refutación

Kevin Ruau, Juan Martín Tosini

Universidad Nacional del Centro de la Provincia de Buenos Aires, Facultad de Ciencias Exactas

{kevinruau.93, juanmartintosini}@gmail.com
http://www.exa.unicen.edu.ar/

Abstract. Con el objetivo de lograr una herramienta didáctica para la enseñanza de conceptos relacionados con la Lógica de Predicados de Primer Orden se desarrolló yaT3 (yet Another Truth Tree Tool). Esta herramienta presenta una interfaz gráfica sencilla de utilizar, simple e intuitiva que provee el soporte necesario para la construcción de Árboles de Refutación. La construcción de un Árbol de Refutación es un método para averiguar la satisfacibilidad de una fórmula. La herramienta yaT3 es software libre y está desarrollada en el lenguaje de programación C++. El proyecto fue desarrollado en el marco de un trabajo final de cátedra para dos materias de una carrera de Informática. Estas materias se dictan en el primer cuatrimestre del segundo año de la misma e involucran conceptos tanto de lógica como de análisis y diseño de algoritmos. Las mismas son Ciencias de la Computación II y Análisis y Diseño de Algoritmos I respectivamente.

Keywords: Lógica, tipo de dato abstracto, árbol de refutación, software didáctico.

1 Introducción

Las materias con contenido de lógica tienen una fuerte presencia en las carreras relacionadas con la Informática. Es vital que en las mismas el alumno adquiera una fuerte práctica en la utilización de formalismos mediante el empleo de una notación sintáctica particular. Aquí yace el punto de partida del desarrollo de este trabajo. La idea central es elaborar una herramienta informática que se presente como intuitiva y fácil de usar desde un comienzo. A partir de estas cualidades, la herramienta debe permitir que el aprendizaje de los contenidos de la materia (Ciencias de la Computación II) tome otro enfoque complementario y resulte más llevadero para el alumno que se enfrenta por primera vez con estos conceptos.

La temática sobre la que se desarrolla la herramienta es la de Árboles de Refutación. Este concepto es un método para determinar la satisfacibilidad de fórmulas a partir de una estrategia de refutación. Dentro del marco práctico de un

curso de lógica, es habitual que se requieran construcciones de estos árboles a partir de fórmulas lógicas para poder determinar consecuencias lógicamente válidas y validez de fórmulas. Si bien inicialmente se planificó abordar la problemática sólo desde el punto de vista de la Lógica de Predicados de Primer Orden, posteriormente la solución se amplió a la Lógica Proposicional.

De esta manera, la construcción de un árbol se desarrolla aplicando una serie de reglas (de tipo A o de tipo B) hasta determinar si una rama del mismo se puede cerrar. Una rama de un árbol se considera cerrada cuando existe en la misma una fórmula y su negación [1][7]. Este proceso de construcción es implementado en la herramienta yaT³. Abordando conceptos más propios de desarrollo, ligados a conocimientos de la materia Análisis y Diseño de Algoritmos I, la herramienta se implementa siguiendo un diseño orientado a clases. Mediante la identificación de TDAs [2], se abstraen características y comportamientos propios de los mismos que posteriormente pueden ser plasmados en una clase de C++, lenguaje utilizado para materializar la solución. La implementación mantuvo a su vez una fuerte presencia de aspectos relacionados con lograr la mayor eficiencia posible a la hora de procesar información. El diseño de un algoritmo que permita construir un Árbol de Refutación respetando las bases teóricas fue también uno de los puntos fuertes del proyecto.

Por último, el desarrollo de una interfaz gráfica intuitiva y sencilla de utilizar fue posible mediante la utilización de las librerías Qt [3], que brindan un entorno completo y documentado de implementación de interfaces gráficas. Este desarrollo involucro el aprendizaje desde cero de la implementación de interfaces gráficas, lo cual resultó tan desafiante como productivo.

La duración del trabajo fue de aproximadamente seis meses, comenzando a partir de haber finalizado las cursadas de las materias involucradas, pertenecientes al primer cuatrimestre del segundo año de la carrera.

2 Elementos del trabajo y metodología

La metodología de trabajo para el desarrollo del proyecto consistió en cuatro fases claramente identificables. En un primer término, se buscó identificar aquellos tipos de datos que permitieran una solución orientada a clases, aplicando nociones del paradigma orientado a objetos. En segundo lugar, se diseñó un algoritmo en el que se utilizan los comportamientos previamente definidos para dar con una solución. En tercer lugar, se analizó la necesidad de un analizador sintáctico-léxico, fundamental para el funcionamiento de la herramienta. Por último, se estudiaron las diversas funcionalidades que se podían ofrecer para lograr un marco didáctico en la herramienta.

2.1 Tipos de Datos Abstractos e implementación.

En la planificación del desarrollo de la herramienta y durante el desarrollo de la misma, surgió la necesidad de especificar ciertos tipos de datos abstractos(TDA). Esto fue realizado para facilitar el trabajo relacionado con la identificación de estructuras y conceptos teóricos esenciales para el entendimiento de los formalismos que se estaban trabajando. Luego, a partir de este estudio de alto nivel de las bases teóricas, se procedió a utilizar los mismos en el desarrollo de los algoritmos de la herramienta. Los tipos de datos más importantes que se definieron, con su respectiva especificación algebraica [4], fueron los TDA Fórmula y TDA Árbol [8].

Este primer paso dio lugar a la fase de implementación de estos tipos de datos que se materializaron en clases en C++. La clase *Fórmula*, valga la redundancia, representa a una fórmula lógica. Su diseño surge a la hora de identificar a todas las operaciones lógicas asociadas tanto a la Lógica de Predicados de Primer Orden como a la Lógica Proposicional. Esta clase es una clase abstracta, compuesta por métodos virtuales puros. De esta definición, surgen las clases *Conjunción*, *Disyunción*, *Condicional*, *Negación*, *ParaTodo*, *Existe*, *Predicado* y *Proposición*. Todas estas clases heredan de k. El comportamiento de estas clases, implementado en sus métodos y atributos privados, varía según las características propias de cada fórmula lógica. Todas estas clases deben implementar el método que les permita agregarse al Árbol de Refutación en construcción. La manera de agregarse, a alto nivel, es según al tipo de regla a la que esté asociada la fórmula lógica, según la Figura 1:

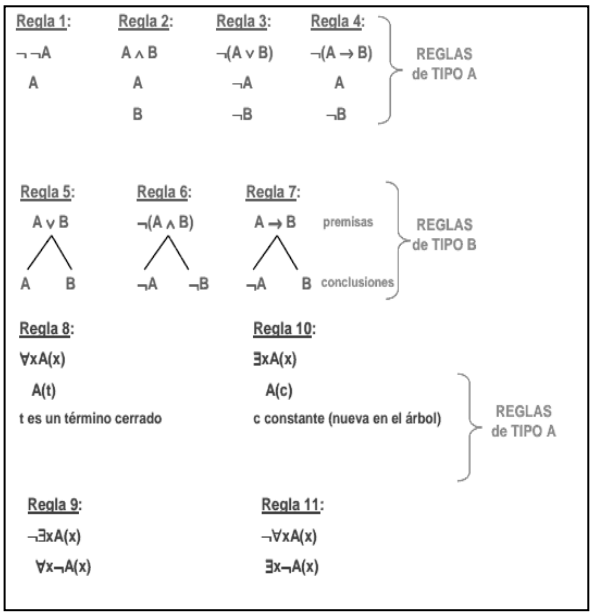

Figura 1

Por su parte, la clase Arbol define la estructura sobre la que se construirá el Árbol de Refutación. La estructura se implementa como un árbol ternario. Es necesario aclarar que este árbol nunca será completo. Es decir, un nodo del árbol podrá tener completos sus dos hijos "extremos" o bien su hijo "medio". Esta decisión de implementación se basa en la intención de priorizar un aspecto de visualización y comprensión de la estructura que almacena los datos. Esta clase guarda una serie de estructuras de almacenamiento internas para almacenar información necesaria en la construcción del árbol. También, principalmente, define los métodos públicos que permiten a un objeto de la clase *Formula* agregarse al árbol en construcción según el tipo de regla que deba aplicarse.

2.2 Algoritmo constructor del Árbol de Refutación**.**

El algoritmo para poder construir eficazmente un Árbol de Refutación se implementó a partir de la utilización de métodos de la clase Arbol [8]. El método que implementa la solución es el de *AgregarDisponible*. Este método permite ir agregando fórmulas al árbol en construcción (según su forma de agregarse) e ir completando el mismo hasta poder o no cerrar sus ramas.

Sin embargo, la herramienta yaT³ también ofrece la posibilidad de que sea el mismo usuario quien pueda ir decidiendo cómo se va expandiendo el árbol según las fórmulas que apliquen su regla de expansión correspondiente. Esta funcionalidad, se implementa mediante el método *AgregaDisponiblePorFormula*. De esta manera el usuario construye el árbol a su medida y no a partir de un vector de disponibilidad que contiene las fórmulas existentes que pueden ser agregadas al árbol.

2.3 Analizador sintáctico-léxico**.**

Para poder trabajar una determinada fórmula a partir de un texto introducido por el usuario se utilizó un analizador sintáctico y léxico. Este requerimiento derivo en el aprendizaje de distintas herramientas y conceptos necesarios para poder darle semántica a una cadena de texto ingresada por el usuario. Para ello se incursionó en el aprendizaje y utilización de dos herramientas, Flex [10] y Bison [6], para el análisis léxico y sintáctico respectivamente.

Bison es un generador de analizadores sintácticos que convierte una descripción para una gramática libre del contexto en un programa en C que analiza esa gramática. Una fuente de Bison (fichero con extensión .y) describe una gramática. El archivo que

se genera indica si un archivo de entrada (que en este caso contiene un string) pertenece o no al lenguaje generado por esa gramática.

Las declaraciones de Bison definen los nombres de los símbolos terminales (tokens) y no terminales, la precedencia de los operadores y los tipos de datos de los valores semánticos de los símbolos. Las reglas gramaticales son las producciones de la gramática que también llevan asociado código en C, y se ejecuta cuando el analizador encuentra las reglas correspondientes.

Por su parte Flex es una herramienta que permite generar analizadores léxicos.

A partir de un conjunto de expresiones regulares, Flex busca concordancia en un archivo de entrada (con extensión .l) y ejecuta acciones asociadas a estas expresiones.

Para el desarrollo del trabajo, se implementaron dos analizadores o parsers: uno para la lógica proposicional y otro para la lógica de predicados.

Los tokens identificados fueron los símbolos utilizados en el curso a la hora de escribir fórmulas. Se respetó la notación para que el alumno ya se encuentre familiarizado con la forma de ingresar una nueva fórmula y no deba invertir tiempo en un nuevo aprendizaje.

Las gramáticas mencionadas formalizadas en la notación BNF, tanto para la Lógica Proposicional como para la Lógica de Predicados de Primer Orden, se encuentran en las Figuras 2 y 3 respectivamente.

<comienzo>::= <condicional>
<condicional>::= <disyuncion> ⟹ <condicional>||<disyuncion>
<disyuncion>::= <conjunción>∨<disyuncion>||<conjunción>
<conjunción>::= <proposición>∧<conjunción>||<proposición>
<proposición>::= IDPROP ||¬ IDPROP ||(<condicional>)

Figura 2

<comienzo>::= <condicional>
<condicional>::= <disyuncion> ⟹ <condicional>||<disyuncion>
<disyuncion>::= <conjunción>∨<disyuncion>||<conjunción>
<conjunción>::= <factor>∧<conjunción>||<factor>
<factor>::= PARATODO IDVAR <factor>||EXISTE IDVAR <factor>||¬<factor>||
(<condicional>)||<predicado>
<predicado>::= IDPRED (<terminos>)
<terminos>::= <terminos>||<terminos>,<termino>
<termino>::= IDVARIABLE||IDFUNCION (<terminos>)

Figura 3

La interacción del parser con las clases que se involucran en la construcción del árbol se realiza mediante el método *parse*. El método parse se encuentra en la clase *EscenaArbol*. Esta clase tiene una fuerte interacción con el usuario, con la interfaz gráfica. Lo que hace este método es construir un árbol de sintaxis abstracta a partir de la cadena de texto ingresada por el usuario.

Es importante mencionar que en el header de la clase se encuentran una serie de variables y funciones externas, que se definen de esa manera ya que se utilizan en otras clases (ambas clases generadas por los parsers y lexers de la Lógica Proposicional y la de Predicados). Estas variables externas son utilizadas por el método *parse* para, a partir de una cadena de texto (fórmula) ejecutar el método *yyparse* y asignar un puntero a la raíz del árbol de sintaxis abstracta (*AST* o *ASTprop*, según corresponda a la Lógica de Predicados de Primer Orden ó a la Lógica Proposicional), es decir, el árbol que genera el parser.

Tanto *AST* como *ASTprop* son punteros al árbol de parse. Este árbol tiene una estructura jerárquica, que se corresponde con la precedencia de los operadores de la lógica. A partir de este árbol, podemos ir construyendo el Árbol de Refutación, ya que haciendo recorridos sobre él podemos obtener información de sus nodos.

A modo de ejemplo, luego de analizar la fórmula de entrada $P \vee \neg Q$ el *AST* quedará de la forma indicada en la Figura 4.

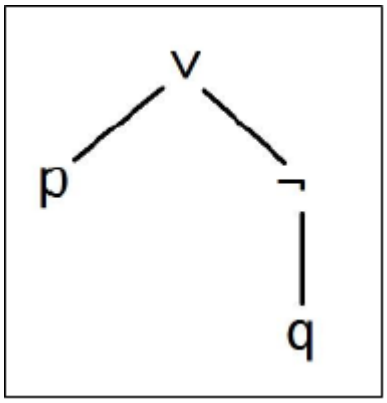

Figura 4

2.4 Interfaz gráfica y funcionalidades**.**

El entorno de desarrollo elegido para trabajar fue el QtCreator 2.8.0, utilizando Qt 4.8.5 [3]. A su vez se utilizó una amplia variedad de librerías provistas por el mismo entorno. El lenguaje utilizado fue C++ y también se aprovecharon facilidades provistas por la librería estándar STL [9].

Como la idea del proyecto siempre fue la de crear una herramienta didáctica, se trabajaron cuestiones relacionadas a la interacción entre el usuario y el programa. El aprendizaje del desarrollo de interfaces gráficas fue un punto importante en el desarrollo del trabajo, en relación al tiempo necesario para asimilar los conceptos básicos que se requieren para lograr una interacción fluida con el usuario.

Como primera medida, se introdujeron tres modos distintos de completar el árbol que se construye.

El primero de ellos consistía en ofrecer, a partir de una fórmula ingresada por el usuario, el Árbol de Refutación construido. El mismo se iba imprimiendo en pantalla automáticamente hasta completarse. Este método, sin embargo, no ofrecía una funcionalidad didáctica más allá de dar una respuesta a un problema planteado.

El segundo modo, consistió en ofrecer al usuario la posibilidad de generar el árbol paso por paso, una regla a la vez. De esta manera, podría ir verificando el desarrollo del mismo y proceder a un análisis más lento de los pasos (reglas) que se estaban ejecutando.

El tercer modo, y de más interacción con el usuario, consiste en que el mismo decida la forma en que se genera el árbol. El método consiste en identificar con un color a las fórmulas que están disponibles para agregarse al árbol y que el usuario decida, a partir de un click, cuál es la que desea agregar al mismo.

De esta manera, se puede generar el árbol a partir de la experimentación del usuario. El usuario altera la aplicación de reglas que implementa la herramienta y logra algo más cercano a lo que puede desarrollar con lápiz y papel. Le añade una cuota de inteligencia en relación a la practicidad de la aplicación de reglas.

Como información adicional, el usuario tiene la posibilidad inmediata de acceder a una breve reseña de las reglas para agregarse al árbol y a otra similar con la explicación de la sintaxis que utiliza la herramienta.

El usuario tiene disponibilidad de utilizar los símbolos introducidos por la cátedra a partir de botones interactivos, para evitar complicaciones en la escritura de los mismos a partir del teclado.

Los documentos generados se pueden guardar y también se puede generar un *.pdf* o un *.png* con el árbol generado.

Con respecto a la interfaz, la herramienta se constituye a partir de una ventana principal, a partir de la cual se pueden abrir tantas pestañas como árboles se quieran construir. La implementación interna de esto se distribuye en las clases *VentanaPrincipal*, *VentanaArbol*, y *EscenaArbol* dentro del proyecto.

Como se mencionó anteriormente, la clase *EscenaArbol* es la que mayor interacción tiene con el algoritmo diseñado para resolver el problema.

Para permitir una reutilización de las fórmulas ingresadas, se agregaron opcionesde guardado de las mismas, como también de apertura desde archivos. El formato de guardado es muy simple; se guarda la fórmula, seguido de un texto que puede variar entre *PRE* o *PRO* (haciendo referencia al tipo de árbol, es decir, PREdicados de primer orden o PROposiciones), luego se guardan las constantes (en caso de que haya, y que corresponda), y por último el número de constantes para poder separarlas de la verdadera fórmula en una futura apertura del archivo.

3 Resultados

La herramienta yaT³ se encuentra funcionando en condiciones óptimas. Fue testeada con los ejercicios planteados en el dictado de una materia que trata la temática de Árboles de Refutación. También fue probada con ejercicios de evaluación, en los que se suele profundizar la complejidad.

Si bien la herramienta va a empezar a utilizarse en la materia Ciencias de la Computación II en el primer cuatrimestre del próximo año, según la opinión de algunos alumnos de la carrera que ya la han utilizado, el software resultó simple de usar, les permitió emplear los conceptos estudiados en el curso y comprobar sus resultados, participando así más activamente en su proceso de enseñanza y aprendizaje.

4 Discusión

El desarrollo de la herramienta se encuentra en el contexto de una serie de trabajos de cátedra sobre los cuales se basa este proyecto [5].

La esencia de estos trabajos es la cumplir con una finalidad didáctica y la herramienta yaT³ cumple con estas premisas a partir de una fuerte interacción con el usuario y una interfaz intuitiva, respetando los formalismos dictados en el curso.

5 Conclusión

El desarrollo de la herramienta yaT³ fue satisfactorio y cumplió con los objetivos planteados inicialmente. Es una herramienta desarrollada por alumnos y para alumnos, hecho motivador en el proceso de aprendizaje y enseñanza. En todo punto de este trabajo se priorizó el fin didáctico que perseguía una posterior utilización de la herramienta. A su vez se produjo un gran aprendizaje mediante el uso de nuevas tecnologías involucradas en la solución.

Las etapas de desarrollo significaron una profundización de los conceptos vistos en las materias para las cuales se realizó este trabajo. Ambas materias eran de distinta índole. Se logró un trabajo integrador a partir de la utilización de conocimientos de programación para desarrollar una herramienta que trabaje con conceptos teóricos de lógica.

La identificación de los tipos de datos que intervenían en la solución ideada fue de vital importancia en la posterior implementación en C++. La abstracción y delegación de comportamiento y responsabilidades de cada entidad que participaba en la solución de la problemática permitió una implementación que sentó las bases en una primera introducción al paradigma orientado a objetos.

El diseño de un algoritmo que permitiese la construcción de un Árbol de Refutación fue una de las partes centrales del proyecto. Su implementación mantuvo el foco en una utilización eficiente de las estructuras de datos creadas para guardar la información necesaria para dar con una solución.

Por otra parte, se incursionó en el aprendizaje de las herramientas de análisis léxico y sintáctico. Estas herramientas eran clave para forjar una interacción con el usuario y que la herramienta puede ser realmente utilizable por alumnos. Como es una herramienta libre en un contexto académico, se utilizó siempre software libe y buscando un soporte de documentación que permita una rápida consulta de las herramientas usadas.

En cuanto a la funcionalidad de yaT³, el objetivo siempre fue que el aprendizaje de los conceptos teóricos que intervienen en la misma sea más llevadero y sencillo de utilizar para el usuario. Se trabajó en una herramienta que complemente con la forma tradicional de aprendizaje y pueda ser utilizada también para una profundización de la práctica del tema.

6 Referencias

1. Apuntes de Cátedra: Ciencias de la Computación II: http://ccomp2.alumnos.exa.unicen.edu.ar/teorias
2. BEN-ARI, Mordechai. Mathematical Logic for Computer Science. Third Edition, 2003.
3. CELANI, Sergio. Apuntes de Cátedra: Ciencias de la Computación II.
4. CORMEN, Thomas. LEISERSON, Charles. RIVEST, Ronald. STEIN, Clifford. Introducction to Algorithms. Third Edition, 2009.
5. DIGIA. Documentación oficial sobre el entorno gráfico QT: http://qtproject.org/doc/
6. FAVRE, Liliana. FELICE, Laura. Apuntes de Cátedra: Análisis y Diseño de Algoritmos I.
7. GERVASONI, Luciano. MAGGIORI, Emmanuel. FOLST: Una herramienta didáctica para la Lógica de Predicados de Primer Orden.
8. GNU Operating System. Documentación oficial sobre la herramienta Bison: http://www.gnu.org/software/bison/bison.htmlSGI. Documentación oficial sobre la librería STL: http://www.sgi.com/tech/stl/
9. The Flex Project. Documentación oficial sobre la herramienta Flex: http://ex.sourceforge.net/