Sistemas Informáticos
Curso 2006-2007

---

# *Una Herramienta para el Testeo de Propiedades Formales de Programas Basada en Técnicas de Demostración Automática*


Javier López Cuadrado
Sergio Pérez Jiménez
Alberto Sánchez Mazarro

Dirigido por:
Prof. Rafael del Vado Vírseda
Dpto. Sistemas Informáticos y Computación


---


Facultad de Informática
Universidad Complutense de

Madrid

# RESUMEN EN CASTELLANO E INGLÉS

La aplicación permite demostrar la corrección de argumentaciones tanto en lógica proposicional como en lógica de primer orden por el método de los tableaux. El usuario puede introducir un conjunto de premisas o cláusulas y comprobar si de ellas se deduce una determinada propiedad que introducirá en la conclusión. La aplicación generará el árbol correspondiente y deducirá del mismo si la argumentación válida o falaz.

Por otra parte, permite la demostración de propiedades muy sencillas de algunos programas. Comprueba (también con el método de los tableaux) si de una precondición, una postcondición y un cierto valor (o rango de valores) de las variables de un programa se puede inferir una propiedad.

El uso de la aplicación es sencillo y está pensado para cualquier tipo de usuario interesado en el estudio de propiedades lógicas. No se requieren grandes conocimientos para poder hacer uso de ella.

The application allows to as much demonstrate the correction of argumentations in propositional logic as in logic of first order by the method of tableaux. The user can introduce a set of premises or clauses and verify if of them a certain property is deduced that will introduce in the conclusion. The application will generate the tree corresponding and will deduce if the argumentation is valid or not.

On the other hand, it allows the demonstration of very simple properties of some programs. It verifies (also with the method of tableaux) if a precondition, a postcondition and a certain value (or rank of values) of the variables of a program can infer a property.

The use of the application is simple and it is thought for any type of user interested in the study of logic properties. Great knowledge is not required to be able to make use of it.

# ÍNDICE

# INTRODUCCIÓN Y MOTIVACIÓN

Una de las áreas de investigación y estudio de más tradición dentro del campo de la Informática Teórica y de la Inteligencia Artificial es la *Demostración Automática* [3] de teoremas y propiedades formales de programas. Básicamente, la Demostración Automática se ocupa de desarrollar métodos lógicos, fundamentados en algún (super/sub) conjunto de la *Lógica de Primer Orden* [2], de forma que permita establecer tanto pruebas formales de propiedades de programas como de implementar dichos métodos en algún lenguaje de programación que automatice tales pruebas.

En este contexto, el propósito del presente trabajo consiste en el desarrollo de una herramienta gráfica, utilizando un lenguaje de Programación Orientada a Objetos, que permita la comprobación automática de propiedades sencillas de programas a partir de su especificación formal, utilizando técnicas de Demostración Automática como el método de los *Tableaux Semánticos* [2].

## 1.1 Lógica de Predicados y Demostración Automática basada en Tableaux Semánticos

La lógica constituye la herramienta formal de razonamiento de la mayor parte de las asignaturas de la carrera de informática, sobre todo de las que están más relacionadas con las matemáticas y la programación, tales como el Álgebra, el Análisis Matemático, la Matemática Discreta, Electrónica Digital, Teoría de Autómatas y Lenguajes Formales, Programación (incluida la Programación Concurrente y la Programación Declarativa), Bases de Datos, etc. En cuanto a la inteligencia artificial, la lógica es el fundamento de todos los métodos de representación del conocimiento y del razonamiento, especialmente en sistemas expertos, razonamiento con incertidumbre (encadenamiento de reglas, lógica difusa, etc), procesado del lenguaje natural, razonamiento espacial y temporal, visión artificial, robótica, etc.

Además de definir criterios formales para reconocer los razonamientos válidos, la lógica debe suministrar métodos de deducción efectivos. El método de refutación que usaremos en este trabajo es el conocido como método de los *Tableaux Semánticos* [2]. Esta técnica utiliza principalmente el método de reducción al absurdo para poder derivar la validez de un razonamiento o de una fórmula, sin tener que tomar en consideración los valores de verdad de las fórmulas en estudio. Así, si se desea comprobar que una fórmula es consecuencia de otras, basta con negarla e incorporarla a esas otras. Si resulta insatisfactible este nuevo conjunto, efectivamente existía una relación de consecuencia.

El método de los tableaux semánticos tiene, entre otras, las siguientes ventajas:

- Es menos costoso de aplicar en muchos casos que otros mecanismos de Deducción Automática.
- Es una buena base para programar demostradores automáticos.
- Puede extenderse a otras lógicas más potentes que la lógica de proposiciones, para las cuales el método de las tablas de verdad deja de tener sentido.

## 1.2 Especificación y Verificación de Programas

Una adecuada metodología de programación debe seguir la secuencia de primero especificar el problema y después implementarlo. La *especificación de un programa* [4,5] debe expresar de forma breve y concisa qué se desea hacer. Permite al usuario razonar sobre las propiedades del programa y actúa como una barrera que permite independizar los razonamientos de corrección sobre los componentes de un programa.

Puesto que el lenguaje natural no resulta adecuado como lenguaje de especificación debido a sus imprecisiones, es posible utilizar una técnica de especificación formal de procedimientos y funciones basada en la *lógica de predicados* conocida *como técnica pre/post* [4,5]. Consiste en proporcionar dos predicados, llamados *precondición* y *postcondición*. El primero caracteriza las condiciones que deben cumplir inicialmente (antes de la ejecución) para que el programa funcione correctamente y el segundo describe las condiciones que se cumplen una vez ha finalizado el programa siempre que se cumplieran las condiciones de la precondición. El par puede verse como un contrato entre el implementador de un algoritmo y el usuario del mismo.

Los métodos formales de Deducción Automática que proporciona la Lógica [6], tienen una aplicación inmediata en la ingeniería del software. El uso de lenguajes de especificación formal es beneficioso en todos los desarrollos, ya que promueve la definición de modelos estructurados, concisos y precisos en diferentes niveles de abstracción, y facilita el razonamiento sobre ellos incluso a un nivel informal. Cuando a las notaciones formales se les asigna una semántica operacional, es posible diseñar herramientas automáticas que detecten ambigüedades en los requisitos iniciales, verifiquen y validen modelos a lo largo del ciclo de desarrollo, ayuden en la evolución y el mantenimiento de los productos y generen automática o semi-automáticamente prototipos o incluso partes del código final. En este sentido, los métodos formales sirven también para propósitos de documentación, ingeniería inversa y reutilización de componentes.

No obstante, aun siendo reconocidas las importantes contribuciones de la Deducción Automática y los métodos formales a la ingeniería del software, su uso no se ha extendido tanto como cabía prever hace un par de décadas.

Lo cierto es que los proyectos de desarrollo formal han fracasado con frecuencia, si bien por razones más ergonómicas que técnicas. La experiencia ha mostrado que introducir métodos formales en un proceso de desarrollo puede provocar dificultades no relacionadas con la potencia de los métodos. En primer lugar, pueden resultar excesivamente restrictivos, tanto porque obligan a sobreespecificar ciertos aspectos como porque constriñen el proceso de desarrollo, que tienen tendencia a monopolizar. En segundo lugar, su introducción puede ser problemática desde un punto de vista práctico, debido a su dificultad real y percibida, de forma que el coste inicial de entrenamiento en esta técnicas puede ser significativo, y en ocasiones los métodos son infrautilizados incluso por parte de ingenieros instruidos en su uso.

Para poder comprobar que un programa es correcto hace falta establecer con precisión su semántica. El hecho de que un programa tenga un error puede resultar sumamente costoso, no sólo en términos económicos, sino que en ciertos casos incluso puede poner en peligro la vida de muchos seres humanos. Por eso es importante disponer de métodos que permitan comprobar que un programa cumple las especificaciones con que fue diseñado.
Generalmente las especificaciones suelen venir dadas en lenguajes natural. Por ejemplo: "Quiero un programa que calcule las nóminas de mis empleados, a partir de los siguientes datos…". Naturalmente, dada la ambigüedad y falta de precisión del lenguaje natural y la ausencia de métodos que permitan una compresión automática del mismo, se hace necesario contar con descripciones formales que indiquen de forma precisa e inequívoca las especificaciones de cada programa. Una vez que se conocen las especificaciones, sería deseable contar con un método, implementable en un programa de ordenador, que generase automáticamente el programa buscado, libre de errores. Dado que esto es todavía ciencia-ficción, al menos sería deseable contar con métodos que permitan comprobar de forma automática o semiautomática que cierto programa cumple las especificaciones, es decir, que hace lo que se espera de él, sin cometer nunca errores; es lo que se conoce como verificación de programas.

Aún estamos lejos de contar con verificadores totalmente automáticos, pero recientemente se han desarrollado ya verificadores semiautomáticos para lenguajes de alto nivel. Vease, por ejemplo, el proyecto KeY, http://i12www.ira.uka.de/~key/, que han conseguido integrar especificación formal (en OCL) y verificación semiautomática en una herramienta CASE de modelado mediante objetos (UML), especialmente diseñada para programar en Java.

Los grandes avances que se han producido en las últimas tres décadas y los intereses de los fabricantes de software por garantizar la fiabilidad de sus productos hacen pensar que en los próximos años se seguirán produciendo progresos muy significativos en este campo. Precisamente por la importancia del tema, es de esperar que el conocimiento de los métodos de verificación formal sea una de las cualidades más valoradas de los ingenieros en informática de un futuro no muy lejano.

## 1.3 Testeo de Propiedades Formales mediante Tableaux Semánticos

El objetivo del presente trabajo para la asignatura de "Sistemas Informáticos" consiste en el desarrollo de una herramienta de Demostración Automática basada en el método de los tableaux semánticos de la lógica de predicados que permita la comprobación automática de propiedades sencillas de programas a partir de su especificación formal y la depuración de respuestas incorrectas y perdidas.

Para lograr este objetivo, en este trabajo se ofrece la implementación concreta en un lenguaje orientado a objetos de algunos de los algoritmos diseñados en Demostración Automática para el método de los tableaux semánticos sobre un (fragmento del) lenguaje de la lógica de primero orden. En particular, se han realizado implementaciones a partir de prototipos ya existentes que sólo utilizaban lenguajes de programación declarativos.

El segundo paso que se ha dado ha consistido en extender la implementación de forma que se pueda trabajar mediante ella sobre un lenguaje extendido de la lógica de primer orden que permita expresar cómodamente tanto la especificación formal de un programa como propiedades sencillas que representen aspectos relacionados con el comportamiento de ese programa. En este punto, las herramientas gráficas desarrolladas hacen posible utilizar la implementación para comprobar, depurar y verificar si ciertas propiedades son o no deducibles a partir de la especificación formal del programa.

Finalmente, la interfaz gráfica desarrollada a partir de la implementación dada del método de tableaux semánticos permite representar gráficamente, y de una manera lo más motivadora y pedagógica posible, las deducciones automáticas obtenidas sobre el comportamiento de los programas. De esta forma, se dispone también de una herramienta gráfica de un resultado visual y motivador de gran utilidad a la hora de entender tanto el funcionamiento del método de los Tableaux Semánticos como el funcionamiento de las especificaciones formales de los programas por parte de otros alumnos de los primeros cursos de la Ingeniería Informática como se pone de manifiesto en la publicación [1].

# ANÁLISIS Y DISEÑO

## 2.1.- Introducción

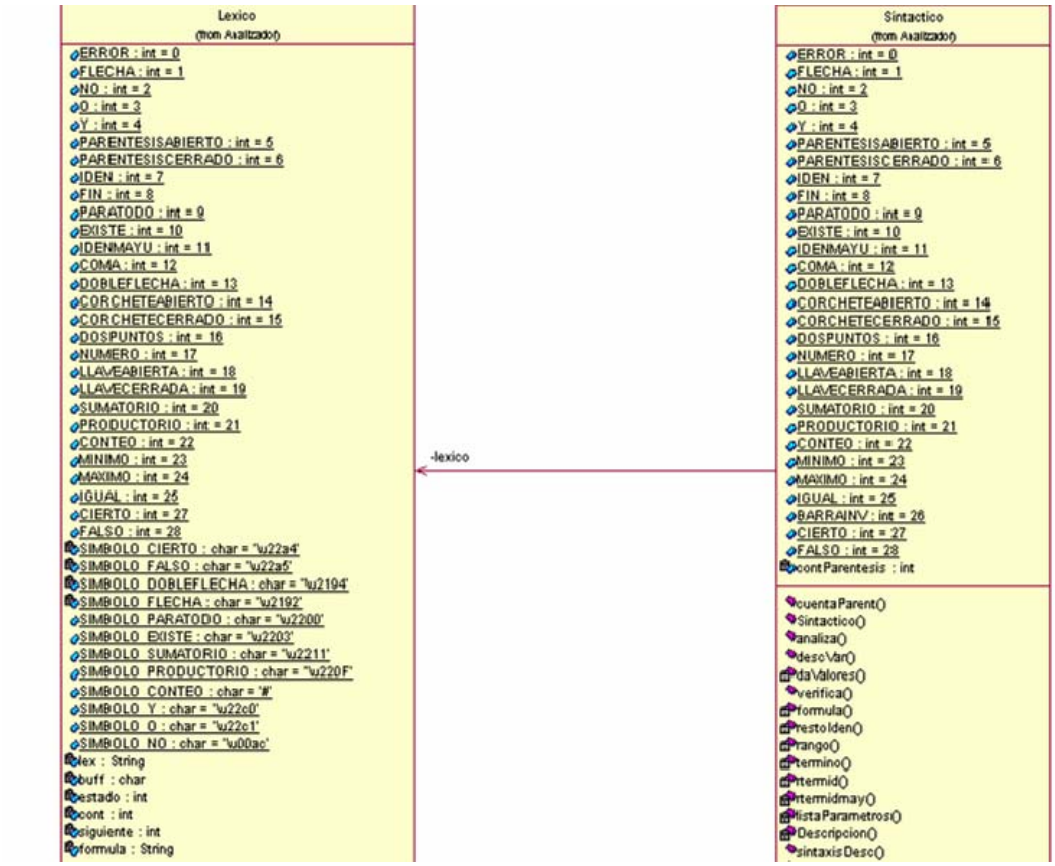
En el siguiente apartado se analizarán las diferentes partes del diseño de la herramienta. En el diseño podemos distinguir cuatro apartados claramente diferenciados, la parte del analizador sintáctico, la parte gráfica, la parte de las fórmulas y la parte de los tableaux. Se han estructurado cada una de estas partes en los siguientes paquetes:

- Paquete "Analizador": Se compone de una clase "Léxico" y una clase "Sintactico". Se encarga de controlar que las secuencias de entrada introducidas por el usuario (las fórmulas) estén escritas de manera adecuada y tengan un significado correcto.
- Paquete "Formulas": Contiene los interfaces y las clases que representan a cada uno de los tipos de fórmulas lógicas que pueden ser introducidas en la herramienta.
- Paquete "Tableaux": Contiene las clases necesarias para la representación del Tableaux. Se compone de dos clases, "NodoTableaux" que representa a un nodo del árbol, y "Tableaux" que representa el propio arbol.
- Paquete "GUI": Contiene las clases necesarias para la representación gráfica de la herramienta. Como son por ejemplo la ventana principal de la aplicación y los distintos paneles de los que se compone.
- Paquete "Idioma": Contiene una única clase que permite al usuario que la herramienta se muestre en uno de los dos idiomas que se han introducido. Los diccionarios de dichos idiomas también están contenidos en este paquete.
- Paquete "Info": Se compone de de una única clase "Info". Esta clase es utilizada para tener una recopilación de toda la información necesaria que los diferentes objetos van a necesitar para el recorrido del tableau, como son las variables utilizadas, las sustituciones producidas, las unificaciones establecidas…

En el siguiente apartado se explicarán más en detalle estos paquetes y se presentarán los diferentes diagramas de clase para su mejor comprensión.

## 2.2 - Diagramas de clases

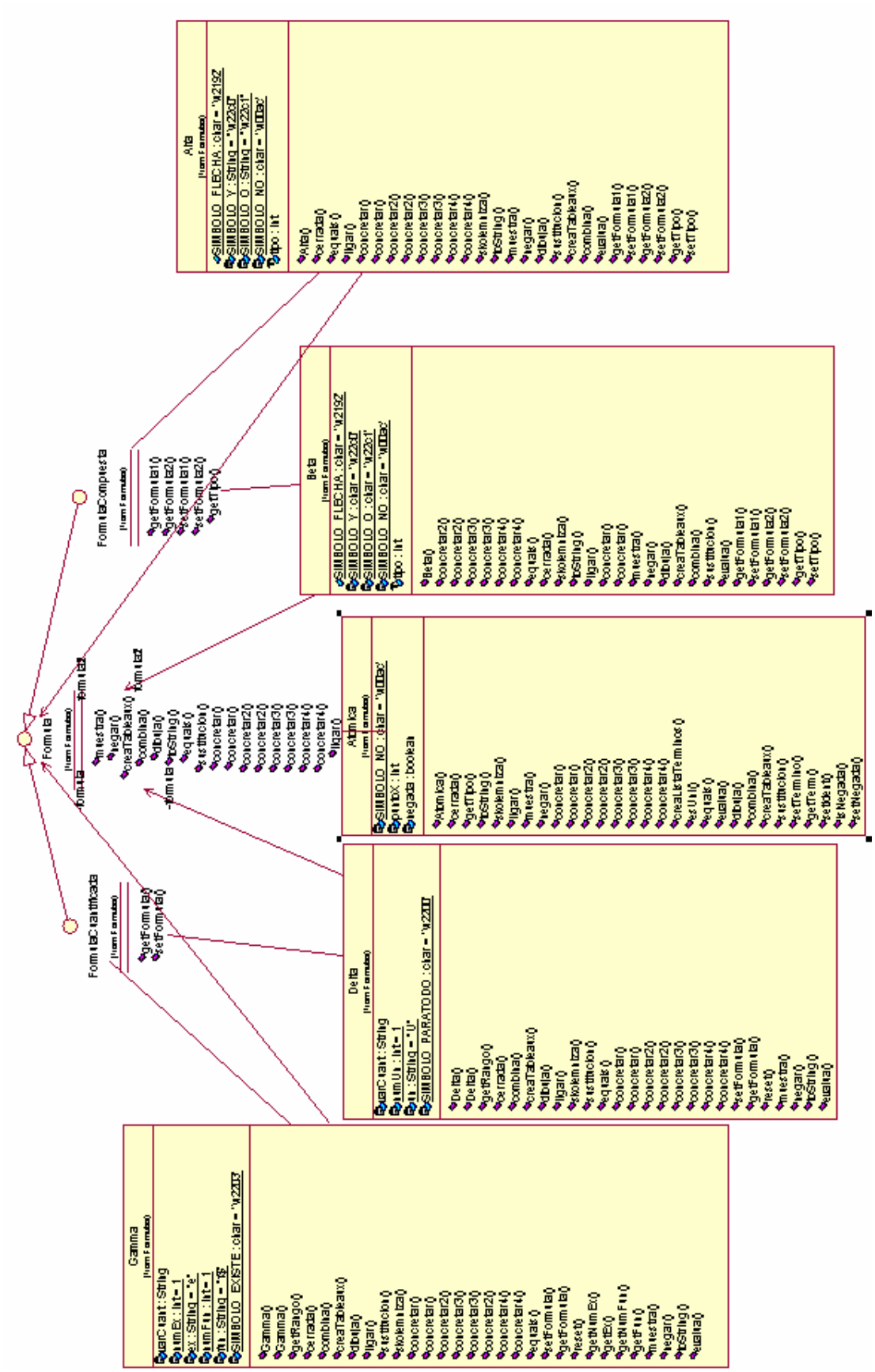
El siguiente diagrama UML representa la relación de clases del paquete "Analizador". Como se comento anteriormente estas dos clases se encargan de reconocer los caracteres introducidos por el usuario en las fórmulas ("Lexico") e interpretar si efectivamente se trata de una fórmula correcta o no ("Sintactico").



El siguiente diagrama UML representa una parte de la relación de clases del paquete "Formulas". Se tiene un interfaz principal "Formula" que contiene los métodos principales de una fórmula lógica que serán implementados por todos los tipos de formula que soporta la herramienta. Este interfaz es implementado directamente por la clase "Atomica" que representa a la fórmula elemental. El resto de formulas estarán compuestas por un número de estas fórmulas atómicas.

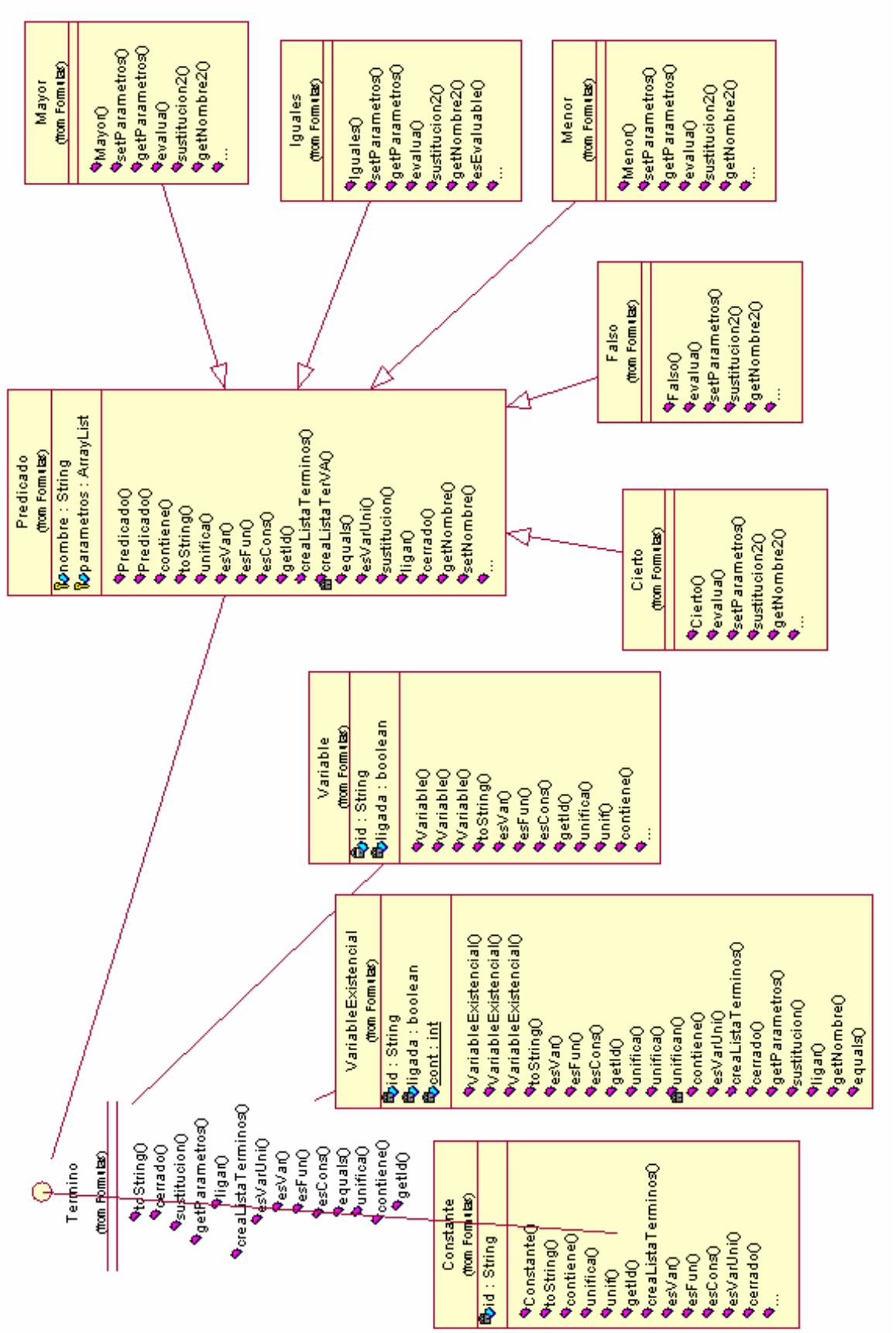
A su vez, se tienen otros dos interfaces que implementan el interfaz fórmula y que diferencia dos clases de fórmula como son las fórmulas compuestas (interfaz "FormulaCompuesta") y las fórmulas cuantificadas (interfaz "FormulaCuantificada"). Por fórmula compuesta se entiende que son fórmulas "alfa" (conjuntivas) y fórmulas "Beta" (disyuntivas). Así mismo por fórmulas cuantificadas se entiende que son las fórmulas "gamma" (existenciales) y las "delta" (universales).

En los apéndices de este documento se tiene información más detallada tanto de las fórmulas, como de los diferentes tipos que se utilizan.


Alta
Beta
Delta
Gamma
Atomica
FormulaCompuesta
FormulaConstruida
Formula

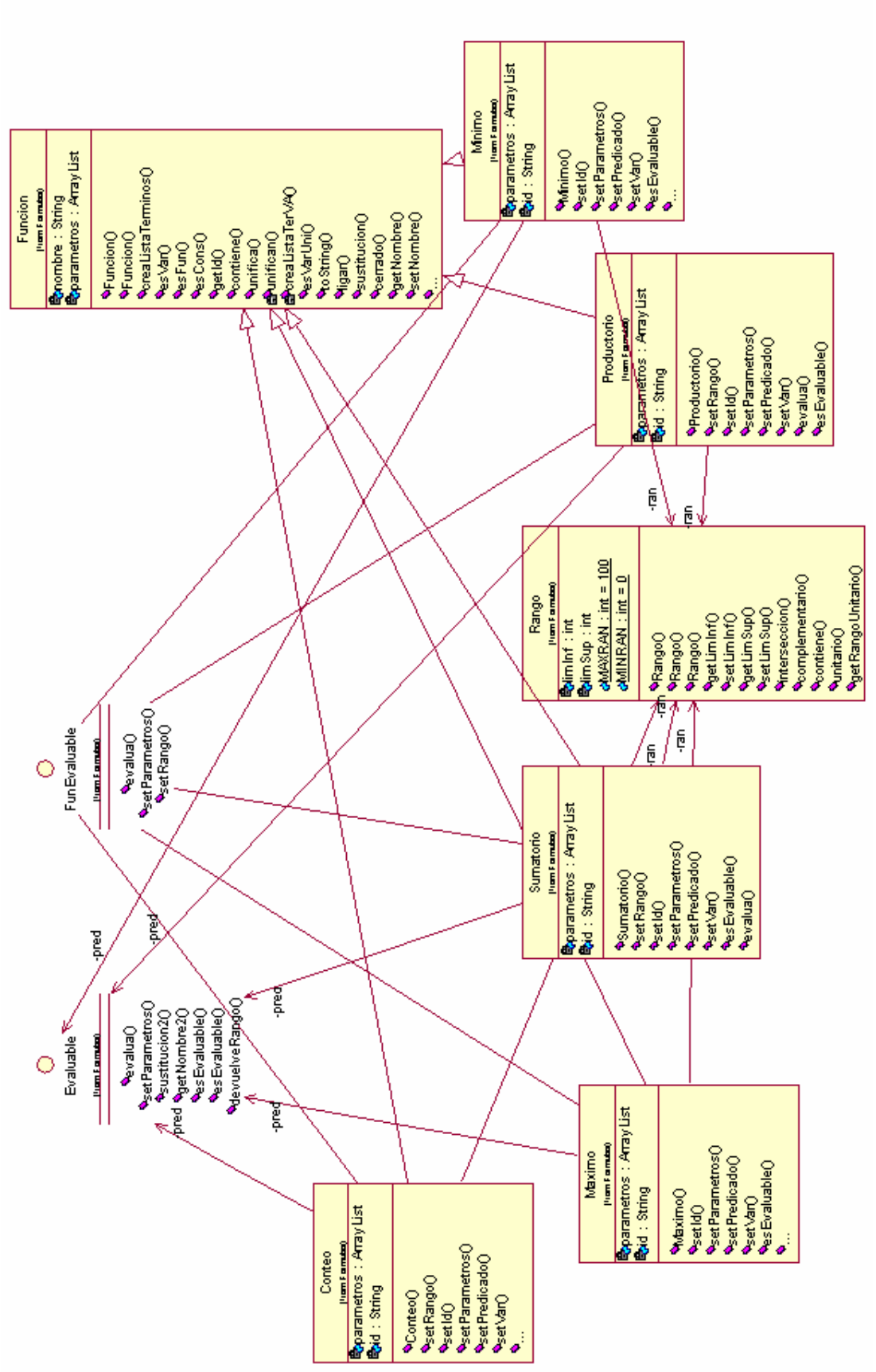
El siguiente diagrama UML es otra parte del paquete "Formulas". En él se representa la relación que hay entre las diferentes clases que conforman los términos de las fórmulas, es decir, los elementos primitivos que caracterizarán desde las fórmulas más sencillas, hasta las más complejas. Como se puede observar, se tiene un interfaz "Término" que directamente es implementado por las clases "Constante", "Variable", "VariableExistencial" y "Predicado".

A su vez, se tienen los distintos predicados que se han decidido implementar como son "Iguales", "Mayor", "Menor", "Cierto" y "Falso". Cada uno Cabe destacar, que en posibles ampliaciones de la herramienta se pueden introducir un mayor número de predicados con suma facilidad.


Mayor
(from Formulas)
Mayor()
setParametros()
getParametros()
evalua()
sustitucion2()
getNombre2()
...
Iguales
(from Formulas)
Iguales()
setParametros()
getParametros()
evalua()
sustitucion2()
getNombre2()
esEvaluable()
...
Menor
(from Formulas)
Menor()
setParametros()
getParametros()
evalua()
sustitucion2()
getNombre2()
...
Falso
(from Formulas)
Falso()
evalua()
setParametros()
sustitucion2()
getNombre2()
...
Predicado
(from Formulas)
nombre : String
parametros : ArrayList
Predicado()
Predicado()
contiene()
toString()
unifica()
esVar()
esFun()
esCons()
getId()
creaListaTerminos()
creaListaTerVA()
equals()
esVarUni()
sustitucion()
ligar()
cerrado()
getNombre()
setNombre()
...
Cierto
(from Formulas)
Cierto()
evalua()
setParametros()
sustitucion2()
getNombre2()
...
Variable
(from Formulas)
id : String
ligada : boolean
Variable()
Variable()
Variable()
toString()
esVar()
esFun()
esCons()
getId()
unifica()
unif()
contiene()
...
VariableExistencial
(from Formulas)
id : String
ligada : boolean
cont : int
VariableExistencial()
VariableExistencial()
VariableExistencial()
toString()
esVar()
esFun()
esCons()
getId()
unifica()
unifica()
unificar()
contiene()
esVarUni()
creaListaTerminos()
cerrado()
getParametros()
sustitucion()
ligar()
getNombre()
equals()
Termino
(from Formulas)
toString()
cerrado()
sustitucion()
getParametros()
ligar()
creaListaTerminos()
esVarUni()
esVar()
esFun()
esCons()
equals()
unifica()
contiene()
getId()
Constante
(from Formulas)
id : String
Constante()
toString()
contiene()
unifica()
unif()
getId()
creaListaTerminos()
esVar()
esFun()
esCons()
esVarUni()
cerrado()
...

El siguiente diagrama UML completa el paquete “Formulas”. En él se representan las diferentes funciones que se han decidido implementar. Todas ellas extienden de la clase “Funcion” e implementan a su vez el interfaz “FunEvaluable”.

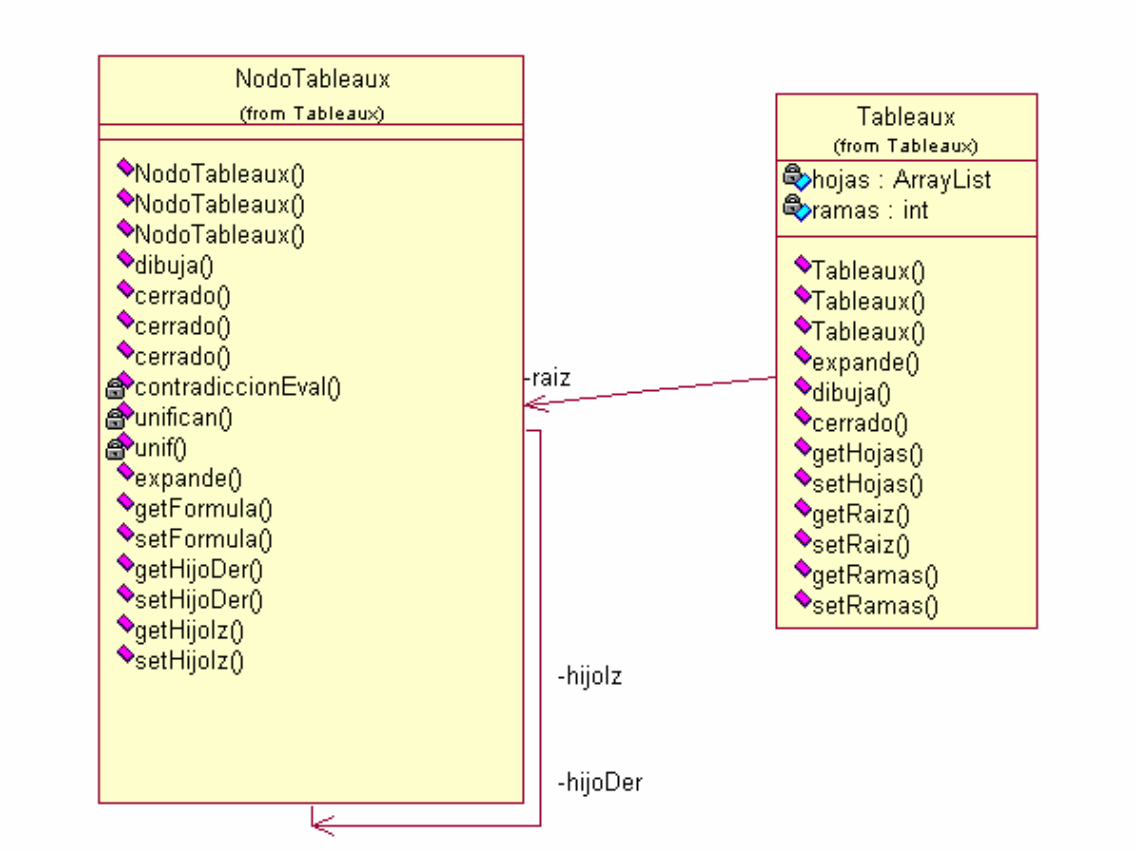
Algunas de las funciones que se han implementado son la función “Minimo”, “Maximo”, “Productorio”, “Sumatorio” y “Conteo”. Todas ellas se introducen con un rango de valores en los que se comprueba la propiedad que llevan asociada. Como se dijo anteriormente, en posibles ampliaciones de la herramienta se podrían introducir más funciones evaluables como pueden ser la suma, la multiplicación, la potencia…


Funcion
nombre : String
parametros : ArrayList
Funcion()
Funcion()
creaListaTerminos()
esVar()
esFun()
esCons()
getId()
contiene()
unifica()
unifican()
creaListaTerVA()
esVarUni()
toString()
ligar()
sustitucion()
cerrado()
getNombre()
setNombre()
Minimo
parametros : ArrayList
id : String
Minimo()
setId()
setParametros()
setPredicado()
setVar()
esEvaluable()
Productorio
parametros : ArrayList
id : String
Productorio()
setRango()
setId()
setParametros()
setPredicado()
setVar()
evalua()
esEvaluable()
Rango
limInf : int
limSup : int
MAXRAN : int = 100
MINRAN : int = 0
Rango()
Rango()
Rango()
getLimInf()
setLimInf()
getLimSup()
setLimSup()
interseccion()
complementario()
contiene()
unitario()
getRangoUnitario()
FunEvaluable
evalua()
setParametros()
setRango()
Evaluable
evalua()
setParametros()
sustitucion2()
getNombre2()
esEvaluable()
esEvaluable()
devuelveRango()
Sumatorio
parametros : ArrayList
id : String
Sumatorio()
setRango()
setId()
setParametros()
setPredicado()
setVar()
esEvaluable()
evalua()
Maximo
parametros : ArrayList
id : String
Maximo()
setId()
setParametros()
setPredicado()
setVar()
esEvaluable()
Conteo
parametros : ArrayList
id : String
Conteo()
setRango()
setId()
setParametros()
setPredicado()
setVar()
-pred
-ran

A continuación se muestra el diagrama de clases del paquete "Tableaux". En este diagrama UML se representan las relaciones entre las dos únicas clases de las que se compone este paquete.

Con estas dos clases se intenta representar la estructura de árbol por la que son característicos los tableaux. Así, se tiene un clase "NodoTableaux" que representa un nodo del árbol. Esta clase esta compuesta por una fórmula atómica y por dos hijos (izquierdo y derecho) que a su vez son "NodoTableaux".

Por último, la clase "Tableaux" representa toda la estructura del árbol a partir de un nodo raiz y de una lista con todas sus hojas además del número de ramas.



Por último, se tiene el diagrama UML que representa la parte gráfica de la herramienta, el paquete "GUI". En este paquete se tiene una clase "Ventana" que representa la ventana principal de la aplicación. En ella se encuentran los menús así como los diferentes paneles.

La clase "PanelArbol" representa el panel donde va a ser dibujado y mostrado por pantalla el árbol que resulta de las argumentaciones introducidas por el usuario.

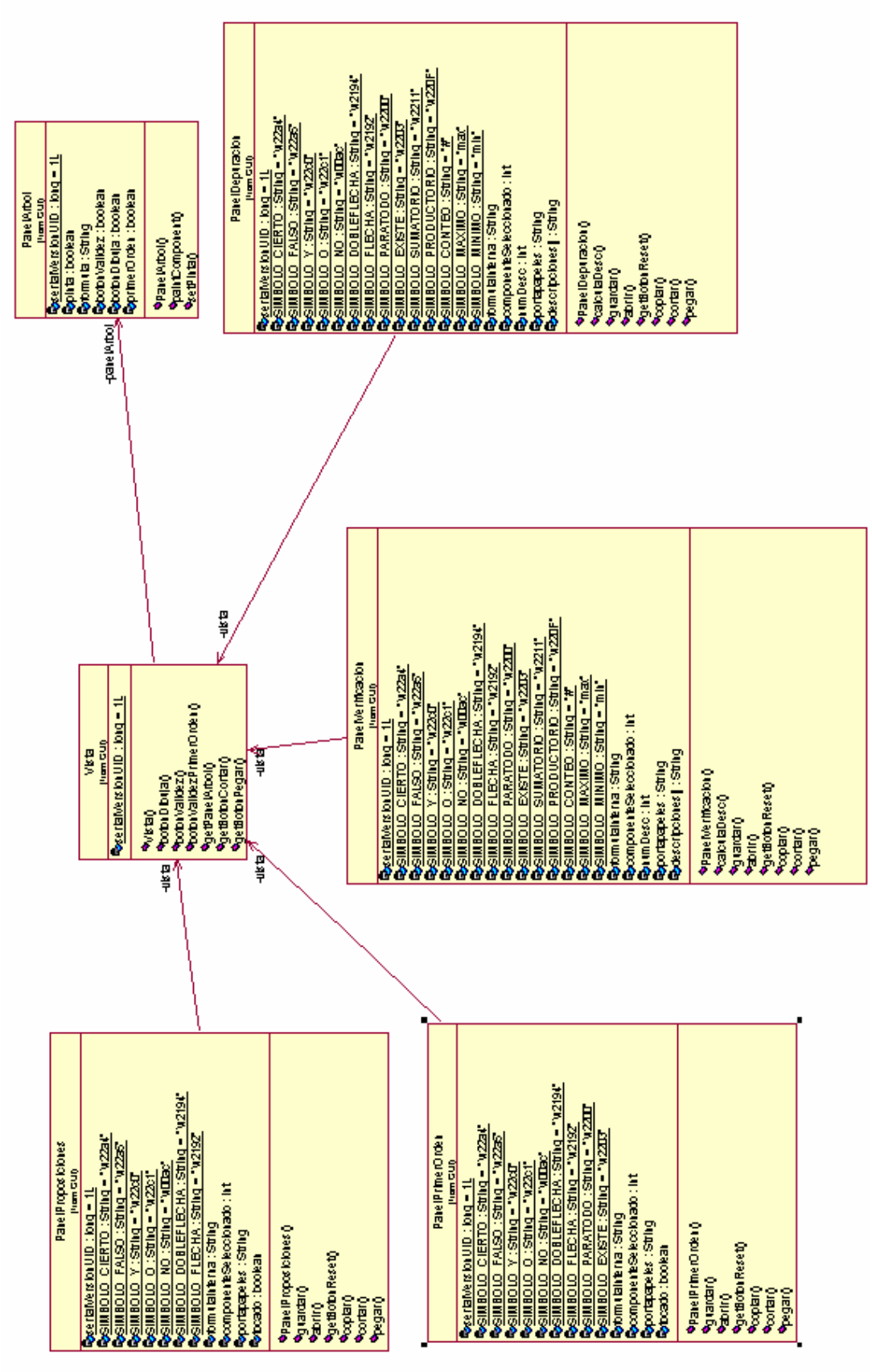
La clase "PanelProposiciones" representa el interfaz gráfico de la lógica proposicional.

La clase "PanelPrimerOrden" representa el interfaz gráfico de la lógica de primer orden.

La clase "PanelVerificación" representa el interfaz gráfico de la parte de verificación de programas.

La clase "PanelDepuración" representa el interfaz gráfico de la parte de depuración de programas.

Más adelante se explicarán cada uno de estos paneles con detalle.



# IMPLEMENTACIÓN Y ASPECTOS TÉCNICOS

## 3.1 - Plataformas y tecnologías utilizadas

En la construcción de esta herramienta se ha utilizado el lenguaje de programación JAVA bajo el entorno de desarrollo Eclipse. Se han utilizado paquetes de las librerías versión jre1.5.0_05 o inferiores, así como Java Swing para la parte gráfica de la herramienta. Ambas tecnologías son de distribución libre y han sido utilizadas tanto en los propios ordenadores personales de cada componente del grupo como en los laboratorios de la Facultad de Informática de la Universidad Complutense de Madrid.

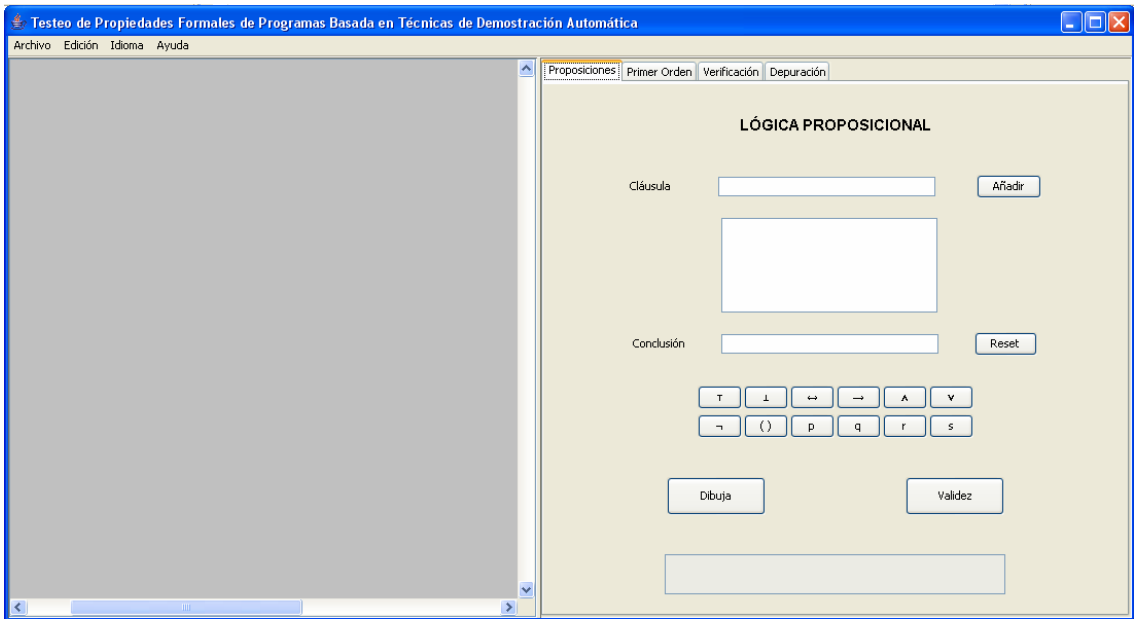
Para la realización de los diagramas UML así como para realizar el diseño de la herramienta se ha utilizado Racional Rose Enterprise de IBM, con la licencia obtenida por la Facultad de Informática de la Universidad Complutense de Madrid.

Por último, se ha utilizado un servidor CVS a través de la empresa Berilos, que ofrece este servicio de forma libre y gratuita.

## 3.2 - Parte interactiva y paneles gráficos

La interfaz de la aplicación consta de una única ventana principal que contiene los menús y los diferentes paneles gráficos sobre los que trabajará el usuario.

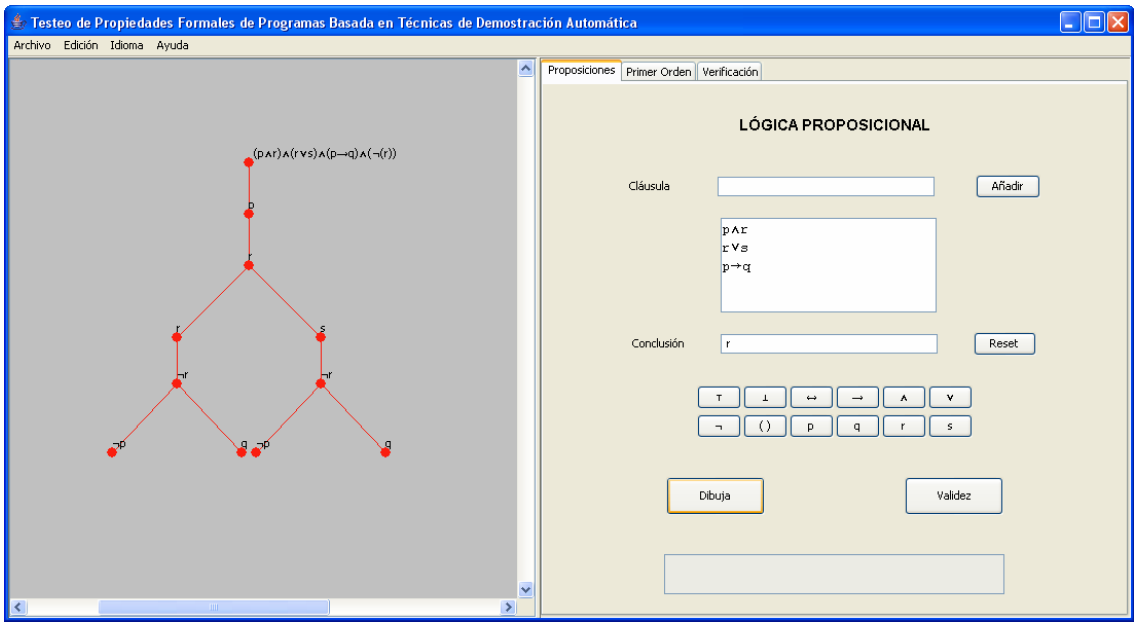
A continuación se muestra una primera vista del programa, sobre la que explicaremos las diferentes partes de las que esta compuesta la ventana principal:




(Figura 1)

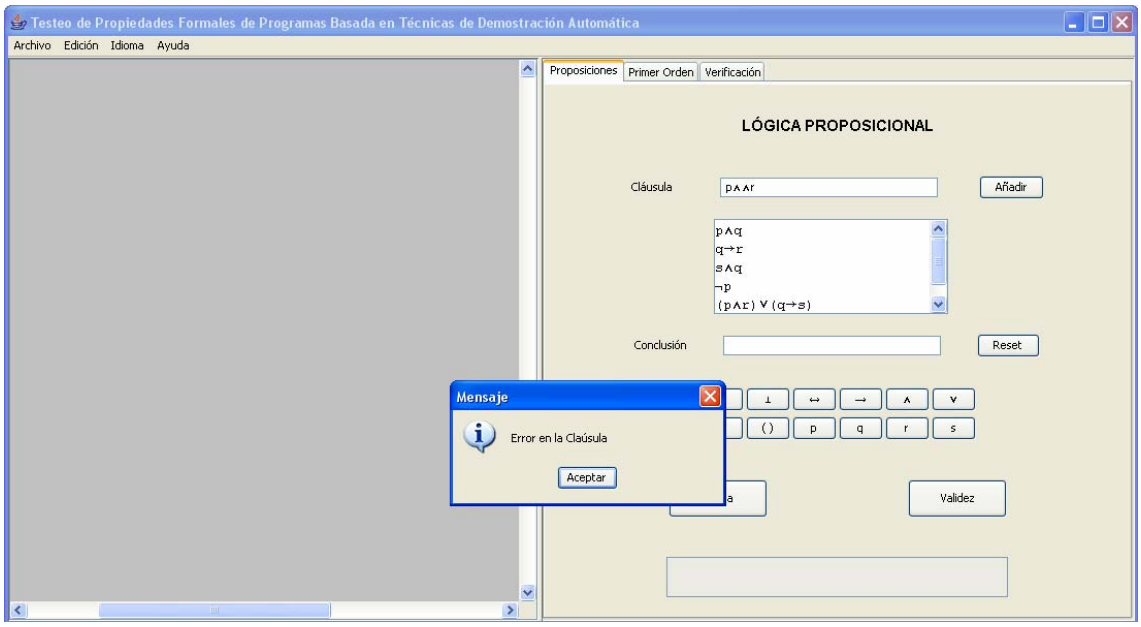
En primer lugar se observa que la ventana está dividida en dos partes bien diferenciadas, parte izquierda y parte derecha. La parte izquierda esta formada por un panel gráfico donde se dibuja el tableau resultante de la expresión que introduzca el usuario. La parte derecha está formada por tres paneles organizados en pestañas que representan cada uno de los niveles lógicos que se han decidido implementar para esta aplicación, lógica de predicados, lógica de primer orden y verificación de programas. Aquí es donde el usuario introduce las cláusulas de su argumentación.

Como se ha dicho anteriormente, en el panel izquierdo se dibuja el árbol que representa el conjunto de cláusulas introducidas por el usuario en el panel derecho. También se han introducido una barra de desplazamiento vertical que se muestra en todo momento, pero que sólo se habilita cuando el tableau excede la altura del panel y una barra de desplazamiento horizontal que permite al usuario ver correctamente el tableau cuando la anchura del árbol se hace demasiado grande. El resultado se muestra en la siguiente imagen.




(Figura 2)

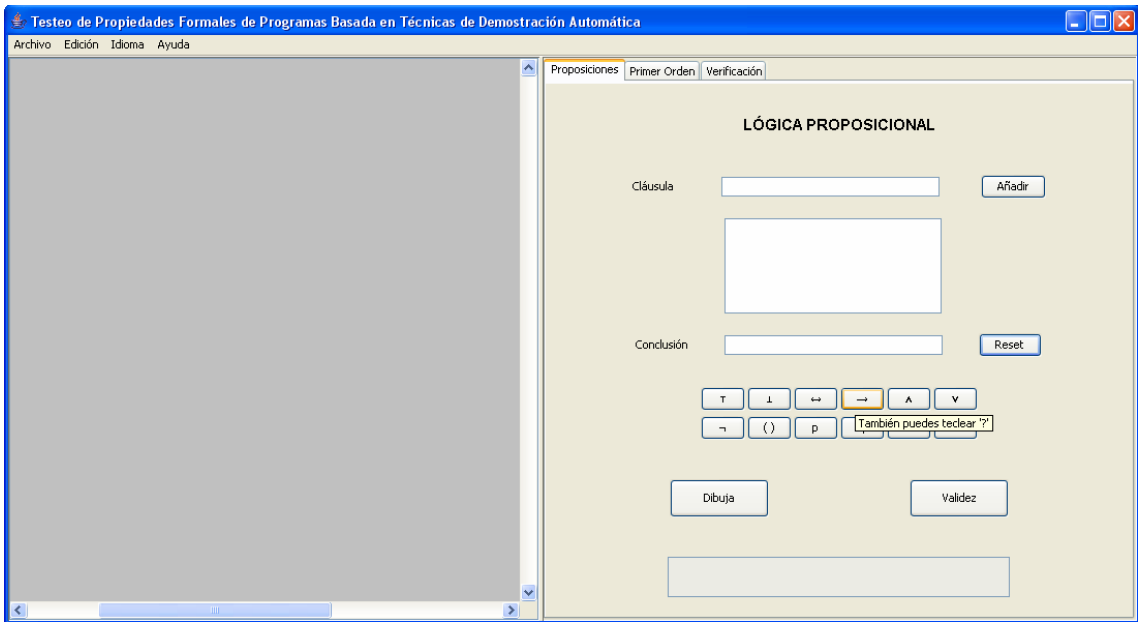
El primero de los paneles de los que se compone el panel derecho de la ventana principal es el de lógica de predicados. El usuario deberá introducir las premisas de su argumentación a través del campo cláusula, una a una, y se tendrán que ir añadiendo al área de texto a través del botón "Añadir". Cada vez que se añade una fórmula al área de texto, se comprueba que la fórmula haya sido introducida correctamente en el campo cláusula, sino la propia aplicación indicará al usuario a través de un mensaje en la pantalla que tiene que corregirla y no la añadirá al área de texto (Figura 3). El usuario podrá introducir el número de premisas que desee en el área de texto y automáticamente se activaran las barras de desplazamiento vertical y horizontal cuando sea necesario (Figura 3). Si por el contrario, el usuario decide no introducir ninguna fórmula en el área de texto, la herramienta tomará como cláusula "cierto", pudiéndose comprobar únicamente la veracidad de una conclusión (Figura 6).




(Figura 3)

La conclusión de la argumentación, debe ser introducida en el campo de texto con el mismo nombre. En este caso, la aplicación analiza que la fórmula haya sido introducida correctamente, cuando el usuario pulsa los botones de "Dibuja" o "Validez". Del mismo modo que ocurría anteriormente con las cláusulas, si el usuario introduce de forma incorrecta la conclusión, la herramienta mostrará un mensaje por pantalla indicando al usuario que debe corregir la fórmula.

Para una mayor comodidad a la hora de introducir las fórmulas y dado que algunos de los caracteres que se utilizan para la lógica, no pueden ser introducidos por teclado directamente, se dispone de una matriz de botones básicos. Con ellos el usuario podrá introducir fácilmente en la herramienta sus fórmulas lógicas. No obstante, han sido habilitadas algunas teclas que en principio no tienen por que ser usadas en la construcción de las fórmulas, para introducir por teclado estos caracteres especiales en la aplicación. Esta información se muestra a través de un mensaje de ayuda cuando el usuario lleva el cursor del ratón sobre uno de estos botones (Figura 4).



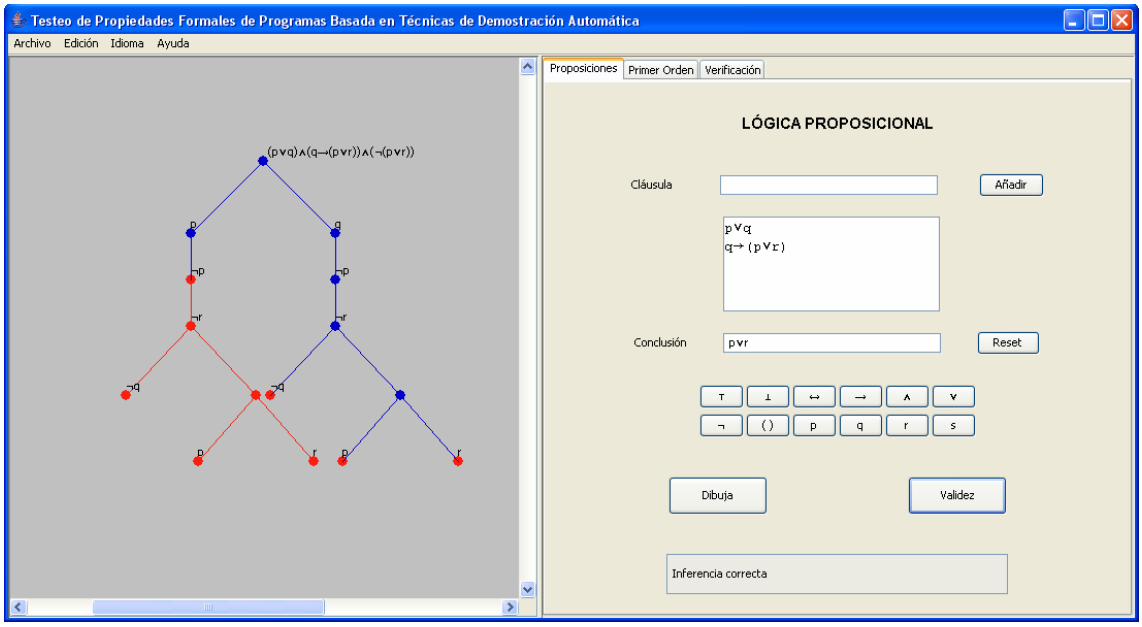

(Figura 4)

Los botones del teclado utilizados para introducir símbolos lógicos en la herramienta son las siguientes:

- ` ; ` .- Carácter utilizado para introducir el símbolo ⊤.
- ` : ` .- Carácter utilizado para introducir el símbolo ⊥.
- ` + `.- Carácter utilizado para introducir el símbolo ∑.
- ` * `.- Carácter utilizado para introducir el símbolo ∏.
- ` ¿ ` .- Carácter utilizado para introducir el símbolo ↔.
- ` ? ` .- Carácter utilizado para introducir el símbolo →.
- ` &' .- Carácter utilizado para introducir el símbolo ∧.
- ` | ` .- Carácter utilizado para introducir el símbolo ∨.
- ` ! ` .- Carácter utilizado para introducir el símbolo ¬.

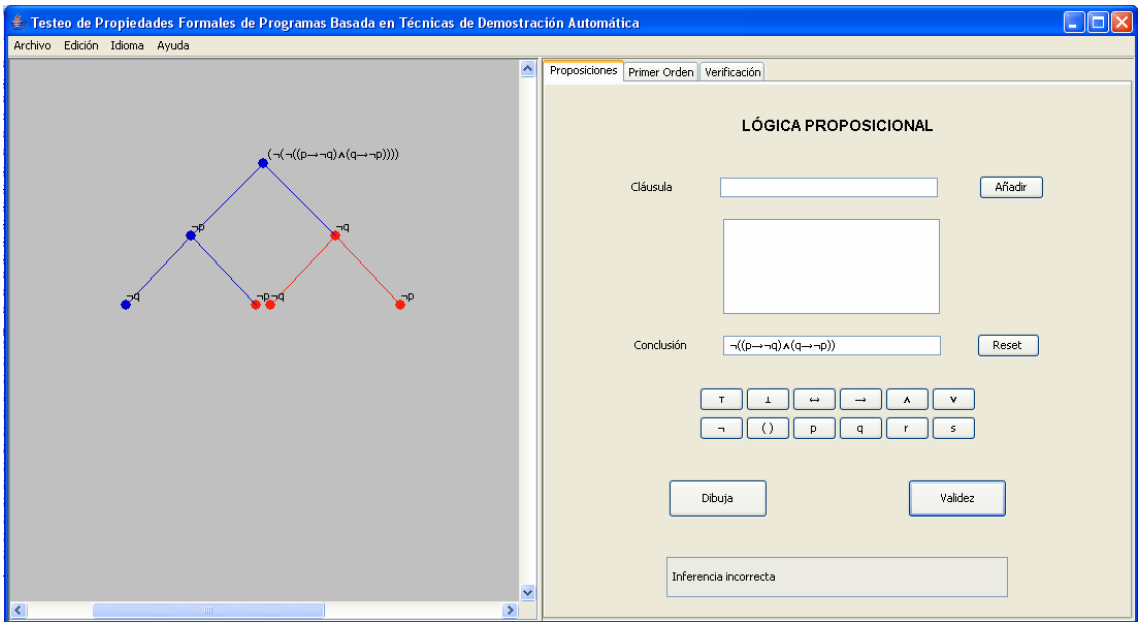
Una vez introducidas las cláusulas y la conclusión, el usuario se dispone a poder dibujar el tableau resultante y a comprobar la veracidad de la argumentación. El botón con la etiqueta "Dibuja" muestra en el panel gráfico de la izquierda el tableu que resulta del conjunto de fórmulas introducidas en el panel derecho (Figura 2). El botón con la etiqueta "Validez", muestra en color azul el recorrido que se ha hecho por cada rama del árbol hasta comprobar si se ha cerrado o no.

Como se observa en el siguiente ejemplo, el recorrido por la rama de la izquierda termina en el momento en que se computa la fórmula atómica "¬p", ya que justo en este punto la rama ha quedado cerrada. En la rama de la derecha, el recorrido llega hasta las hojas del árbol, en ese instante se comprueba que efectivamente todas las ramas han sido cerradas.




(Figura 5)

En este otro ejemplo, se puede observar el comportamiento del recorrido del árbol cuando una de las ramas queda abierta. En este caso, la hoja de la rama izquierda queda coloreada en azul, lo que indica que esa rama ha quedado abierta y por lo tanto, en ese instante, el recorrido del árbol finaliza sin comprobar el estado del reto de ramas.

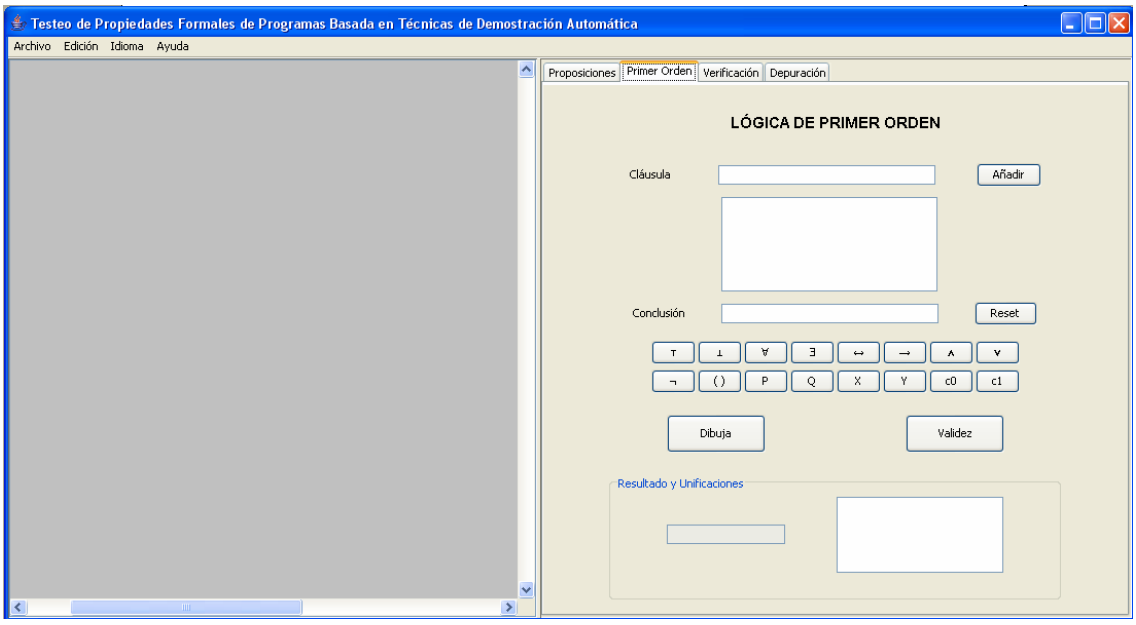



(Figura 6)

El resultado final de la validación se muestra en el campo de texto situado en la parte inferior del panel a través de los mensajes "Inferencia incorrecta" (el tableau queda abierto) o "Inferencia correcta" (el tableau queda cerrado). Este campo de texto ha sido inhabilitado para que el usuario no pueda escribir en el, al igual que se ha hecho con el área de texto antes descrito.

Por último, el usuario tiene la opción de una vez comprobada su argumentación, volver a introducir otra. Por ello, hemos situado en el panel un botón de "reset" que limpia completamente ambos paneles, el izquierdo y el derecho, con todos sus campos.

El segundo panel que conforma el panel de pestañas es el que tiene por etiqueta "Lógica de Primer Orden" (Figura 7). Este panel es similar al que se ha comentado anteriormente, por lo que se comentarán únicamente las novedades introducidas.




(Figura 7)

La mayor diferencia reside en el hecho de que se han introducido cuantificadores universales y existenciales para la construcción de fórmulas. Esto permite al usuario moverse en un ámbito de la lógica de proposiciones mucho más amplio.

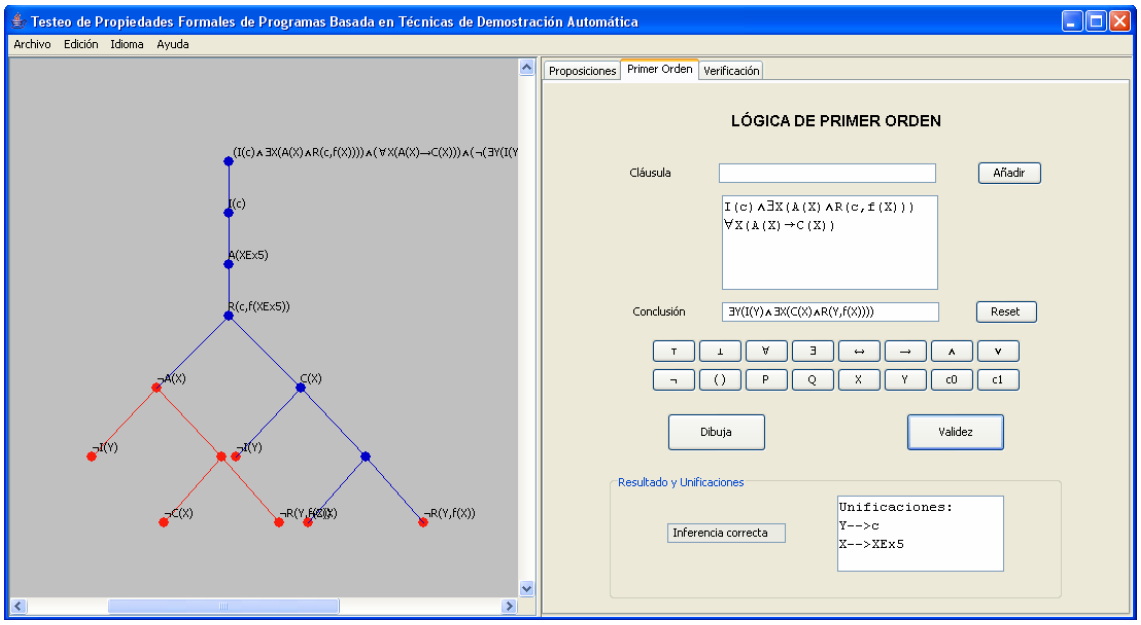
Los nuevos elementos gráficos introducidos en el panel son en primer lugar, los botones que permiten introducir los símbolos lógicos que representan el cuantificador universal, "para todo", y el cuantificador existencial, "existe". Estos botones han sido añadidos a la matriz de botones. Al igual que con el resto de botones, se ha facilitado que el usuario pueda introducir los símbolos directamente desde teclado, las teclas que realizan esta función son:

- ` $ `.- Carácter utilizado para introducir el símbolo ∀.
- ` @'.- Carácter utilizado para introducir el símbolo ∃.

Esta información se muestra a través de un mensaje de ayuda al mover el cursor del ratón sobre los botones destinados a estos símbolos.

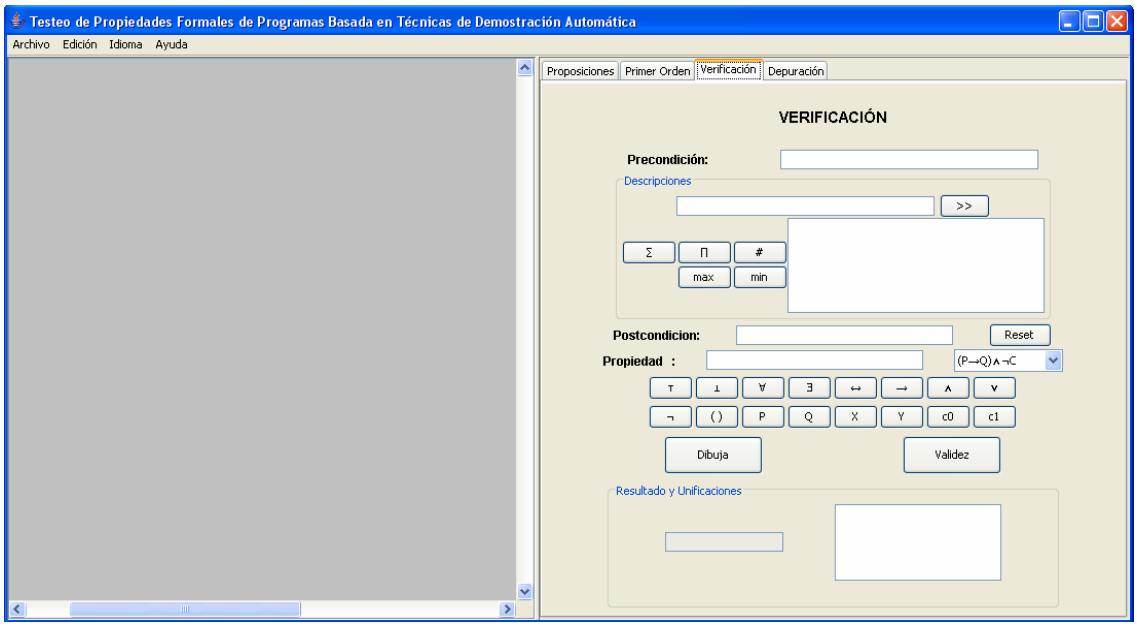
Una vez introducidas las cláusulas de la premisa y la conclusión, el usuario se dispone a comprobar la veracidad de su argumentación. Para ello, y al igual que en el panel anterior, el usuario dispone de una opción que únicamente dibuja el tableau resultante (botón "Dibuja") y otra que lo comprueba (botón "Validez"). El resultado se muestra en el campo de texto inferior mediante los mensajes de "Inferencia Correcta" e "Inferencia incorrecta".

Además, se ha añadido en este panel una nueva área de texto en donde la aplicación muestra al usuario las unificaciones de variables universales que se han podido realizar durante la ejecución del programa. Al igual que en la anterior área de texto, en esta también se han introducido barras de desplazamiento que se habilitarán en el momento que sean necesarias.



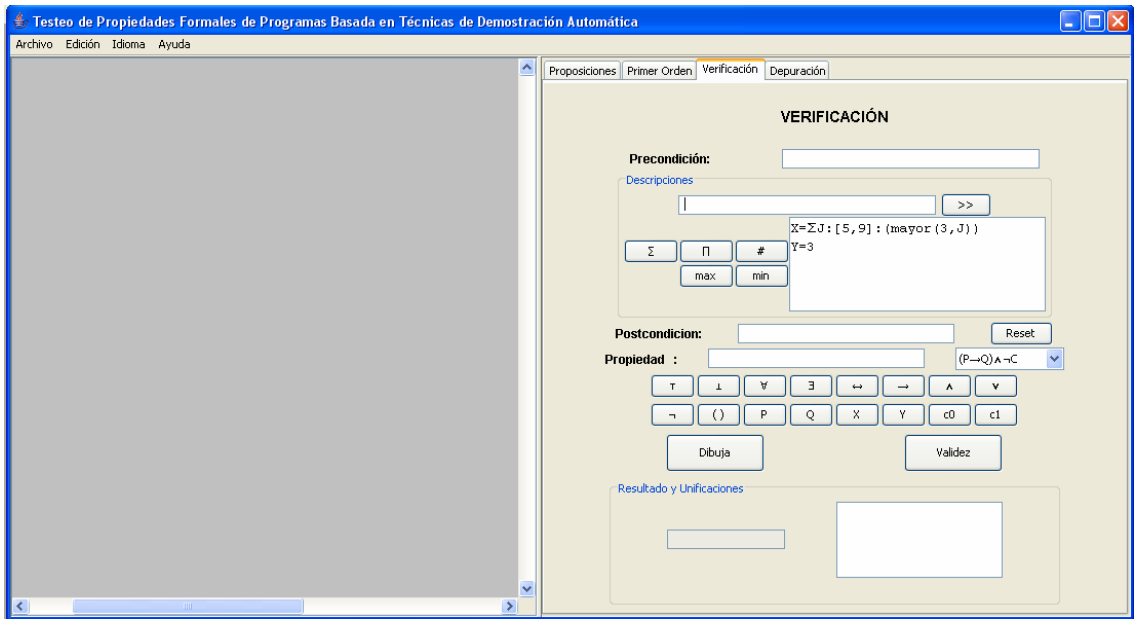

(Figura 8)

El siguiente panel de la herramienta es el panel de Verificación (Figura 9). En él el usuario puede comprobar propiedades de programas simplemente introduciendo una precondición, una poscondición unos valores de variables (descripciones) y verificando que se cumple la conclusión o propiedad introducida.




(Figura 9)

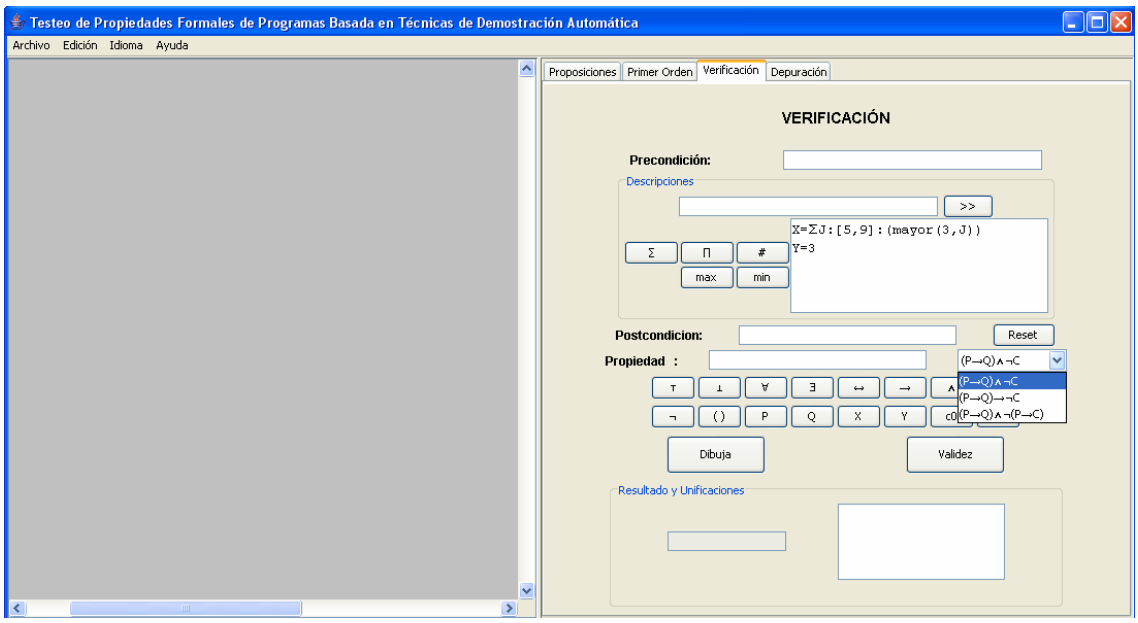
Como se puede observar, el aspecto es similar al de los paneles anteriormente comentados. Las diferencias más representativas son que en este panel se ha introducido un área diferente de descripciones, en donde el usuario puede introducir asignaciones de variables simples o asignación de variables con funciones evaluables (Figura 10), que vienen dadas en una matriz de botones en al mismo espacio que el área.




(Figura 10)

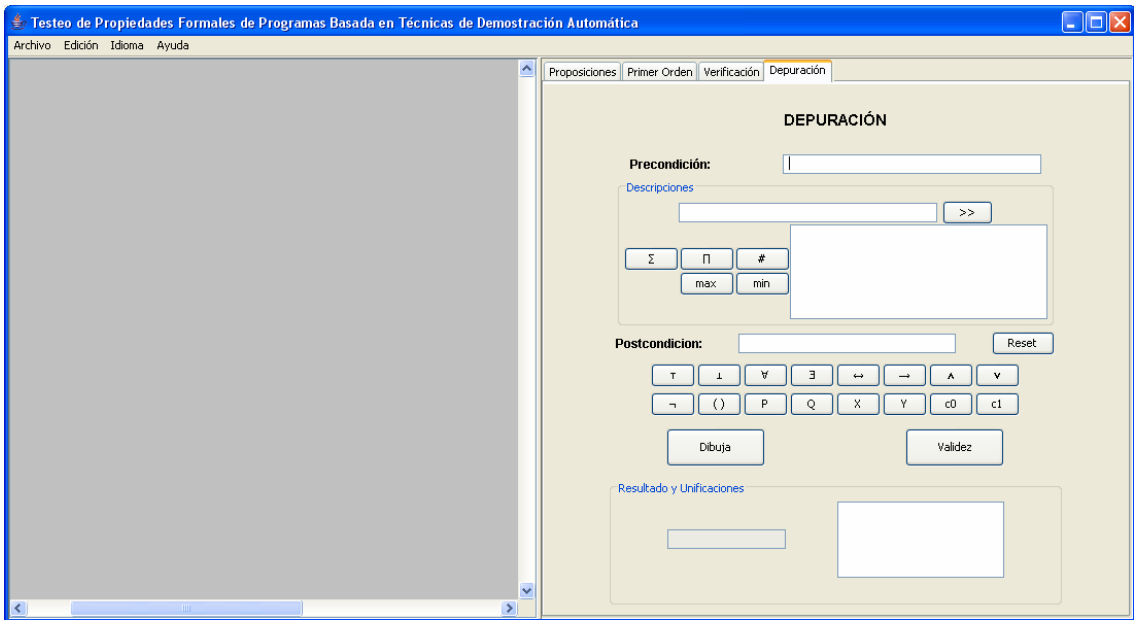
Hay que recordar que las variables a las que queramos asignar algún valor deben ir en mayúsculas que es la manera en la que se ha decidido que se representen las variables.

Una función más introducida en este panel, es la opción de dejar al usuario decidir por como desea realizar la comprobación de su argumentación. Para ello se ha dispuesto de un menú desplegable (Figura 11) al lado del campo propiedad.




(Figura 11)

El último panel de la herramienta es el panel de depuración en el que el usuario puede con sólo una precondición, una postcondición y las descripciones testear la implicación entre diferentes propiedades de un programa (Figura 12). La forma de uso es similar a la de los paneles anteriormente comentados.




(Figura 11)

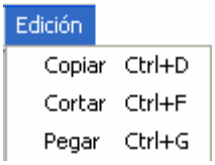
Por último y común a todos los paneles anteriormente comentados, el usuario dispone de una barra de menú con la que puede acceder a las distintas opciones que a continuación se explican.

El primer elemento de esta barra es un menú de archivo. En él simplemente se han añadido las funciones de "Abrir", "Guardar" y "Salir" con las que el usuario puede fácilmente recuperar argumentaciones guardadas anteriormente (Figura 12).


(Figura 12)

Cabe destacar que el formato en el que se guardan las argumentaciones es ".tab".

El siguiente elemento de la barra de menú es el sub-menú "Edición". En él el usuario puede seleccionar las funciones típicas de "Copiar", "Cortar" y "Pegar" sobre los diferentes componentes de texto de los paneles de la herramienta (Figura 13).

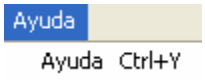



(Figura 13)

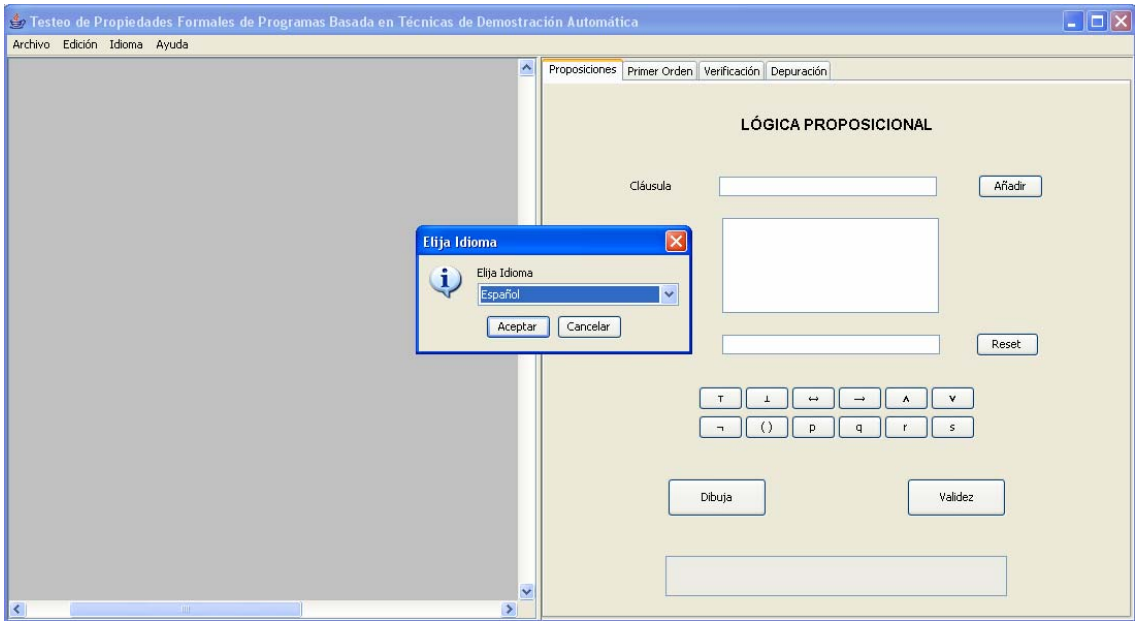
Cabe destacar que todas las fórmulas introducidas en las áreas de texto, han de ser introducidas desde el campo de texto correspondiente, es decir, sobre las áreas de texto el usuario sólo puede copiar, pero en ningún caso puede cortar ni pegar.

Los dos últimos elementos de la barra de menú son el sub-menú "Idioma" (Figura 14) y el sub-menú "Ayuda" (Figura 15). El primero de ellos permite al usuario seleccionar entre los dos idiomas de los que dispone la herramienta, español e inglés. Para ello, el usuario debe seleccionar el idioma deseado (Figura 16) y cerrar la aplicación (Figura 17). Cuando se vuelva a ejecutar la aplicación, los cambios serán mostrados (Figura 18).

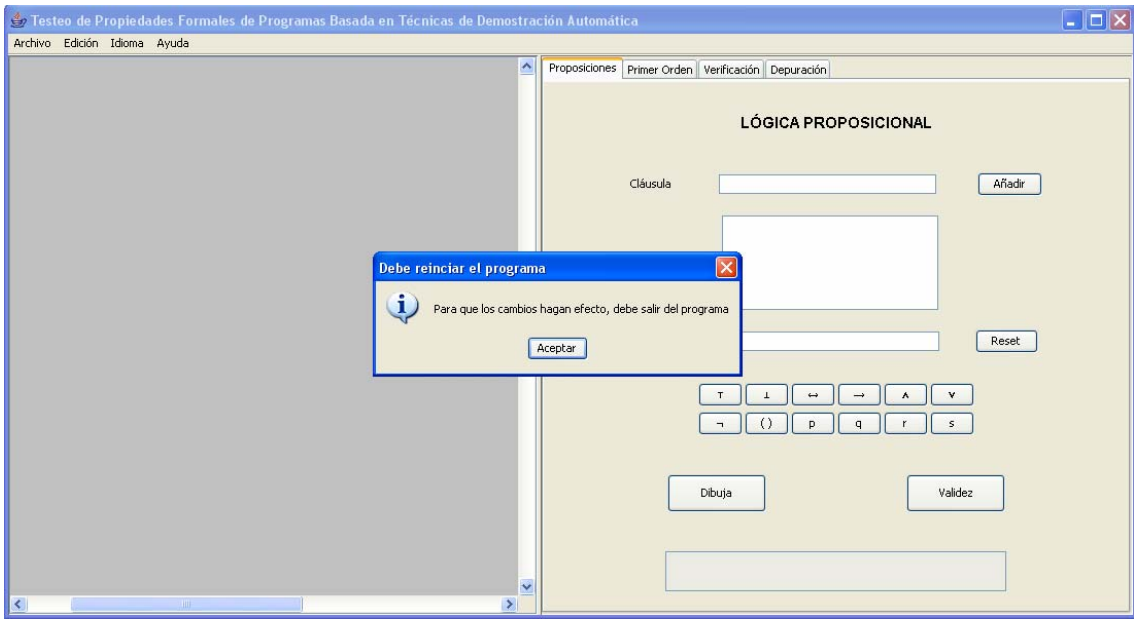

(Figura 14)



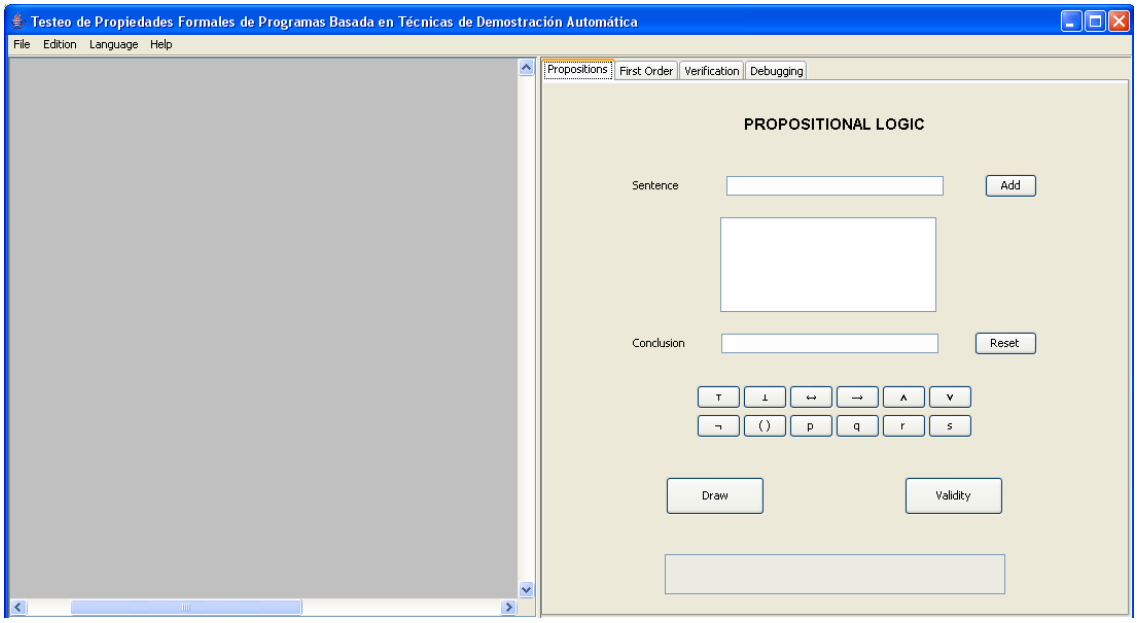

(Figura 15)




(Figura 16)




(Figura 17)




(Figura 18)

El sub-menú ayuda abre un documento en formato .html donde se recoge toda la información aquí explicada.

## 3.3 - Tableaux: Implementación y Parte Gráfica

Los tableaux son las herramientas que se utilizarán para comprobar la veracidad o falsedad de las fórmulas.

Un tableau es un árbol binario en el que los nodos son fórmulas atómicas. Además, nuestra estructura cuenta con un contador de ramas del árbol y las hojas se almacenan también en un array.

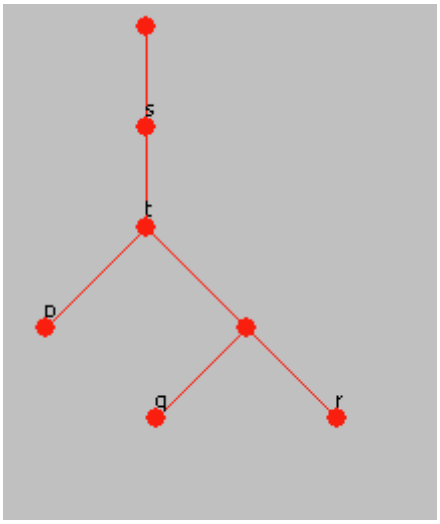
En cada uno de los distintos módulos de nuestra aplicación, se crea una fórmula a partir de las cadenas que introduce por pantalla el usuario (siempre que estas sean correctas). Dado que cada tipo de fórmula genera un tipo de tableau diferente, las propias fórmulas son las encargadas de generar su tableau. Esto se realiza del siguiente modo:

Tanto las fórmulas alfa (conjunciones) como las fórmulas beta (disyunciones) generan de forma recursiva dos tableaux diferentes: uno para la parte izquierda de la fórmula, y otro para la parte derecha. La diferencia reside en la forma en que estos tableaux se combinan en cada caso.

La idea intuitiva de las fórmulas alfa es situar en la misma rama ambas partes de la conjunción, de forma que en cuanto una de las partes sea "falsa" esa rama pueda ser descartada a la hora de intentar cerrar el árbol. Para ello, lo primero que se hace es mirar cuál de los dos tableaux tiene menor número de ramas, para situar el tableau más pequeño más cercano a la raíz y así no aumentar innecesariamente el tamaño del tableau resultante. Una vez que se sabe cuál de los dos tableaux es menor, añadimos el tableau mayor a todas y cada una de las ramas del tableau menor.

En el caso de las fórmulas beta la forma en que se combinan los tableaux es diferente, ya que lo que se pretende es representar una disyunción. Cada vez que se presenta una disyunción se crea una nueva rama en el árbol. Así pues, en este caso el resultado de la combinación es un nuevo tableau en el que el hijo izquierdo es la parte izquierda de la disyunción y el hijo derecho la parte derecha, mientras que la raíz queda vacía (es simplemente un punto de bifurcación).

Por ejemplo, para crear el tableau para la fórmula "(pv qv r) ∧ (s∧ t), se genera el siguiente árbol:




Una vez que se ha creado el tableau, se precisa de un método para comprobar si éste se cierra o no. Este método será invocado cuando el usuario quiera demostrar la veracidad o falsedad de una argumentación, y es el más interesante y complejo de esta clase, por lo que a continuación se trata de explicar brevemente.

Simplificando, se puede entender que una rama se puede cerrar por tres motivos:

1) Que en ella se encuentre una fórmula y su negación (por ejemplo "p" y "¬p").

2) Que en ella se encuentre un predicado con parámetros y otro predicado negado cuyos parámetros unifican con los parámetros del predicado sin negar (por ejemplo "P(X,Y)" y "¬P(c0,c1)".

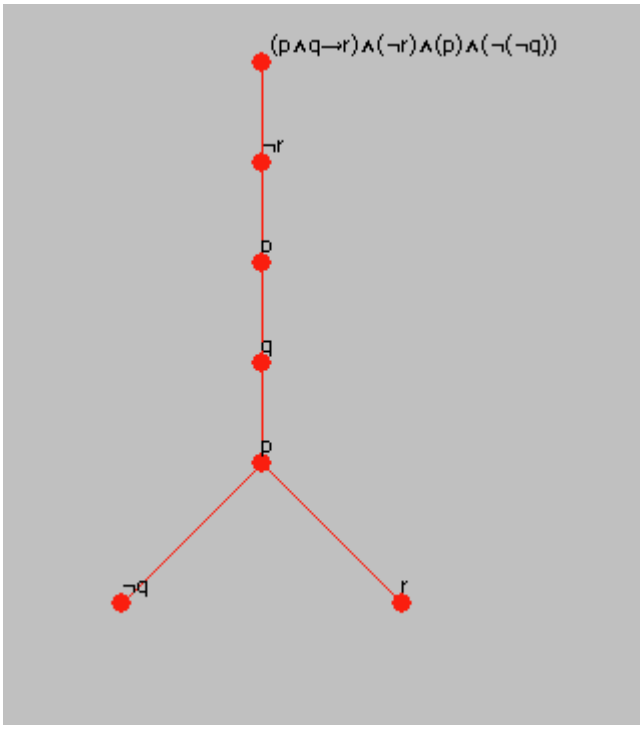
3) Que se encuentre alguna contradicción en el caso de que el predicado sea evaluable (por ejemplo "mayor(4,6)" o "iguales(X,5) ∧ menor(X,3)").

Por lo tanto, el método retorna "cierto" (la rama se cierra) si en la rama se detecta alguna de las condiciones anteriores. En caso contrario, llama recursivamente al mismo método para sus hijos izquierdo y derecho si los tuviera. Si se ha llegado al final de la rama sin que esta se haya cerrado devuelve "falso". Recordemos por último, que al ser recursivo, el método tan sólo devuelve "cierto" cuando todas las ramas del árbol han sido cerradas.

La aplicación no diferencia entre predicados proposicionales y predicados de primer orden, sino que trata a los primeros como un subconjunto de los segundos (sin argumentos), de modo que el método es válido para ambos casos.

En cuanto a la parte gráfica, representamos los tableaux con una estructura clásica de árbol. El nodo "raíz" se sitúa en lo alto y el árbol crece hacia abajo a partir de él.

Como ya se ha explicado, los nodos del árbol son fórmulas, y tienen un hijo izquierdo y probablemente un hijo derecho (en el caso de ser una disyunción). De esta forma es como los dibujamos. En los nodos escribimos la fórmula, mientras que los hijos los dibujamos con unas líneas que representan las ramas del árbol.



Las líneas verticales implican conjunciones, mientras que las oblicuas equivalen a disyunciones. Como ya se comentó, a fin de disminuir el tamaño del árbol (su número de ramas), tratamos de retrasar lo máximo posible las disyunciones para que no haya que repetir información en todas las ramas.

En el caso de la lógica de primer orden, el procedimiento es análogo. La única diferencia es que en los nodos, las fórmulas aparecerán con parámetros en el caso de que los predicados introducidos los contuviesen.

La limitación que hemos tenido en este caso ha sido un problema de espacio en el caso de generar árboles con gran cantidad de fórmula, ya que las ramas se solapan y resulta complicada la visión del tableau.

## 3.4 - Posibilidades de extensión de la herramienta

Dado que se trata de una herramienta para testeo y verificación de propiedades de programas, las posibilidades de extensión más interesantes irían encaminadas a facilitar la transformación entre las construcciones de lenguajes de programación y el lenguaje de la lógica utilizado en la parte de verificación de la herramienta.

De esta manera, sería posible implementar un compilador que convirtiera, automáticamente, instrucciones imperativas sencillas en fórmulas lógicas manejables por la herramienta. A continuación, a modo de ejemplo, se muestra la transformación de una intrucción IF-THEN-ELSE

if B(X,Y) (B(X,Y) & iguales(Z,3)) | (¬B(X,Y) & iguales(Z,5))
then Z := 3 →
else Z := 5

Para darle a esta ampliación una verdadera utilidad práctica, sería necesario introducir nuevos predicados y funciones evaluables, que permitan aumentar el conjunto de programas testeables por la aplicación. Esta parte sería sencilla gracias a la implementación del programa, ya que bastaría introducir nuevas clases que implementaran los interfaces FuncionEvaluable o PredicadoEvaluable. Además, sería aconsejable introducir tipos en las construcciones, con el fin de aumentar las posibilidades de funcionamiento.

Por otra parte, sería factible desarrollar una transformación "en la otra dirección". Es decir, aprovechar el manejo de las fórmulas de la lógica para derivar automáticamente programas sencillos, con la seguridad de que serían correctos por la propia verificación realizada por la herramienta. Como ejemplo serviría la anterior tranformación mostrada, con la flecha en el sentido contrario.

Otras posibilidades de extensión son las referentes a otras clases de lógica, modificando convenientemente las estructuras de las fórmulas, y las condiciones según las cuales se cierran los tableaux

# RECOPILACIÓN DE EJEMPLOS

## 4.1 - Ejemplos de Tableaux en Lógica Proposicional

A continuación se presentan algunos ejemplos del funcionamiento de la herramienta para la lógica de proposiciones.

El usuario tan solo tiene que introducir cláusulas, hacer clic en el botón añadir cada vez que haya terminado la escritura de una cláusula y por último introducir la conclusión y pulsar "Validez". Automáticamente la aplicación generará el árbol resultante de la conjunción de todas las cláusulas y la negación de la conclusión y tratará de cerrarlo para decidir si la inferencia es o no correcta.

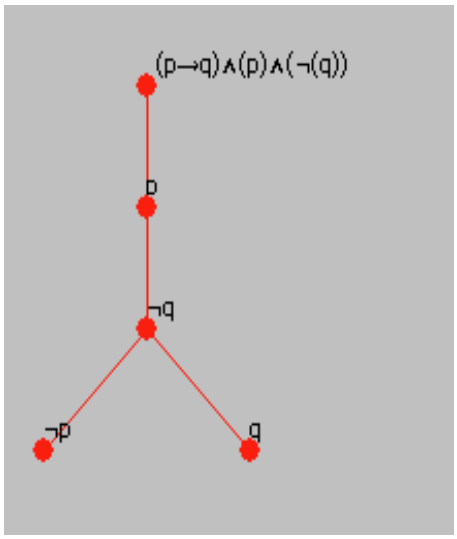
Ejemplo 1:

Sea la argumentación:

-Si llueve hace frío.
-Llueve

Conclusión:
-Hace frío

Cláusulas: p->q, p
Conclusión: q
Inferencia correcta





Se puede observar cómo al negar la conclusión todas las ramas del árbol se cierran, con lo que la fórmula resultante es insatisfactible y por tanto la inferencia es correcta.

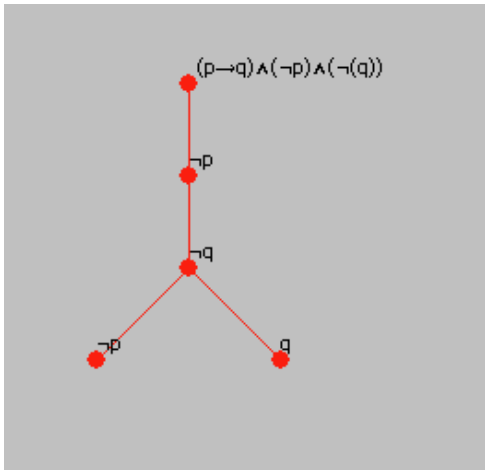
Ejemplo 2:

Sea ahora la argumentación:

-Si llueve hace frío.
-No llueve.

Conclusión:
-Hace frío.

Cláusulas: p->q, ¬p
Conclusión: q
No se puede inferir la conclusión.





En este caso vemos como la rama de la izquierda queda abierta, con lo que la fórmula es satisfactible, y por lo tanto no se puede garantizar la inferencia.

## 4.2 - Ejemplos de Tableaux en Lógica de Primer Orden

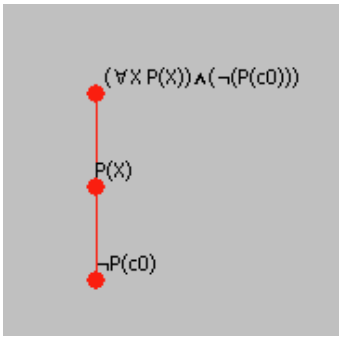
Veamos ahora algunos ejemplos de lógica de primer orden.

Nuevamente, el funcionamiento de la herramienta es análogo al del caso proposicional. Las únicas diferencias que se incorporan son la posibilidad de añadir cuantificadores y un recuadro para comprobar las unificaciones que han tenido lugar durante el proceso de intentar cerrar el árbol.

Ejemplo 1:

Tenemos la argumentación.

-Todo el mundo es una persona.




Conclusión:

-c0 es una persona.

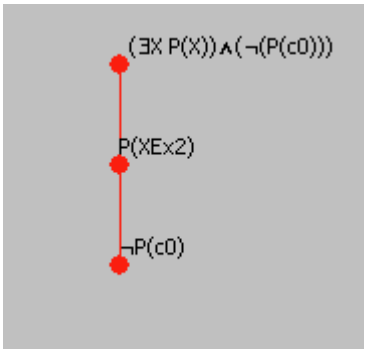
Cláusula: $\forall X(P(X))$
Conclusión: P(c0)
Inferencia correcta.


En este caso, al unificar la constante "c0" con la variable universal X el árbol se cierra y la inferencia es correcta.

Ejemplo 2:

Argumentación:
-Existe alguien que es persona.

Conclusión:
-c0 es una persona.




Cláusula: ∃ X P(X)
Conclusión: P(c0)
Inferencia incorrecta.

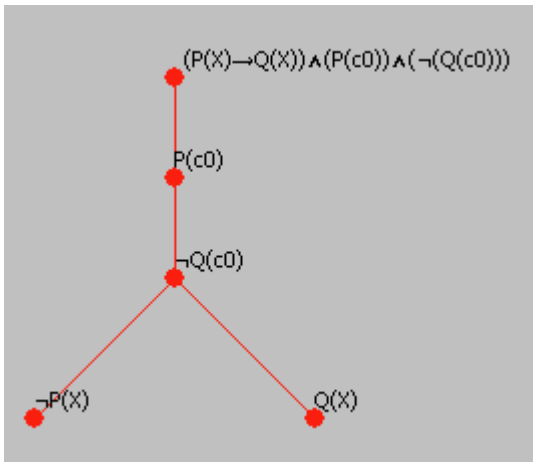

En este caso, al ser la "X" una variable existencial la unificación no puede llevarse a cabo.

Ejemplo 3:

Argumentación:
-Los ratones comen queso.
-Pepe es un ratón.

Conclusión:
-Pepe come queso.

Cláusulas: P(X)→Q(X)
P(c0)
Conclusión: Q(c0)
Inferencia correcta.





De nuevo, al hacer la unificación de X con c0, se cierra el árbol.

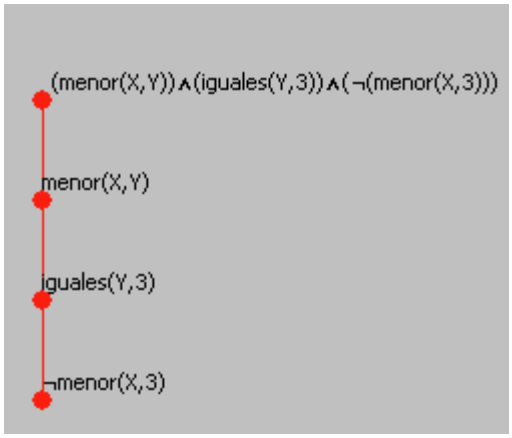
Veamos un último ejemplo con predicados evaluables.

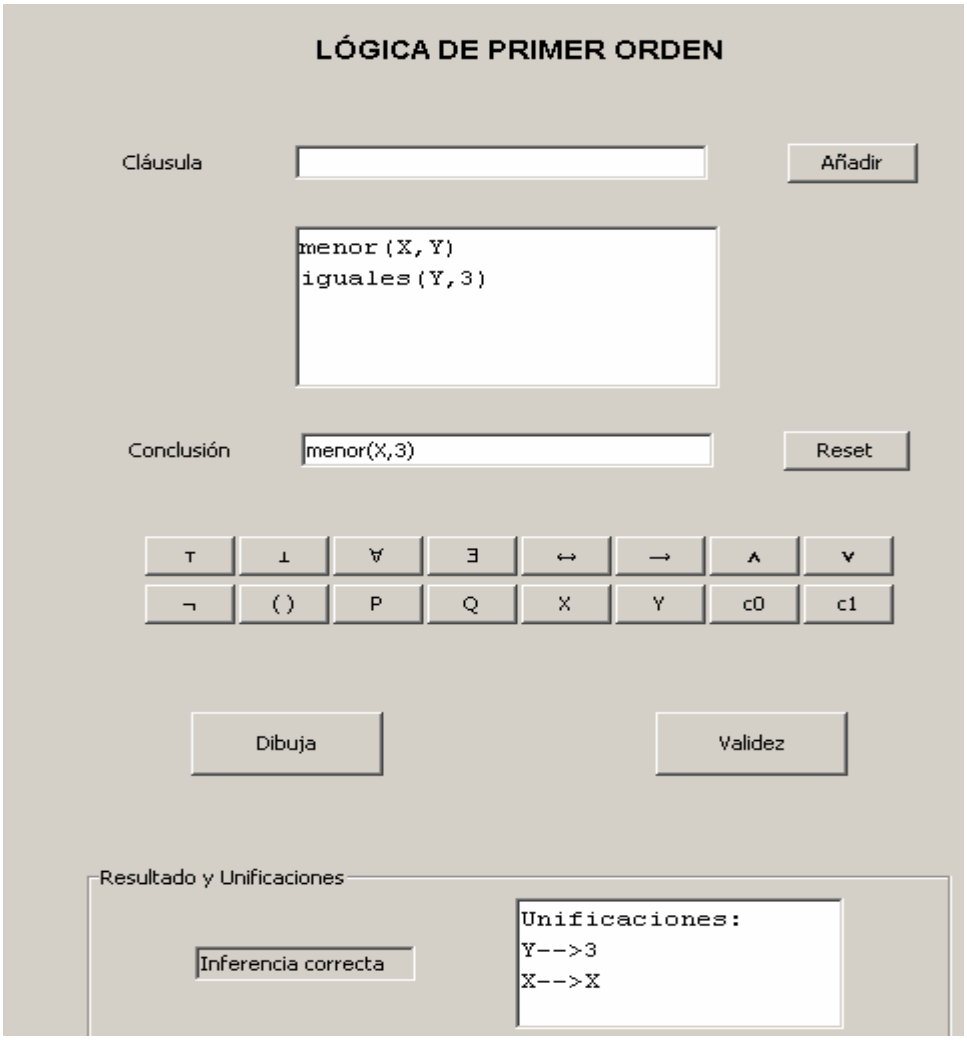
Ejemplo 4.

Argumentación:
-X es menor que Y
-Y es igual a 3.

Conclusión:
-X es menor que 3.

Cláusulas: menor(X,Y)
iguales(Y,3)
Conclusión: menor(X,3)
Inferencia correcta.



## 4.3 – Ejemplos de testeo de propiedades de programas

Por último se muestran algunos ejemplos de propiedades sencillas de programas que pueden ser comprobadas con nuestra herramienta.

En este caso, el usuario puede introducir una precondición, una postcondición y unas descripciones de variables (para que las variables tomen un valor concreto) y comprobar si de ellas se desprende una determinada propiedad o conclusión. El tipo de implicación que desea testear se puede seleccionar con una lista de elección.

Nuevamente con el botón "Validez" el programa muestra el árbol creado y comprueba si la propiedad se cumple o no.

Ejemplo 1:

Pre:X>0 AND Y>0
Post: X==M AND Y==M

Propiedad: M>0

Precondición: mayor(X,0) ∧ mayor(Y,0).

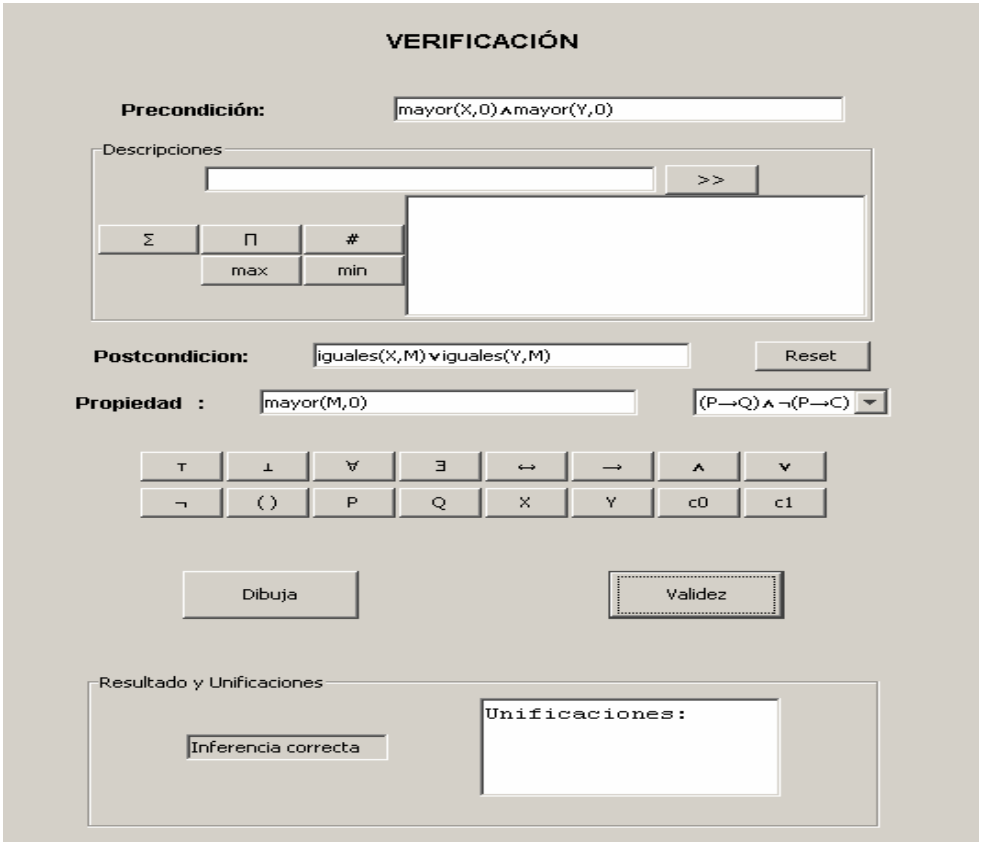
Postcondición: iguales(X,M) ∨ iguales(Y,M).

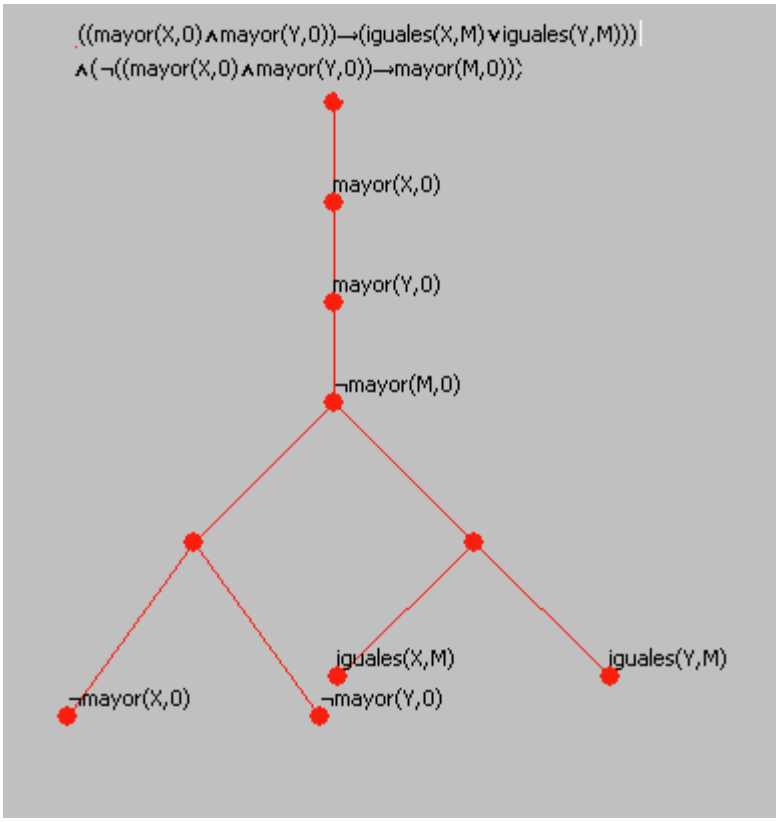
Propiedad: mayor(M,0).

Tipo de implicación: (P->Q) ∧ ¬(P->C)

Resultado: Inferencia correcta.

En este caso no se han utilizado descripciones para fijar el valor de las variables.




((mayor(X,0)∧mayor(Y,0))→(iguales(X,M)∨iguales(Y,M)))
∧(¬((mayor(X,0)∧mayor(Y,0))→mayor(M,0)))
mayor(X,0)
mayor(Y,0)
¬mayor(M,0)
iguales(X,M)
iguales(Y,M)
¬mayor(X,0)
¬mayor(Y,0)

Ejemplo 2:

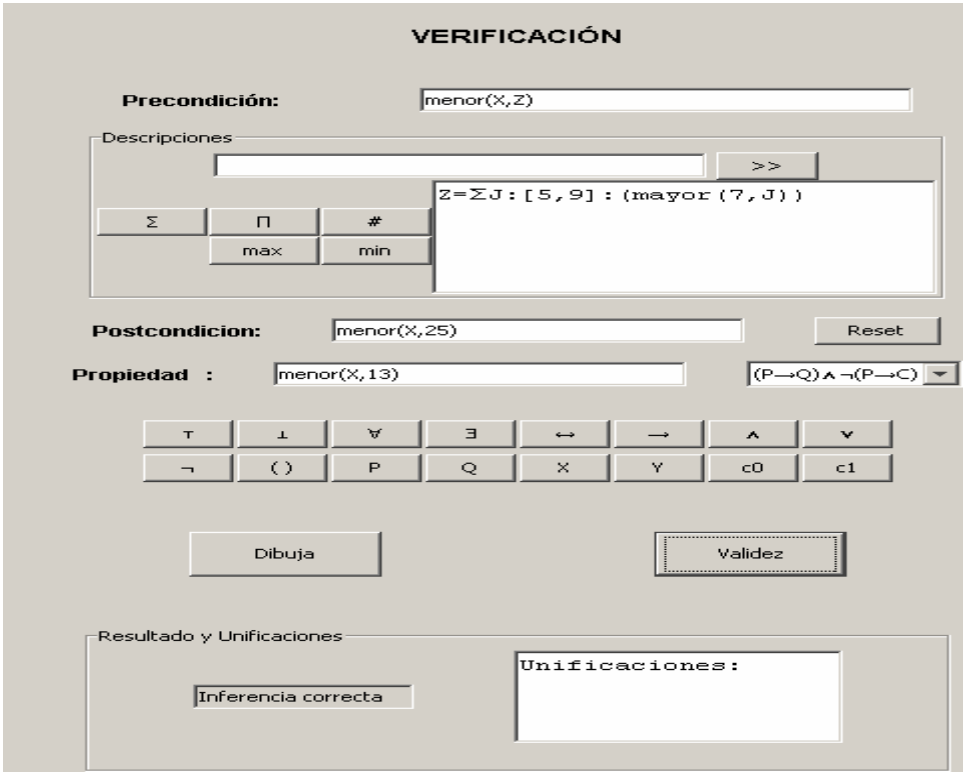
Pre :X<Z
Post :X<25

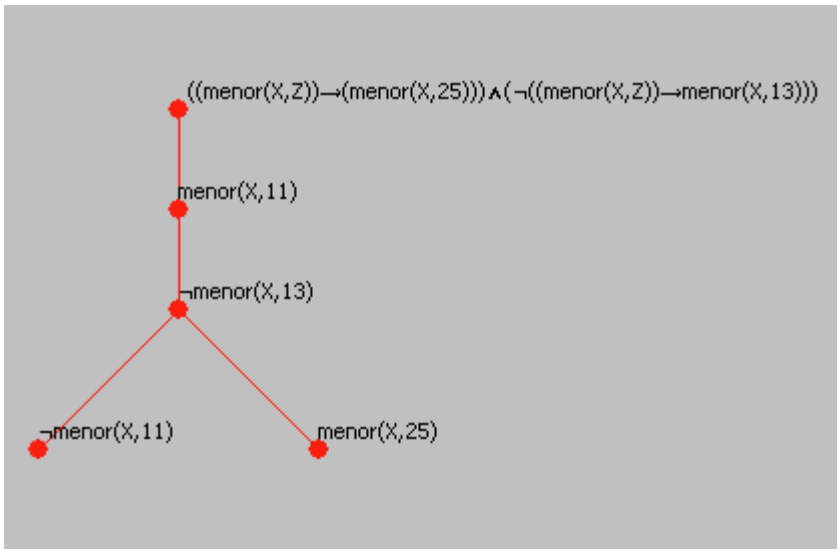
Propiedad :X<13

Preondición: menor(X,Z)
Postcondición: menor(X,25)
Propiedad: menor(X,13)
Descripciones: Z=∑J:[5,9]☹mayor(7,J))

Resultado: inferencia correcta.

El valor de Z es 11 y así aparecerá en la construcción del tableau.



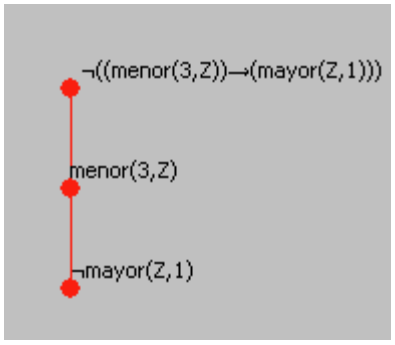

((menor(X,Z))→(menor(X,25)))∧(¬((menor(X,Z))→menor(X,13)))
menor(X,11)
¬menor(X,13)
¬menor(X,11)
menor(X,25)

Por último vamos a mostrar un par de ejemplos que se pueden realizar sencillamente en la ventana de "Depuración" donde se testean cómodamente implicaciones entre propiedades de programas.

Ejemplo 3:

Precondición: menor(3,Z)
Postcondición: mayor(Z,1)

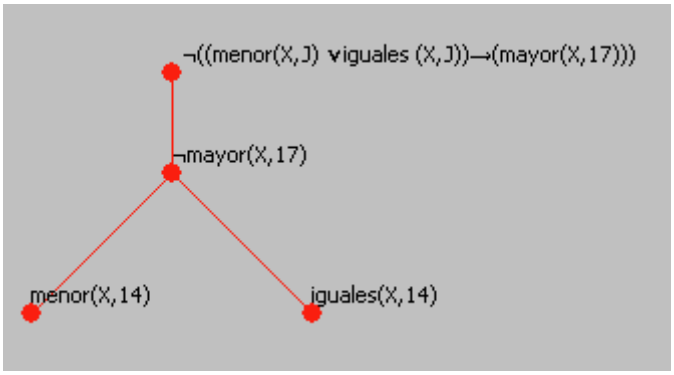
Resultado: inferencia correcta.




Ejemplo 4:

Precondición: menor(X,J) v  iguales (X,J)
Descripciones: J=∑K:[2,8]☹menor(K,6))
Postcondición: mayor(X,17)

Resultado: Inferencia incorrecta.

# MANUAL DE USUARIO

## 5.1 – Ejecución de la Aplicación

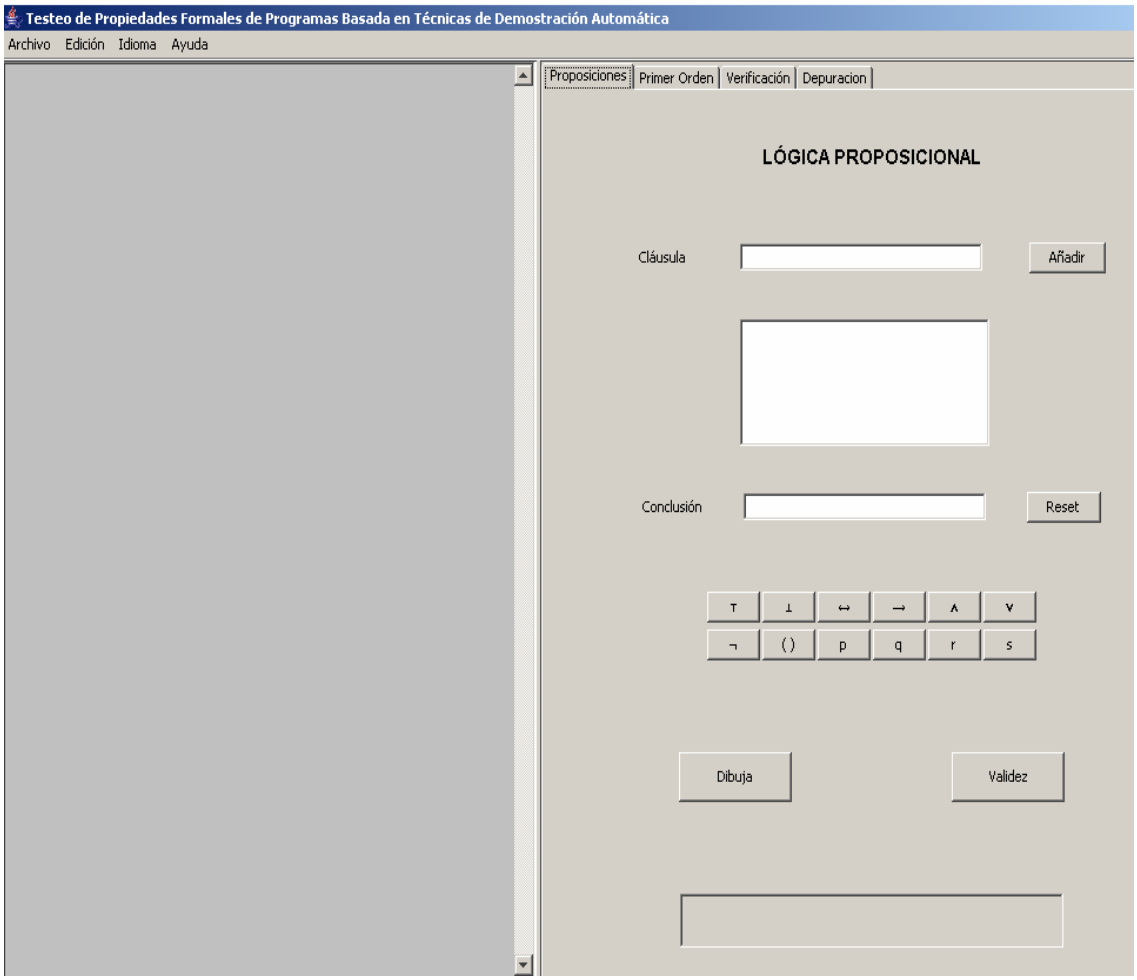
La herramienta dispone de un instalador .exe para Windows. El usuario debe en primer lugar instalar la herramienta en su máquina siguiendo los pasos habituales de un programa de instalación.

Una vez instalado el programa, es requisito fundamental que el usuario tenga instalada en su ordenador la Máquina Virtual de Java, ya que lo que ejecutará será una aplicación .jar. Esto permite a cualquier usuario ejecutar la herramienta bajo cualquier sistema operativo, sólo es necesario tener la Máquina Virtual de Java instalada.

Para ejecutar el programa simplemente se tiene que hacer doble clic sobre el archivo .jar situado en el directorio de instalación seleccionado

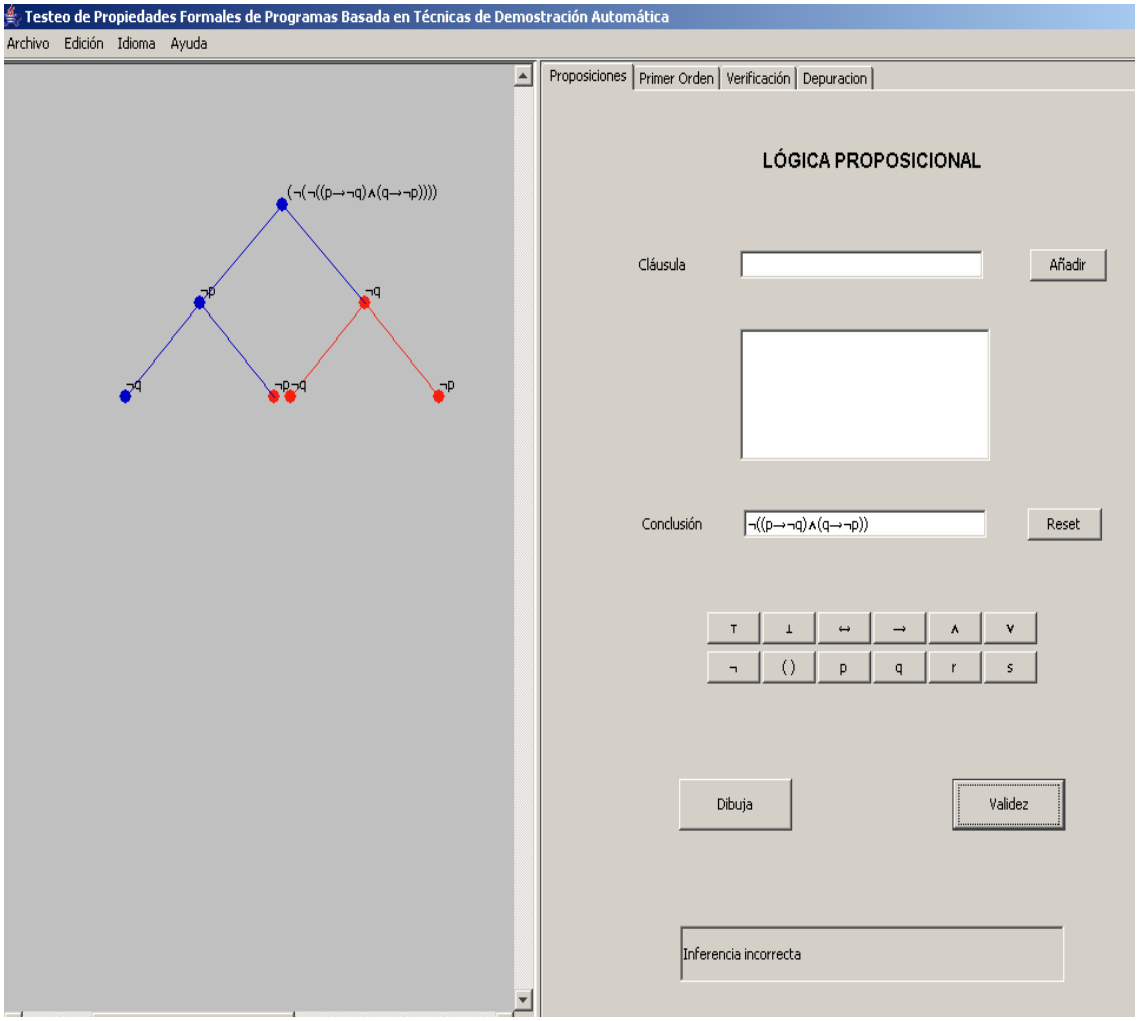
## 5.2 – Tableaux proposicionales

Para trabajar con los tableaux proposicionales, el usuario tan sólo debe ejecutar la aplicación y situarse en la pestaña "Proposiciones", que es la que aparece por defecto al comenzar la ejecución. Si cambia de vista para utilizar otras posibilidades de la herramienta, basta con volver a hacer clic en la pestaña para restaurar la vista de los tableaux proposicionales.



Una vez situados sobre esta pestaña, el usuario introducirá las premisas en el campo de texto etiquetado como "cláusulas". Cada vez que haya terminado de introducir una de estas cláusulas, debe pulsar en el botón "Añadir" para que la cláusula sea validada. En este momento la cláusula aparece en un área de texto no editable. Cuando haya introducido las cláusulas que deseaba, el último paso el añadir la conclusión en el campo destinado para ello. A continuación se puede hacer uso de los botones "Dibuja" para ver el árbol generado o "Validez" para comprobar la veracidad de su argumentación.

En el caso de que la argumentación no sea válida, la primera rama que quede abierta en el dibujo aparecerá coloreada en azul, de modo que el usuario pueda obtener información sobre por qué no es correcta su deducción.



El botón “Reset” limpia todos los campos de texto y sitúa a la herramienta en el estado adecuado para recibir una nueva argumentación.

Para una mayor comodidad del usuario, existen unos botones que permiten escribir cualquier fórmula sin necesidad del uso del teclado. Basta con pinchar sobre ellos y en el campo de texto aparecerá el símbolo requerido. También existen teclas especiales para escribir los símbolos de las conectivas. Situando el ratón sobre el botón que representa este símbolo aparece un mensaje indicando la tecla asignada a cada operador.

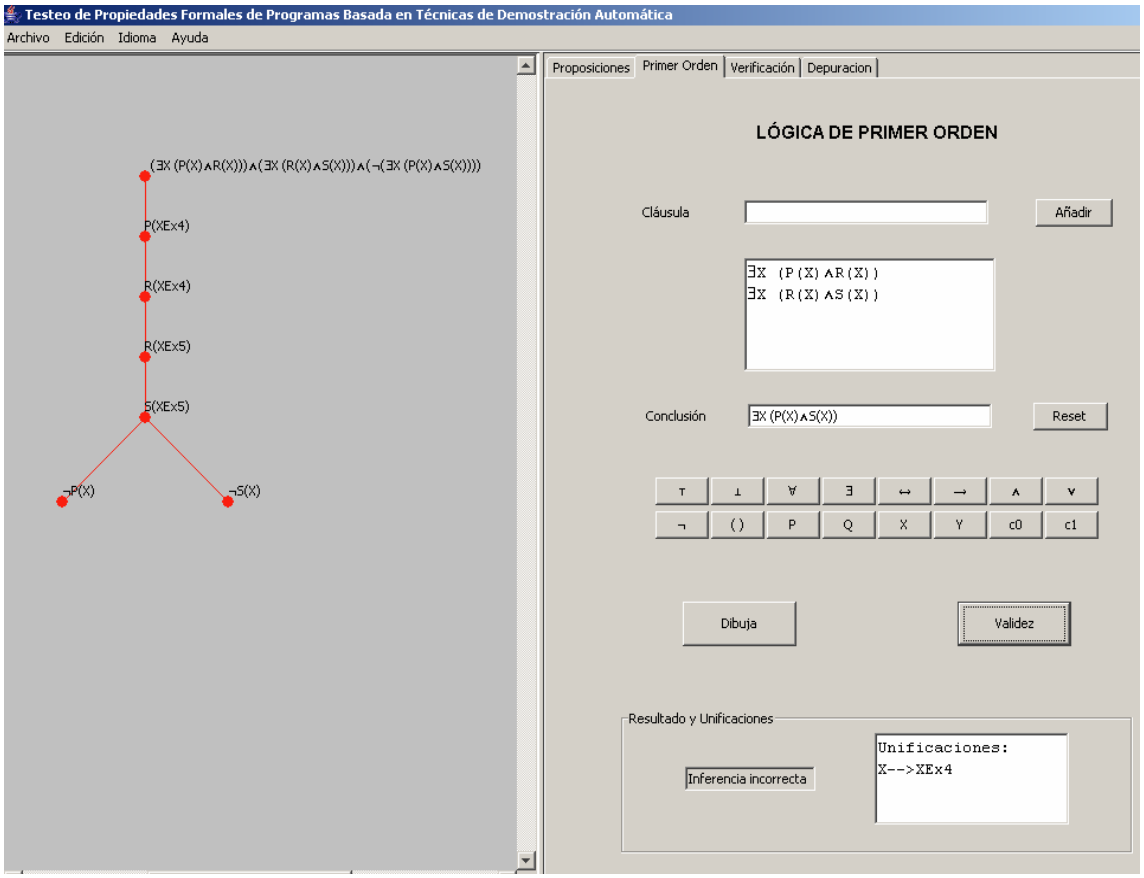
También se ofrece la posibilidad de salvar y cargar por archivo una argumentación.
Para salvar una argumentación (la que se acaba de introducir), basta con ir al menú “Archivo -> Guardar” y salvar normalmente el fichero con la extensión “.tab”. Para cargar un archivo previamente salvado, ir a “Archivo -> Cargar” y seleccionar el archivo deseado. Con ello, la argumentación aparecerá en el panel y quedará lista para ser testeada.

## 5.3 – Tableaux de primer orden

El uso de esta parte de la herramienta es análogo al explicado para lógica proposicional, ya que nuevamente hemos utilizado la propiedad de que la lógica de primer orden no es sino una extensión de la proposicional.

Así, los únicos cambios con respecto al punto anterior son la ampliación de botones para la escritura de las fórmulas debido a los cuantificadores y las variables y constantes y la información sobre las unificaciones.

Ahora en lugar de utilizar colores para distinguir las ramas abiertas y cerradas, lo que se le muestra al usuario es el conjunto de unificaciones que se han llevado a cabo en el intento de cerrar el tableau.



Por último, hay que tener claros algunos convenios de notación:
Cuando en una fórmula aparece una variable sin cuantificar, esto equivale a que se ha cuantificado universalmente. Para que se considere existencial, ha de ser expresado de forma explícita por medio del cuantificador existencial.

Las constantes que son parámetros de un predicado se expresan con minúscula y las variables con mayúscula.

Cuando en el árbol aparece una variable existencial, su nombre comienza por "Ex*Variables*", a fin de distinguirla de las universales.

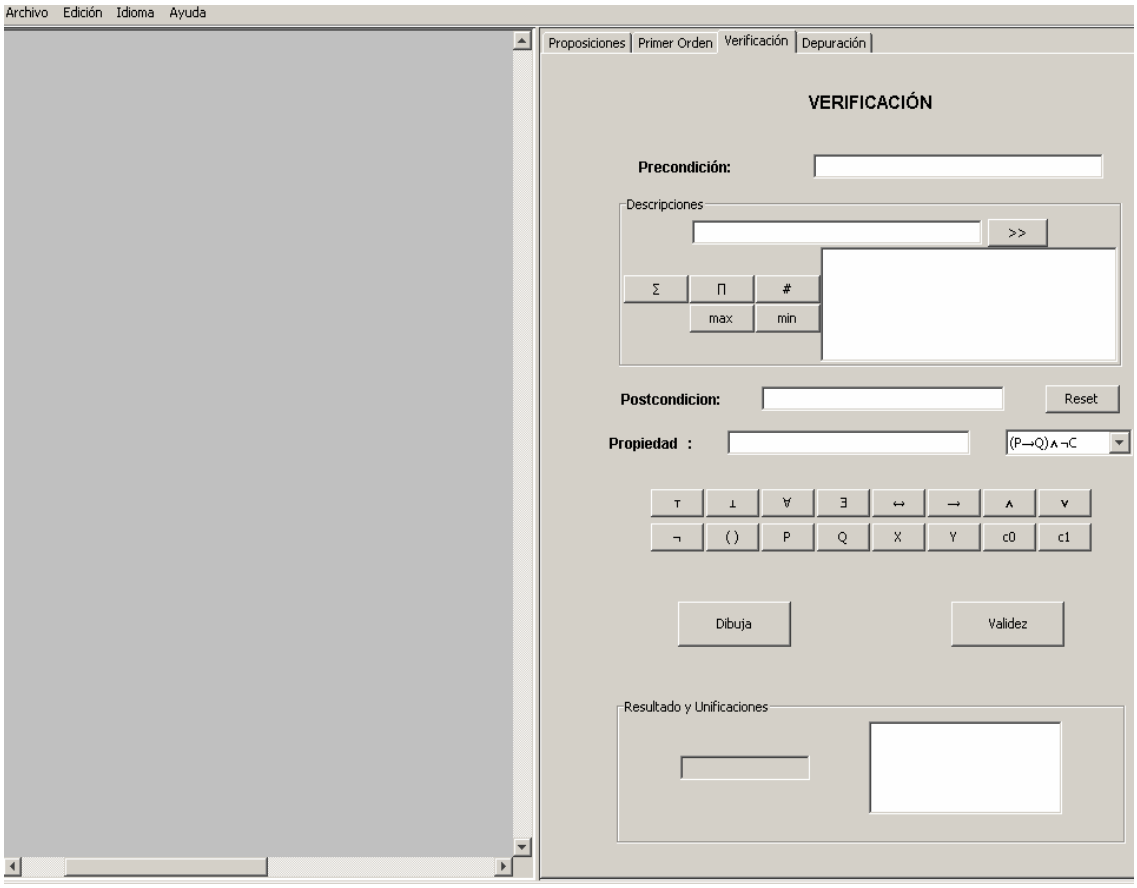
## 5.4 – Deducción de propiedades a partir de su especificación

Esta sección permite al usuario verificar propiedades de programas de un modo similar a como se comprobaban las argumentaciones en los casos anteriores.

El usuario tan sólo debe introducir la precondición, la postcondición y la propiedad que desea deducir de ellas y pulsar el botón "Validez" para que la herramienta haga las comprobaciones. Además, tiene la posibilidad de elegir entre distintos tipos de relaciones entre precondición, postcondición y propiedad.

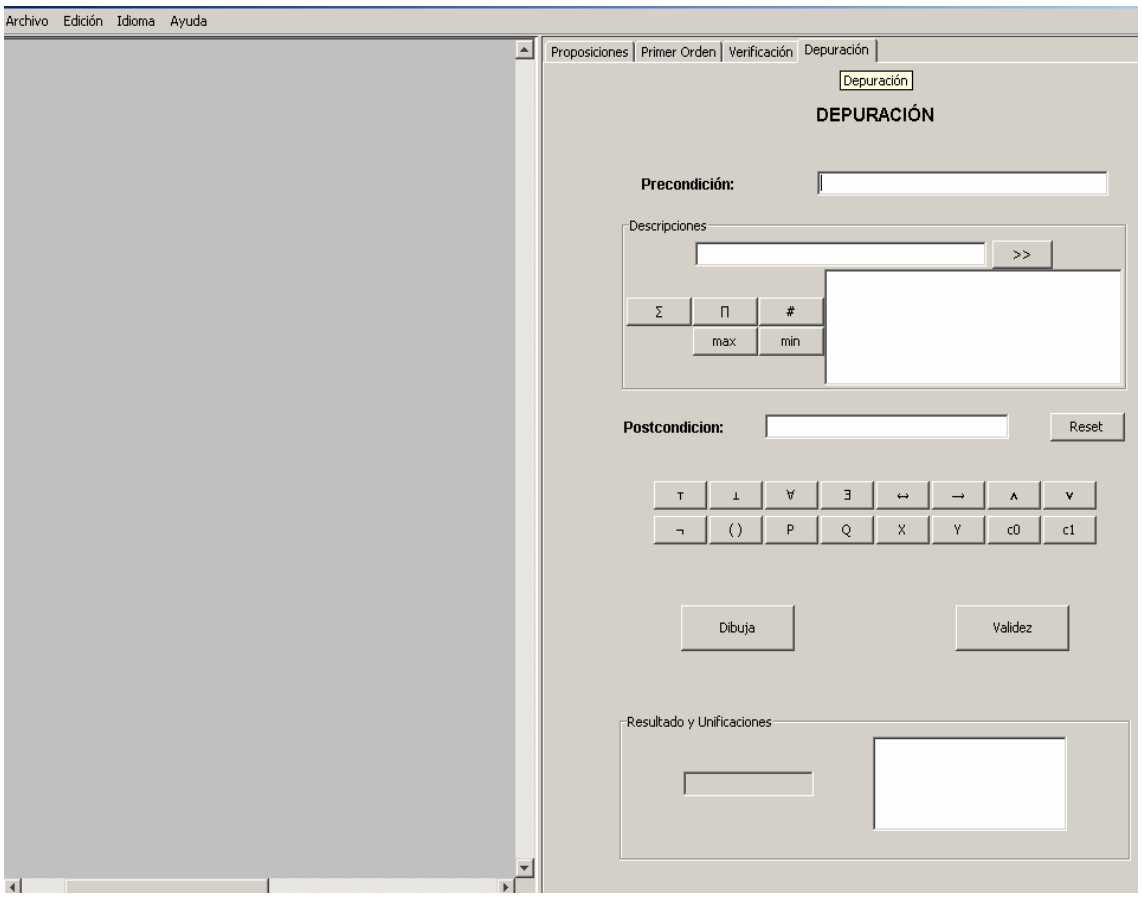
Por último, es posible asignar valores concretos a las variables en el apartado de descripciones. Para ello, se pueden utilizar tanto asignaciones directas del tipo "X:=3" como expresiones utilizando los cuantificadores sumatorio, productorio o conteo o las funciones max o min. Por ejemplo: Z=∑J:[5,9]☹mayor(7,J)).

La vista general del panel es:



Como en los paneles anteriores, se ofrece la posibilidad de salvar y cargar las propiedades a fin de que el usuario no tenga que rescribirlas constantemente.

Por último, la pestaña "Depuración" permite la comprobación de propiedades sencillas mediante una simple implicación. La idea de este panel es realizar comprobaciones de propiedades problemáticas surgidas en el panel "Depuración".




La fórmula final que se comprueba es siempre "precondición->postcondición".

## 5.5 – Ayuda de la Aplicación

Durante cualquier momento de la ejecución de la aplicación, el usuario tiene la posibilidad de consultar la ayuda de la herramienta. Para ello, basta con pulsar el menú "Ayuda" que le guiará en caso de alguna duda sobre el funcionamiento de la aplicación.

El contenido de la ayuda es un fichero html en el que se explica a grandes rasgos lo que se ha comentado en los puntos anteriores referentes al manual de usuario, con lo que no se repetirá aquí.

# VALORACIÓN DEL TRABAJO REALIZADO

Como ya se ha comentado con anterioridad, se pretende que nuestra herramienta sirva como base teórica para un mayor desarrollo futuro de trabajos en el campo de la demostración automática.

Por otra parte resulta útil durante el desarrollo del aprendizaje de la lógica. Con este fin, la herramienta ha sido puesta a disposición de los alumnos en el campus virtual de la UCM para que lo utilicen los alumnos de los primeros cursos.

# APÉNDICES

## APÉNDICE A: Analizador Sintáctico

La construcción de los tableaux se realiza a partir de objetos Fórmula, mientras que la información introducida por el usuario es simplemente una cadena de caracteres. Por tanto, es necesario aplicar una traducción antes de comenzar el procesamiento.

Esta traducción se realiza mediante un analizador sintáctico predictivo recursivo, con un único símbolo de anticipación. La gramática sobre la que opera este analizador, junto con las transformaciones necesarias para adaptarla a este tipo de análisis (para utilizar esta clase de analizador es imprescindible que la gramática sea de tipo LL1), se muestra a continuación.

Form → FormAtom
| FormComp
| FormCuant
| ¬ Form

FormAtom → pred
| pred(Args)

Args → Args Arg
| Arg

Arg → cons
| var
| fun(Args)

FormComp → FormConj
| FormDisy

FormConj → Form ∧ Form
| ¬ (Form ∨ Form)
| ¬ (Form → Form)

FormDisy → Form ∨ Form
| ¬ (Form ∧ Form)
| Form → Form

FormCuant → ∀ Form
| ∃ Form

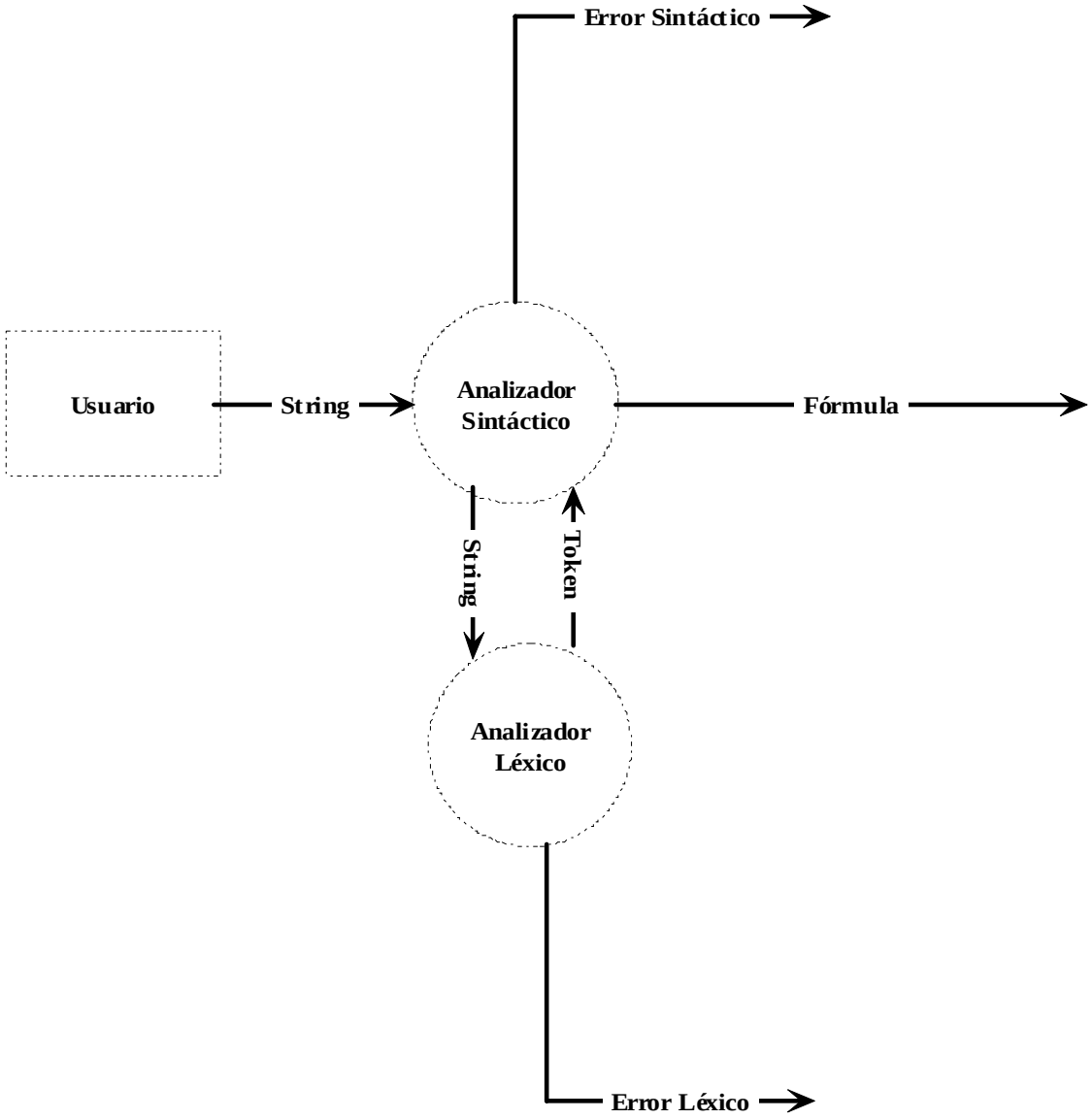
Considerando la lógica proposicional como un subconjunto de la lógica de primer orden (donde las proposiciones son predicados de 0 argumentos), un único analizador es suficiente para procesar cualquier tipo de fórmula.

Dado que el alfabeto del analizador sintáctico es el conjunto de tokens (unidades léxicas) de la gramática, y que el alfabeto de entrada es el conjunto de caracteres imprimibles, se ha de introducir un analizador léxico que descomponga la cadena introducida por el usuario en sus unidades léxicas. Este proceso no se realiza de manera independiente, si no que es requerido por el analizador sintáctico a medida que va necesitando nuevos tokens. Es decir, los analizadores léxico y sintáctico entrelazan su operación.

Los posibles errores léxicos o sintácticos presentes en la entrada introducida por el usuario se detectan durante esta fase, produciendo un mensaje de error y deteniendo la ejecución en caso de que no sea posible construir una fórmula correcta.

Adicionalmente, el analizador sintáctico realiza otras funciones con el fin de facilitar el procesamiento posterior de la fórmula. La principal de ellas es la de renombrar las variables ligadas, introduciendo nombres de variables reservados (no permitidos para el usuario) con el fin de evitar posteriores problemas de ambigüedad en las referencias a nombres de variables. Además, el analizador almacena información que posteriormente utilizarán las diversas etapas del procesamiento (nombres de funciones junto con su aridad, nombres de constantes introducidos...).

Gráficamente, el funcionamiento es como sigue:



Ejemplo 1

El usuario introduce la cadena "(p $\wedge$ q)", que será el String que reciba el analizador sintáctico. El resultado del análisis será una fórmula alfa, de tipo 1, compuesta de dos fórmulas atómicas.

Ejemplo 2

La cadena introducida es "(p $\wedge\wedge$ q)". Dado que no existe ninguna producción en la gramática que contemple en uso de dos símbolos '$\wedge$' consecutivos, esta entrada producirá un error sintáctico que se mostrará por pantalla abortando el análisis

Ejemplo 3

Se introduce "(p % q)". En este caso, no se puede descomponer la cadena en tokens, dado que '%' no es un símbolo del lenguaje de las fórmulas. Por tanto, el analizador léxico mostrará un error y se detendrá el análisis.

APÉNDICE B: Representación de las fórmulas

Internamente, se distinguen tres clases fundamentales de fórmulas: las atómicas, las compuestas y las cuantificadas.

Las fórmulas atómicas se representan, simplemente, por un predicado (que incluye el símbolo que lo representa, y su conjunto de argumentos), y un indicador de negación. En el caso de la lógica proposicional, se almacenará un predicado de 0 argumentos, por lo que resulta innecesario hacer una separación explícita entre fórmulas atómicas proposicionales y de primer orden.

Ejemplo 1:

La fórmula atómica P(X,c1), se representará como un objeto Predicado de nombre P y argumentos [X,c1] y el indicador negado = false.

Ejemplo 2:

Un átomo proposicional ¬q será simplemente un Predicado de identificador q y lista de argumentos [], junto con el indicador negado = true.

En el caso de las fórmulas compuestas se distinguen dos subcasos, las fórmulas alfa (conjuntivas), y las fórmulas beta (disyuntivas). En ambos casos, la representación consiste en dos fórmulas independientes, y un entero en el rango [1,3] que indica el modo en el que se relacionan esas dos fórmulas.

En las fórmulas alfa, los tres posibles tipos se corresponden con los siguientes casos:

Tipo 1 → F1 & F2
Tipo 2 → ¬(F1 | F2)
Tipo 3 → ¬(F1 -> F2)

Para las beta, los tipos representan:

Tipo 1 → F1 | F2
Tipo 2 → ¬(F1 & F2)
Tipo 3 → F1 -> F2

Ejemplo 1:

P(X) & Q(Y) se representará como una fórmula alfa de tipo 1, donde F1 es la fórmula atómica P(X) (almacenada como se vio anteriormente) y F2 es Q(Y)

Ejemplo 2:

(p & q) -> r será una fórmula beta de tipo 3. En este caso, F1 será a su vez una fórmula alfa de tipo 1, y F2 una atómica

Las fórmulas cuantificadas engloban las delta (universales) y las gamma (existenciales). En este caso se almacena una única fórmula F, así como el identificador de la variable implicada en la cuantificación y el rango en el que esta se encuentra. En el caso de que no se especifique el rango, se utiliza el máximo rango posible.

Ejemplo:

@X:[3,8]:P(X) se almacenará internamente como una fórmula gamma, con F la fórmula atómica P(X), e identificador de variable cuantificada "X". Además, se almacenará el rango especificado.

Ejemplo:

#Y:(Q(Y) -> S(c1)). En este caso se trata de una fórmula delta, cuya F será una beta de tipo 3, y su variable cuantificada "Y". Dado que no se especifica rango, se tomará el mayor rango posible(desde MINRAN hasta MAXRAN, constantes declaradas en la clase Rango).

## APÉNDICE C: Código, Métodos Relevantes

A continuación se muestra el código del método "Cerrados" que comprueba si un tableau es cerrado.

```
public boolean cerrado(Hashtable lista, Hashtable rangos,
                       Hashtable valores,Point p, Graphics g,
                       int tam,int ancho,int alto, Point puntoFin){
// Guardo en la lista el identificador de predicado, con información
de //negado/no negado y su lista de parámetros. Si
// estaba, añado nueva lista de parámetros. Si estaba negado,
compruebo
// unificación.
// Fórmula es una atómica, con Predicado y Negado
try{
      Color cAzul = new Color(0,0,200);
      Color col = new Color(0,0,0);
      col = cAzul;
      g.setColor(col);
      if (formula != null) {
            formula.dibuja(p,g,tam,puntoFin);
            Predicado pred = (Predicado)formula.getTerm();
            String id = pred.getNombre();
            String idNeg = "!"+id;
            if (!formula.isNegada()) {
// tengo que buscar en la tabla hash su complementaria, idNeg
                  if (lista.containsKey(idNeg)) {
//Comprobar si unifica la lista de parámetros con alguna de las
complementarias
                        ArrayList parCom =
                                    (ArrayList)lista.get(idNeg);
                        for (int i = 0;i<parCom.size();i++){
                              if
(unif((ArrayList)parCom.get(i),pred.getParametros())) {

                                    Info.volcarLocales();
                              return true;
                              }
                        }
                  }
// si he llegado hasta aquí, es que no se ha cerrado y tengo que meter
el nuevo pred
                  if (lista.containsKey(id)){
                        ArrayList listaPar = (ArrayList)lista.get(id);
                        listaPar.add(pred.getParametros());
                        lista.put(id,listaPar);
                  }
                  else {
                        ArrayList listaPar = new ArrayList();
                        listaPar.add(pred.getParametros());
                        lista.put(id,listaPar);
                        }
                  }
                  else {
// tengo que buscar en la tabla hash su complementaria, id
                        if (lista.containsKey(id)) {
//Comprobar si unifica la lista de parámetros con alguna de las
complementarias
                              ArrayList parCom =
(ArrayList)lista.get(id);
                              for (int i = 0;i<parCom.size();i++){
```

```
                                          if
(unif((ArrayList)parCom.get(i),pred.getParametros())) {
                                                Info.volcarLocales();
                                                return true;
                                          }
                                    }
                              }
// si he llegado hasta aquí, es que no se ha cerrado y tengo que meter
el nuevo pred
                              if (lista.containsKey(idNeg)){
                                    ArrayList listaPar =
(ArrayList)lista.get(idNeg);
                                    listaPar.add(pred.getParametros());
                                    lista.put(idNeg,listaPar);
                              }
                              else {
                                    ArrayList listaPar = new
ArrayList();
                                    listaPar.add(pred.getParametros());
                                    lista.put(idNeg,listaPar);
                              }
                        }


                  }

                  g.setColor(col);
                  ((Graphics2D)g).fillOval((int)p.getX()-
5,(int)p.getY()-5,10,10);

                  if (hijoIz != null && hijoDer == null){
                        g.setColor(col);
                        Point fin = new
                              Point((int)p.getX(),(int)p.getY()+tam);

      g.drawLine((int)p.getX(),(int)p.getY(),(int)fin.getX(),(int)fin.
getY());
                        return
hijoIz.cerrado(lista,rangos,valores)||hijoIz.cerrado((Hashtable)lista.
clone(),(Hashtable)rangos.clone(),

      (Hashtable)valores.clone(),fin,g,tam,ancho,alto,puntoFin);

                  }
                  else if (hijoIz != null && hijoDer != null){
                        g.setColor(col);

      g.drawLine((int)p.getX(),(int)p.getY(),(int)p.getX()+tam+ancho,(
int)p.getY()+tam+alto);

      g.drawLine((int)p.getX(),(int)p.getY(),(int)p.getX()-tam-
ancho,(int)p.getY()+tam+alto);

      //((Graphics2D)g).fillOval((int)p.getX()+tam+ancho-
5,(int)p.getY()+tam+alto-5,10,10);
                        //((Graphics2D)g).fillOval((int)p.getX()-tam-
ancho-5,(int)p.getY()+tam+alto-5,10,10);
                        Point p1 = new Point((int)p.getX()-tam-
ancho,(int)p.getY()+tam+alto);
                        Point p2 = new
Point((int)p.getX()+tam+ancho,(int)p.getY()+tam+alto);
                        return
(((hijoIz.cerrado(lista,rangos,valores)&&hijoDer.cerrado(lista,rangos,
valores)||hijoIz.cerrado((Hashtable)lista.clone(),(Hashtable)rangos.cl
```

```
one(),(Hashtable)valores.clone(),p1,g,tam-5,ancho-
2,alto+2+20,puntoFin)
                                    &&
hijoDer.cerrado((Hashtable)lista.clone(),(Hashtable)rangos.clone(),(Ha
shtable)valores.clone(),p2,g,tam-5,ancho-2+5,alto+2,puntoFin))
                        ));
                  }
                  else{
                        return false;
                  }
            }

            catch(Exception e){return false;}
      }
```

```
public boolean cerrado(Hashtable lista,Hashtable rangos,Hashtable
valores){
      if (formula!=null){
            Predicado pred = (Predicado)formula.getTerm();
            String id = pred.getNombre();

// Primero, se comprueba si se trata de un predicado evaluable
            if (Info.evaluable(id)){
                  if (contradiccionEval(rangos,valores))
                        return true;
                  }
// Tratamos de cerrar por unificación
//else return this.cerrado(lista,rangos);
                  }

// Si el nodo no provoca el cierre de la rama, tratamos de cerrar sus
hijos
                  if (hijoIz != null && hijoDer == null){
                        return
hijoIz.cerrado((Hashtable)lista.clone(),(Hashtable)rangos.clone(),(Has
htable)valores.clone());
                  }
                  else if (hijoIz != null && hijoDer != null){
                        return
(hijoIz.cerrado((Hashtable)lista.clone(),(Hashtable)rangos.clone(),(Ha
shtable)valores.clone())
                                          &&
hijoDer.cerrado((Hashtable)lista.clone(),(Hashtable)rangos.clone(),(Ha
shtable)valores.clone()));
                  }
                  else
                        return false;
            }
```

```
private boolean contradiccionEval(Hashtable rangos,Hashtable valores){

                  Predicado pred = (Predicado)formula.getTerm();
                  String id = pred.getNombre();

                  //Construimos el predicado evaluable
                  Evaluable eval = Info.getPred(id);
                  eval.setParametros(pred.getParametros());
                  String varInv="";
                  int numVars = 0;
                  for (int i=0;i<pred.getParametros().size();i++){
                        if (pred.getParametros().get(i) instanceof
Variable){

      varInv=((Variable)pred.getParametros().get(i)).getNombre();
                              numVars++;
                        }
                  }

                  // Si se puede evaluar a falso, terminamos
                  if (eval.esEvaluable()){
                        if (!eval.evalua()){
                              if (!formula.isNegada())
                                    return true;
                        }
                        else
                              if (formula.isNegada())
                                    return true;
                  }

                  if (numVars == 1){
                        // Comprobamos si tenemos un rango o un valor
para la variable involucrada
                        Rango rangoViejo = null;
                        if (rangos.containsKey(varInv)){
                              rangoViejo = (Rango)rangos.get(varInv);
                        }
                        int valorViejo = -1;
                        if (valores.containsKey(varInv)){
                              valorViejo =
((Integer)valores.get(varInv)).intValue();
                        }

                        // Igualdad/desigualdad
                        if (id.equals("iguales")) {
                              int valor = ((Iguales)eval).dameValor();
                              if (valor != -1){
                                    if (!formula.isNegada()){
                                          if
(valor!=valorViejo&&valorViejo!=-1)//Esto lo he añadido yo
                                                return true;
                                          if (rangoViejo!=null){
                                                if
(!rangoViejo.contiene(valor))
                                                      return true;
                                          }
                                          Integer valorNuevo = new
Integer(valor);

      valores.put(varInv,valorNuevo);
                                    }
                                    else { //Caso de !iguales
                                          if (valor==valorViejo)
                                                return true;
                                          if (rangoViejo!=null){
```

```
                                                if
(rangoViejo.unitario()){
                                                      if
(valor==rangoViejo.getRangoUnitario())
                                                            return
true;
                                                }
                                          }

                                    }
                              }
                        }

                        // Comparación

                        else if (id.equals("menor")){
                              if(!formula.isNegada()){
                                    try{

                                          if
(varInv.equals(((Variable)pred.getParametros().get(0)).getNombre())){
                                                int
valor=Integer.parseInt(((Constante)pred.getParametros().get(1)).getId(
));
                                                if (valorViejo != -1) {

      if(valorViejo>=valor) return true;
                                                }
                                                else {
                                                      Rango
rangoNuevo=eval.devuelveRango();
                                                      if (rangoNuevo !=
null) {
                                                            if
(rangoViejo!=null)

      rangoNuevo=rangoViejo.interseccion(rangoNuevo);
                                                            if
(rangoNuevo == null)

      return true;

      rangos.put(varInv,rangoNuevo);
                                                      }
                                                }
                                          }
                                    }
                                    catch(Exception e){

      if(varInv.equals(((Variable)pred.getParametros().get(1)).getNomb
re())){
                                                int
valor=Integer.parseInt(((Constante)pred.getParametros().get(0)).getId(
));
                                                if (valorViejo != -1) {

      if(valorViejo<=valor) return true;
                                                }
                                                else {
                                                      Rango
rangoNuevo=eval.devuelveRango();
                                                      if (rangoNuevo !=
null) {
                                                            if
(rangoViejo!=null)
```

```
      rangoNuevo=rangoViejo.interseccion(rangoNuevo);
                                                            if
(rangoNuevo == null)

      return true;

      rangos.put(varInv,rangoNuevo);
                                                                  }
                                                            }
                                                      }

                                                }
                                          }


                                          else if(formula.isNegada()){

                                                try {

      if(varInv.equals(((Variable)pred.getParametros().get(0)).getNomb
re())){
                                                            int
valor=Integer.parseInt(((Constante)pred.getParametros().get(1)).getId(
));

                                                            if (valorViejo != -1) {

      if(!(valorViejo>=valor)) return true;
                                                            }
                                                            else {
                                                                  Rango
rangoNuevo=eval.devuelveRango();//Esto hay que mirarlo (tiene que ser
como en mayor)

      rangoNuevo=rangoNuevo.complementario();
                                                                  if (rangoNuevo !=
null) {
                                                                        if
(rangoViejo!=null)

      rangoNuevo=rangoViejo.interseccion(rangoNuevo);
                                                                        if
(rangoNuevo == null)

      return true;

      rangos.put(varInv,rangoNuevo);
                                                                  }
                                                            }
                                                      }
                                                }
                                                catch(Exception e){

      if(varInv.equals(((Variable)pred.getParametros().get(1)).getNomb
re())){
                                                            int
valor=Integer.parseInt(((Constante)pred.getParametros().get(0)).getId(
));
                                                            if (valorViejo != -1) {

      if(valorViejo>valor) return true;
                                                            }
                                                            else {
```

```
                                                      Rango
rangoNuevo=eval.devuelveRango();

      rangoNuevo=rangoNuevo.complementario();
                                                      if (rangoNuevo !=
null) {
                                                            if
(rangoViejo!=null)

      rangoNuevo=rangoViejo.interseccion(rangoNuevo);
                                                            if
(rangoNuevo == null)

      return true;

      rangos.put(varInv,rangoNuevo);
                                                      }
                                                }
                                          }
                                    }


                              }
                        }

                        else if (id.equals("mayor")){
                              if(!formula.isNegada()){
                                    try{
                                          if
(varInv.equals(((Variable)pred.getParametros().get(0)).getNombre())){
                                                int
valor=Integer.parseInt(((Constante)pred.getParametros().get(1)).getId(
));
                                                if (valorViejo != -1) {

      if(valorViejo<=valor) return true;
                                                }
                                                else {
                                                      Rango
rangoNuevo=eval.devuelveRango();
                                                      if (rangoNuevo !=
null) {
                                                            if
(rangoViejo!=null)

      rangoNuevo=rangoViejo.interseccion(rangoNuevo);
                                                            if
(rangoNuevo == null)

      return true;

      rangos.put(varInv,rangoNuevo);
                                                      }
                                                }
                                          }
                                    }
                                    catch(Exception e){

      if(varInv.equals(((Variable)pred.getParametros().get(1)).getNomb
re())){
                                                int
valor=Integer.parseInt(((Constante)pred.getParametros().get(0)).getId(
));
                                                if (valorViejo != -1) {
```

```
      if(valorViejo>=valor) return true;
                                                }
                                                else {
                                                      Rango
rangoNuevo=eval.devuelveRango();
                                                      if (rangoNuevo !=
null) {
                                                            if
(rangoViejo!=null)

      rangoNuevo=rangoViejo.interseccion(rangoNuevo);
                                                            if
(rangoNuevo == null)

      return true;

      rangos.put(varInv,rangoNuevo);
                                                      }
                                                }
                                          }
                                    }
                              }

                              else if(formula.isNegada()){
                                    try{
                                          if
(varInv.equals(((Variable)pred.getParametros().get(0)).getNombre())){
                                                int
valor=Integer.parseInt(((Constante)pred.getParametros().get(1)).getId(
));

                                                if (valorViejo != -1) {

      if(!(valorViejo<=valor)) return true;
                                                }
                                                else {
                                                      Rango
rangoNuevo=eval.devuelveRango();//Esto hay que mirarlo (tiene que ser
como en mayor)

      rangoNuevo=rangoNuevo.complementario();
                                                      if (rangoNuevo !=
null) {
                                                            if
(rangoViejo!=null)

      rangoNuevo=rangoViejo.interseccion(rangoNuevo);
                                                            if
(rangoNuevo == null)

      return true;

      rangos.put(varInv,rangoNuevo);
                                                      }
                                                }
                                          }
                                    }
                                    catch(Exception e){

      if(varInv.equals(((Variable)pred.getParametros().get(1)).getNomb
re())){
                                                int
valor=Integer.parseInt(((Constante)pred.getParametros().get(0)).getId(
));
```

```
                                                if (valorViejo != -1) {

      if(valorViejo<valor) return true;
                                                }
                                                else {
                                                      Rango
rangoNuevo=eval.devuelveRango();

      rangoNuevo=rangoNuevo.complementario();
                                                      if (rangoNuevo !=
null) {
                                                            if
(rangoViejo!=null)

      rangoNuevo=rangoViejo.interseccion(rangoNuevo);
                                                            if
(rangoNuevo == null)

      return true;

      rangos.put(varInv,rangoNuevo);
                                                      }
                                                }
                                          }
                                    }


                              }
                        }
                  }

                        //el predicado contiene 2 variables
                        else if (numVars == 2){

                              //obtenemos los rangos/valores de las 2
variables
                              String var1 =
((Variable)pred.getParametros().get(0)).getNombre();
                              String var2 =
((Variable)pred.getParametros().get(1)).getNombre();

                              Rango ran1 = null;
                              Rango ran2 = null;
                              int val1 = -1;
                              int val2 = -1;

                              if (rangos.containsKey(var1)){
                                    ran1 = (Rango)rangos.get(var1);
                              }
                              if (valores.containsKey(var1)){
                                    val1 =
((Integer)valores.get(var1)).intValue();
                              }
                              if (rangos.containsKey(var2)){
                                    ran2 = (Rango)rangos.get(var2);
                              }
                              if (valores.containsKey(var2)){
                                    val2 =
((Integer)valores.get(var2)).intValue();
                              }

                              //si una de las variables tiene un valor
concreto, el caso es análogo al de una variable
                              if (id.equals("iguales")){
                                    if(!formula.isNegada()){
```

```
                                        if(val1 == -1){
      if(ran1!=null&ran2!=null){
      if(ran1.interseccion(ran2)==null) return true;
                                                      else
{rangos.put(var1,ran1.interseccion(ran2));
      rangos.put(var2,ran1.interseccion(ran2));//así acotamos el rango
                                                      }
                                              }
                                        }
                                        else if(!ran2.contiene(val1))
return true;
                                        else {
                                              Integer valorNuevo=new
Integer(val1);
      valores.put(var2,valorNuevo);
                                        }
                                        if(val2 ==-1){
      if(ran1!=null&ran2!=null){
      if(ran2.interseccion(ran1)==null) return true;
                                                      else
{rangos.put(var1,ran1.interseccion(ran2));
      rangos.put(var2,ran1.interseccion(ran2));//así acotamos el rango
                                                      }
                                              }
                                        }
                                        else if(!ran1.contiene(val2))
return true;
                                        else {
                                              Integer valorNuevo=new
Integer(val2);
      valores.put(var1,valorNuevo);
                                        }
                                  }
                                  else if(formula.isNegada()){//Sólo
sé que no son distintos si son unitarios y mismo valor
                                        if(val1 != -1&&val2 !=-1){
                                              if(val1==val2) return
true;
                                        }
                                  }


                            }
                            else if(id.equals("menor")){
                                  if(!formula.isNegada()){
                                        if(val1!=-1 && val2!=-1 &&
val1>=val2) return true;
                                        if(val1 != -1){
      if(ran2!=null&&val1>=ran2.getLimSup()) return true;//mirar si>=
                                        }
                                        if(val2 !=-1){
      if(ran1!=null&&val2<=ran1.getLimInf()) return true;
                                        }
```

```
                                          else
if(ran2!=null&&ran1!=null&&ran1.getLimSup()>=ran2.getLimInf()) return
true;
                                    }
                                    else{
                                          if(val1!=-1 && val2!=-1 &&
val1<val2) return true;
                                          if(val1 != -1){

      if(ran2!=null&&val1<=ran2.getLimInf()) return true;
                                          }
                                          if(val2 !=-1){

      if(ran1!=null&&val2>ran1.getLimSup()) return true;
                                          }
                                          else if
(ran2!=null&&ran1!=null&&ran1.getLimSup()<ran2.getLimInf())return
true;
                                    }
                              }

                              else if(id.equals("mayor")){
                                    if(!formula.isNegada()){
                                          if(val1!=-1 && val2!=-1 &&
val1<=val2) return true;
                                          if(val1 != -1){

      if(ran2!=null&&val1<=ran2.getLimInf()) return true;
                                          }
                                          if(val2 !=-1){

      if(ran1!=null&&val2>=ran1.getLimSup()) return true;
                                          }
                                          else if (ran1!=null &&
ran2!=null && ran1.getLimInf()<=ran2.getLimSup())return true;
                                    }
                                    else{
                                          if(val1!=-1 && val2!=-1 &&
val1>val2) return true;
                                          if(val1 != -1){
                                                if(ran2 != null &&
val1>ran2.getLimSup()) return true;
                                          }
                                          if(val2 !=-1){
                                                if(ran1!=null &&
val2<ran1.getLimInf()) return true;
                                          }
                                          else if(ran1!=null &&
ran2!=null && ran1.getLimInf()>ran2.getLimSup()) return true;
                                    }
                              }

                        }

                  return false;
            }
```

Métodos que crean el tableau y combinan las fórmulas Alfa.

```
public Tableaux creaTableaux(){
            Tableaux t1,t2;

            switch (tipo) {
            case (1): {t1 = formula1.creaTableaux();
                  t2 = formula2.creaTableaux();
                  break;
            }// fin tipo = 1
            case (2): {t1 = formula1.negar().creaTableaux();
                  t2 = formula2.negar().creaTableaux();
                break;
            }// fin tipo = 2
            case (3): {t1 = formula1.creaTableaux();
                  t2 = formula2.negar().creaTableaux();
                  break;
            }// fin tipo = 3
            default: {t1 = null;t2 = null;}
            }
            Tableaux resultado = combina(t1,t2);
            return resultado;
      }

      public Tableaux combina(Tableaux t1,Tableaux t2){
            //colocamos encima el de menor ramificación
            Tableaux resultado;
            ArrayList temp = new ArrayList();
            if (t1.getRamas() <= t2.getRamas()){
                  resultado = t1;
                  for (int i=0;i<resultado.getHojas().size();i++){

      ((NodoTableaux)resultado.getHojas().get(i)).setHijoIz(t2.getRaiz
().getHijoIz());

                        temp.addAll(t2.getHojas());
                        if (t2.getRaiz().getHijoDer()!=null) {

      ((NodoTableaux)resultado.getHojas().get(i)).setHijoDer(t2.getRai
z().getHijoDer());

                              temp.addAll(t2.getHojas());
                        }

                  }
                  resultado.setRamas(t1.getRamas()*t2.getRamas());
                  resultado.setHojas(temp);
                  return resultado;
            }
            else return combina(t2,t1);
      }
```

Métodos que crean el tableau y combinan las fórmulas Beta.

```
public Tableaux creaTableaux(){

            Tableaux t1,t2;

            switch (tipo) {
            case (1): {t1 = formula1.creaTableaux();
                  t2 = formula2.creaTableaux();
                  break;
            }// fin tipo = 1
            case (2): {t1 = formula1.negar().creaTableaux();
                  t2 = formula2.negar().creaTableaux();
                break;
            }// fin tipo = 2
            case (3): {t1 = formula1.negar().creaTableaux();
                  t2 = formula2.creaTableaux();
                  break;
            }// fin tipo = 3
            default: {t1 = null;t2 = null;}
            }

            Tableaux resultado = combina(t1,t2);
            return resultado;
      }

      public Tableaux combina(Tableaux t1,Tableaux t2){

            NodoTableaux ntIz,ntDer;
            if (t1.getRaiz().getHijoDer() == null)
                  ntIz = t1.getRaiz().getHijoIz();
            else
                  ntIz = t1.getRaiz();
            if (t2.getRaiz().getHijoDer() == null)
                  ntDer = t2.getRaiz().getHijoIz();
            else
                  ntDer = t2.getRaiz();
            NodoTableaux raiz = new NodoTableaux(null,ntIz,ntDer);
            Tableaux resultado = new Tableaux();
            resultado.setRaiz(raiz);
            resultado.setHojas(t1.getHojas());
            boolean ok = resultado.getHojas().addAll(t2.getHojas());
            resultado.setRamas(t1.getRamas()+t2.getRamas());
            return resultado;
```

# BIBLIOGRAFÍA

[1] **Isabel Pita, Rafael del Vado**. *Estudio de una experiencia de aprendizaje interactivo para la asignatura de Estructura de Datos a través del Campus Virtual*. Proceedings of the *IV Jornada Campus Virtual UCM*. Madrid (España), 2007 (por aparecer).

[2] **María Teresa Hortalá, Javier Leach Albert, Mario Rodríguez Artalejo**. *Matemática Discreta y Lógica Matemática*. 2ª Edición. Editorial Complutense, 2001.

[3] **Melvin Fitting**. *First-Order Logic and Automated Theorem Proving*. 2ª Edición. Springer, Graduate Texts in Computer Science, 1990.

[4] **Narciso Martí Oliet, Clara Segura Díaz, Alberto Verdejo**. *Especificación, derivación y análisis de algoritmos. Ejercicios resueltos*. Colección Prentice Practica, Pearson/Prentice Hall, 2006.

[5] **Ricardo Peña**. *Diseño de Programas: Formalismo y Abstracción*. 3ª Edición. Pearson Prencite Hall, 2005.

[6] **A. Sarasa, Rafael del Vado**. *Propositional Logic Learning Objects.* Proceedings of the *Second International Congress on Tools for Teaching Logic* (SICTTL'06). Salamanca (España).

# PÁGINA DE AUTORIZACIÓN

Se autoriza a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.