SAT. Una herramienta didáctica para el problema de la satisfacibilidad

Tomás Covelli, Enzo Nicolás Horquín, Martín Santillán Cooper

Resumen. La herramienta 'SAT' fue creada como trabajo final de dos materias del segundo año de la carrera de Ingeniería de Sistemas. SAT permite, dada una fórmula de la Lógica Proposicional expresada en forma normal conjuntiva (FNC), decidir si es satisfacible o no. En otras palabras, encontrar una configuración de valores para los literales de determinada fórmula booleana que evalúen la misma en verdadera. El objetivo de esta herramienta es el de complementar el estudio de los estudiantes en el área de Lógica Proposicional contando con una herramienta didáctica y fácil de usar. Se muestran dos aplicaciones que resuelven problemas computacionales a partir de la Lógica Proposicional.

1 Introducción

La Lógica es la ciencia que formaliza métodos válidos de razonamiento. El apartado más básico de la Lógica es la Lógica Proposicional. Sus fórmulas están formadas por variables proposicionales y conectores. Las variables, que se denotan con letras minúsculas, son la representación formal de proposiciones atómicas y pueden ser verdaderas o falsas, mientras que un literal representa a una variable o su negación. Los conectores se encargan de combinar literales para la construcción de cláusulas ó fórmulas proposicionales (conjunto de cláusulas). En general, se usan 4 conectores básicos: conjunción, disyunción, implicación y negación ($\wedge$, $\vee$, $\rightarrow$, $\sim$ o $\neg$ respectivamente) [6][9].

La satisfacibilidad booleana o satisfacibilidad proposicional (abreviado SAT), es el problema de determinar si existe una interpretación posible que satisfaga a una fórmula booleana o proposicional dada. En otras palabras, se encarga de detectar qué valores deben tomar los literales (verdadero o falso) de una fórmula booleana, para que la misma sea satisfacible (es decir, verdadera). En caso de que no haya una configuración posible que haga verdadera a dicha fórmula, se dice que la misma es insatisfacible.

2 SAT: La Herramienta

La herramienta 'SAT' busca determinar el resultado de la satisfacibilidad de una fórmula proposicional. El objetivo de este trabajo fue la implementación del algoritmo Davis Putnam Logemann Loveland (DPLL) que determina si cierta fórmula en FNC es satisfacible y del algoritmo de Búsqueda de Valuaciones, que indica todas las combinaciones posibles de valores que pueden tomar los literales para que la fórmula final sea verdadera.

Con el fin de que esta herramienta sea lo más didáctica posible (ya que está destinada al uso de alumnos de segundo año de la carrera), se introdujo el concepto de 'variable fija'. Una variable fija es una variable que siempre debe ser verdadera.

A continuación, se detalla la funcionalidad de cada uno de los algoritmos y su alcance.

adfa, p. 1, 2011.

2.1 Algoritmo Davis Putnam Logemann Loveland

Existen diferentes técnicas para conocer la satisfacibilidad de una fórmula booleana. Una de ellas (utilizada por la herramienta 'SAT') es la resolución proposicional. Se basa en la aplicación de continuas "resolventes" sobre una fórmula proposicional en FNC (organizada como una conjunción de cláusulas, donde cada cláusula es una disyunción de literales) hasta llegar a uno de dos posibles resultados: conjunto vacio (∅) o cláusula vacia (⊥).

Sean C1 y C2 dos cláusulas que conforman una determinada fórmula booleana, y suponiendo que $l \in C1$ y $\sim l \in C2$, definimos a la resolvente de la siguiente manera:

$$Res(C_1,C_2) = (C_1 - \{l\}) \cup (C_2 - \{\sim l\})$$

Si el proceso de resolución finaliza en ∅, se dice que el conjunto de cláusulas es satisfacible. En caso contrario (si hallamos en el camino ⊥), el mismo es insatisfacible.

El algoritmo DPLL sistematiza el proceso de resolución. Es comúnmente utilizado para casos en los que el conjunto de cláusulas es muy grande, y resulta complejo resolver el problema de SAT manualmente.

DPLL se basa principalmente en dos tipos de eliminaciones de literales: la unitaria y la pura. Antes de conocer cada eliminación es importante resaltar el concepto de cláusula unitaria y literal puro. Una cláusula C es unitaria si C tiene sólo un literal. Por otro lado, un literal l es puro de S si $l \in S$ y $\sim l \notin S$.

Conociendo las definiciones anteriores es posible definir ambos tipos de eliminación. Sea S un conjunto de cláusulas y {l} una cláusula unitaria de S. Entonces la eliminación unitaria de l en S es el conjunto obtenido borrando en S todas las cláusulas que contienen a l, y aplicando resolvente entre l y ~l en las cláusulas que contengan a ~l. Luego, si S = { {p,q,~r}, {p,~q}, {~p}, {r,u} }, eliminación unitaria (S,l):

S = { {p,q,~r}, {p,~q}, {~p}, {r,u} } [unitaria {~p}]
S = { {q,~r}, {~q}, {r,u} } [unitaria {~q}]
S = { {~r}, {r,u} } [unitaria {~r}]
S = { {u} }

Por otro lado, la eliminación pura en S es el conjunto obtenido al borrar todas las cláusulas que contengan a cierto literal puro l. Por lo tanto, si S = { {p,q}, {p,~q}, {r,q} }, eliminacionPura(S,l):

S = { {p,q}, {p,~q}, {r,q} } [puro p]
S = { {r,q} } [puro r]
S={{}}

Conociendo ahora cada tipo de eliminación, podemos enfocarnos plenamente en el algoritmo. El algoritmo DPLL recibe de entrada un conjunto de cláusulas S, y produce una salida que puede tomar dos valores: consistente e inconsistente. El procedimiento del algoritmo DPLL de S se muestra en Fig.1.

Un ejemplo de una ejecución simbólica del algoritmo DPLL es el siguiente:

S = {{a, b}, {¬a, b}, {a, ¬b}, {a, ¬d}, {¬a, ¬b, ¬c}, {b, ¬c}, {c, ¬f }, {f }}
[Unitaria {f }]
S = {{a, b}, {¬a, b}, {a, ¬b}, {a, ¬d}, {¬a, ¬b, ¬c}, {b, ¬c}, {c}}
[Unitaria {c}]
S = {{a, b}, {¬a, b}, {a, ¬b}, {a, ¬d}, {¬a, ¬b}, {b}} [Unitaria {b}]

S = {{a}, {a, ¬d}, {¬a}}
[Unitaria {¬a}]
S = {{ },{¬d}}

Luego, como ⊥ ∈ S, S es inconsistente.

2.2 Algoritmo generador de valuaciones

Para realizar la búsqueda de todas las posibles valuaciones que hacen al conjunto de cláusulas satisfacible, se implementó un algoritmo utilizando la técnica de Backtracking [3]. Éste es un método de búsqueda exhaustiva que analiza todas las posibles combinaciones de valores de verdad de las variables proposicionales que pertenecen al conjunto de cláusulas. Al aplicar esta técnica, la generación de la gran cantidad de ´hijos´ que representan el espacio de búsqueda, resulta en un algoritmo ineficiente. Para solucionar este problema se implementó una función “poda”.

La poda analiza si se puede o no llegar a una solución a partir de un cierto nodo del árbol que representa el espacio de búsqueda. En este caso, la poda se lleva a cabo si se encuentra una cláusula vacía.

En Fig. 2 se muestra el pseudo-código que genera las posibles valuaciones, mientras que Fig 3 muestra el árbol generado por el algoritmo a partir de la fórmula en FNC S= { {~a,b,~c} , {a,~d}, {d} }.

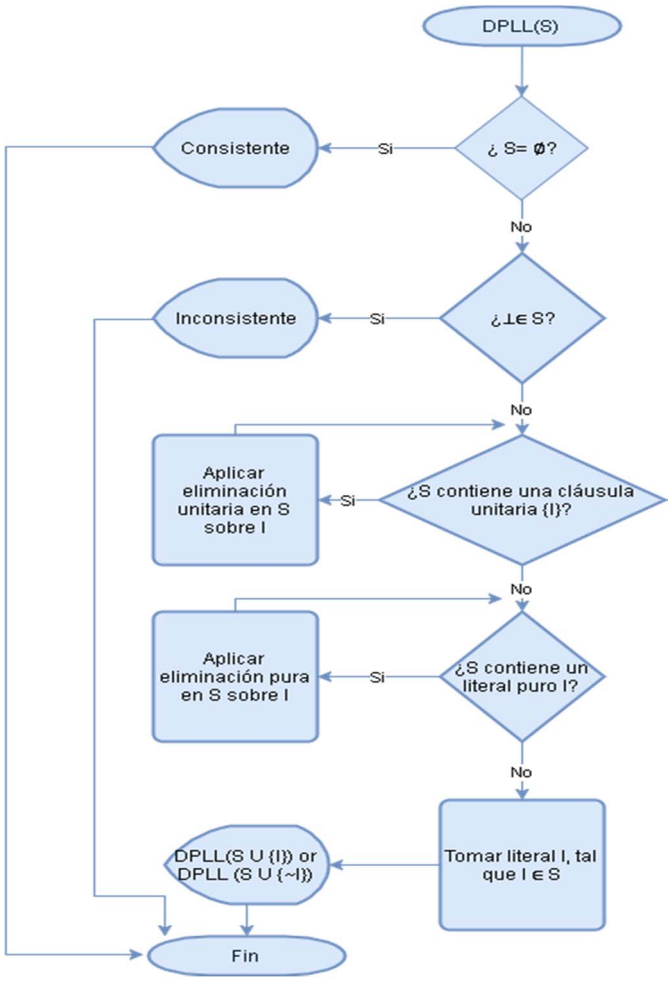

Fig. 1. Procedimiento del algoritmo DPLL.

```
void generar_Valuaciones (formula, valparcial, variablesFijas, valuaciones)
{
  if (! variablesFijas.empty() )
  {
    agregaVariablesFijas(valParcial, variablesFijas); //siempre serán parte de la solución
    eliminacionUnitariaVariablesFijas(variablesFijas, conjunto);
  }
  generar_Valuaciones(conjunto, valparcial, valuaciones);
}

void generar_Valuaciones (formula, valparcial, valuaciones)
{
  eliminacionUnitariaSucesiva(conjunto, valparcial);
  if (formula = ∅)
    completar_Valuaciones(formula, valparcial, valuaciones);
  else if (⊥ ∉ formula)
  {
    Tomar literal  l ∈ conjunto;
    generar_Valuaciones (conjunto ∪ {l} , valparcial,valuaciones);
    generar_Valuaciones(conjunto ∪ {~l}, valparcial,valuaciones);
  }
}
```

Fig. 2. Pseudocódigo del algoritmo generador de valuaciones.

En Fig.3 se muestra el árbol generado por el algoritmo a partir de la fórmula en FNC S= { {~a,b,~c} , {a,~d}, {d} }.

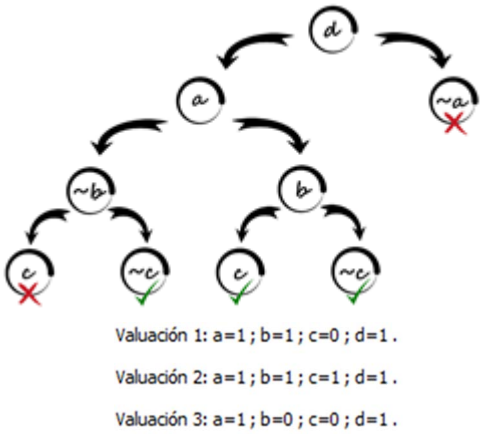

Fig. 3. Árbol de ejecución.

3 La herramienta

La funcionalidad principal de esta herramienta es la ejecución de los algoritmos mencionados anteriormente, a partir de un conjunto de cláusulas ingresadas por el usuario.

La carga de cláusulas es sencilla para el usuario considerando que son los alumnos que están aprendiendo los conceptos involucrados. Los literales se cargan separados entre comas y al seleccionar "Cargar cláusula" el conjunto se irá formando debajo.

Una vez que el conjunto ya esté finalizado, el usuario puede optar entre resolver el problema de satisfacibilidad mediante el algoritmo de DPLL ("Ver resolución") (véase Fig.5), u obtener todas las valuaciones posibles para los literales ("Ver valuaciones"). En la Figura 4 se muestra la interfaz de la herramienta.

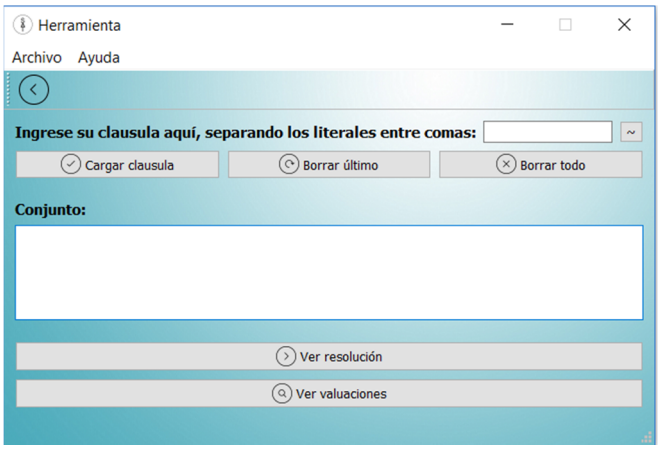

Fig. 4. Interfaz de la herramienta.

Una vez desarrollada la herramienta principal es fácil extender su funcionalidad para resolver problemas de satisfacción de restricciones (CSP, por sus siglas en inglés). Esto se debe a que mediante la lógica proposicional basta con interpretar las restricciones que se deducen del enunciado del problema y expresarlas como una fórmula proposicional en FNC de tal manera que pueda ser entrada para el algoritmo implementado. Por lo tanto,

en este punto, la dificultad en resolver un problema reside sólo en crear correctamente la fórmula proposicional y pasarla a FNC.

A continuación, se detallan los problemas para los cuales el conjunto de cláusulas está definido siendo los mismos problemas cuya lógica es conocida para el usuario alumno [4][5].

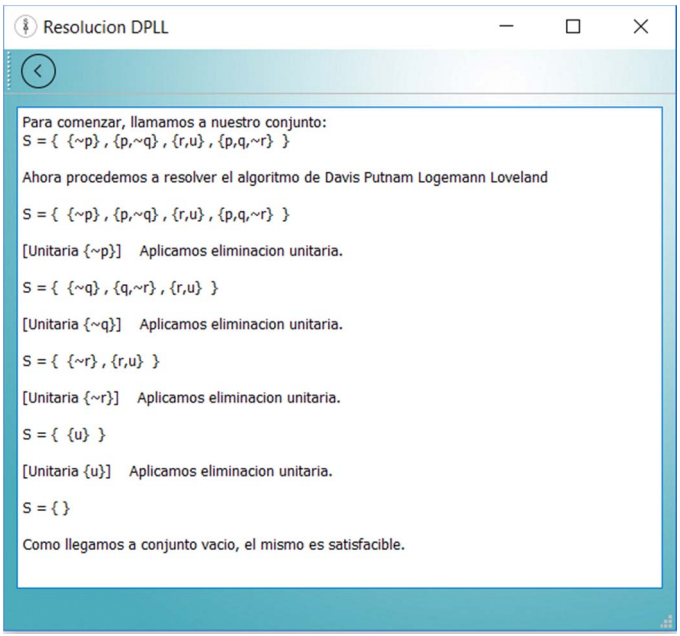

Fig. 5. Ejemplo de “Ver valuaciones”.

3.1 Problema de las N Reinas

El ‘Problema de las n Reinas’ consiste en colocar N reinas sobre un tablero de ajedrez de tamaño NxN sin que éstas se ´coman´ entre sí. El problema original, que fue planteado por Max Bezzel, trataba sólo con 8 reinas, pero al resolverlo computacionalmente es fácil extenderlo a cualquier número N. Este problema fue incluido porque es fácil e intuitivo de resolver a partir de la Lógica Proposicional, ya que su única dificultad reside en generar correctamente el conjunto de cláusulas para expresar las restricciones con elementos de la Lógica Proposicional.

Además de poder ver la resolución paso a paso y todas las valuaciones posibles, en esta herramienta el usuario puede ver las posibles soluciones encontradas por SAT plasmadas en un tablero de ajedrez con las N reinas colocadas en sus correspondientes lugares (véase Fig. 7).

Con el fin de darle más opciones al usuario, se incluyó la posibilidad de fijar reinas. Fijar una reina consiste en que el usuario puede elegir una celda del tablero y fijar una reina en ese lugar, por lo que una determinada disposición de reinas será solución si y sólo si ésta posee las reinas que han sido fijadas en sus respectivo lugares. La Figura 6 muestra la interfaz sencilla para este problema.

Se muestra ahora el método para la generación de las cláusulas en un tablero con 2 reinas. Se utiliza una variable proposicional por cada celda del tablero, donde cij indica el valor de verdad de “En la celda (i,j) hay una reina”. Se deben cumplir dos restricciones.

1ª restricción: Hay una reina por fila.

- (c11 ∨ c12) ∧ (c21 ∨ c22)

Que en FNC, se escribe:

- { {c11 , c12} , {c21 , c22} }

2ª restricción: Si en una casilla hay una reina, entonces no hay ninguna otra en su misma fila, columna o diagonal.

- c11 ➞ (~c12 ∧ ~c21 ∧ ~c22)
- c12 ➞ (~c22 ∧ ~c21)
- c21 ➞ ~c22

Usando equivalencias y operaciones lógicas se obtiene:

- (~c11 V ~c12) ∧ (~c11 V ~c21) ∧ (~c11 V ~c22)
- (~c12 V ~c22) ∧ (~c12 V ~c21)
- (~c21 V ~c22)

Que en FNC, se escribe:

- { { ~c11 , ~c12} , {~c11 , ~c21} , {~c11 , ~c22} }
- { { ~c12 , ~c22} , {~c12 , ~c21} }
- { { ~c21 , ~c22} }

Por lo tanto, el conjunto de cláusulas resultante es:

{{c11,c12},{c21,c22},{~c11,~c12},{~c11,~c21},{~c11,~c2}, {~c12,~c22} ,
{~c12,~c21} , {~c21,~c22}}

Es interesante observar que la cantidad de cláusulas que se generan para representar las restricciones crece rápidamente a medida que aumenta la cantidad de reinas (con 8 reinas se necesitan 736 cláusulas), por lo que en esta aplicación resulta didáctico ver las diferentes valuciones encontradas y su representación en el tablero de ajedrez, mientras que ver las resolución paso a paso del algoritmo DPLL para determinar la satisfacibilidad del conjunto de cláusulas no tiene tanto sentido.

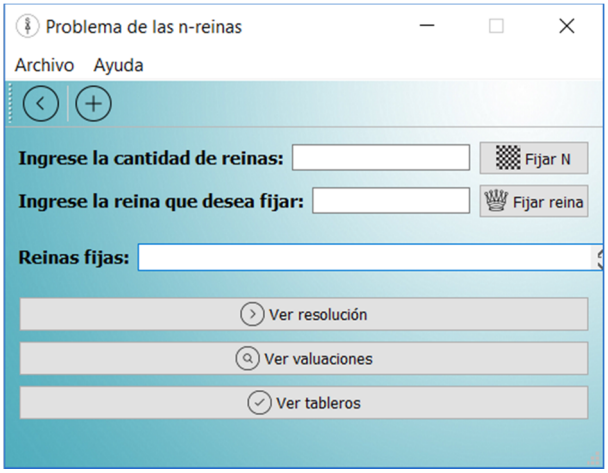

Fig. 6. Interfaz de NReinas.

3.2 Problema del Coloreo de Grafo

El problema del ´Coloreo de un Grafo' consiste en poder colorear un grafo con 3 colores, de tal manera que vértices conectados por una misma arista no posean el mismo color. Este problema, como el anterior, ha sido incluido por la facilidad de resolverlo a partir de la Lógica Proposicional. La Figura 6 muestra la interfaz para este problema.

Se le permite al usuario agregar hasta un máximo de seis vértices y armar todas las posibles combinaciones de aristas entre los vértices agregados.

Además de poder ver la resolución paso a paso y todas las valuaciones posibles, en esta herramienta el usuario podrá ver las soluciones encontradas por SAT dibujadas en un grafo con todos los vértices (coloreados) y aristas correspondientes.

Se muestra ahora el método para la generación de las cláusulas para el problema del coloreo de grafo. Las variables proposicionales son de la forma Color_v que representa a la proposición "el vértice v es de color Color", donde "Color" puede ser Rojo, Verde o Azul y "v" pertenece a V, el conjunto de vértices. Se deben cumplir tres restricciones.

1ª restricción: Cada vértice tiene al menos un color.

Se agrega la siguiente cláusula, para cada $v \in V$.

- {Rojo_v, Azul_v, Verde_v }

2ª restricción: Cada vértice tiene un solo color.

- Color1_v$\rightarrow$~Color2_v

Lo que, por reglas lógicas, equivale a:

- ~Color1_v V ~Color2_v

Que en FNC, se escribe:

- {~Color1_v , ~Color2_v}

Para el caso en que solo se usan 3 colores, se agrega la siguiente cláusula, para cada $v \in V$.

{~Rojo_v , ~Azul_v} , {~Rojo_v ,~Verde_v} , {~Azul_v , ~Verde_v}

3ª restricción: Dos vértices conectados por un mismo arco no tienen el mismo color.

Sea A el conjunto de aristas, $\forall$ arista $(a,b) \in A$, se debe cumplir que:

- Color_a$\rightarrow$~Color_b

Lo que, por reglas lógicas, equivale a:

- ~Color_a V ~Color_b

Que en FNC, se escribe:

- {~Color_a , ~Color_b}

Para el caso en que solo se usan 3 colores, se agrega la siguiente cláusula, para cada arista $(a,b) \in A$.

{ {~Rojo_a , ~Rojo_b} ,{~Azul_a , ~Azul_b} , {~Verde_a , ~Verde_b}

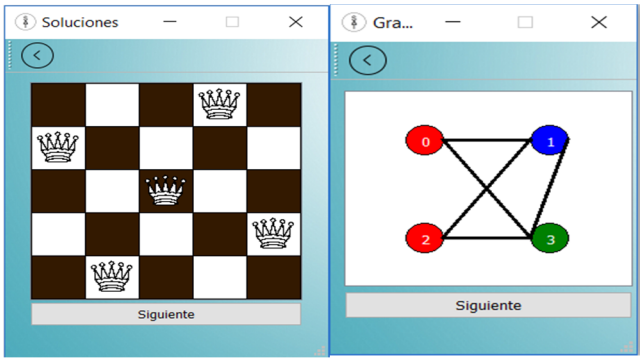

Fig. 7. Resolución para el problema del coloreo del grafo y de N reinas

4 Implementación

El diseño de la herramienta SAT es un diseño orientado a clases. Se crearon clases para representar cada uno de los elementos de la Lógica Proposicional. Ellos son: la clase Literal, la clase Cláusula y la clase ConjuntoCláusulas. Ellas son usadas para crear fórmulas que luego serán gestionadas por la clase DavisPutnam, en la cual se implementaron los algoritmos explicados en la sección II.

El lenguaje utilizado es C++ [1][8], y para la interfaz gráfica se utilizó el entorno de desarrollo QT [2]. Este entorno es muy adecuado para esta aplicación ya que no sólo cuenta con una interfaz sencilla para crear aplicaciones sino que también tiene un gran número de librerías y documentación para crearlas. Por otro lado, Qt resultó ser la mejor opción por su compatibilidad con el lenguaje C++.

El diagrama de clases que se muestra en la Figura 8 resume la implementación realizada.

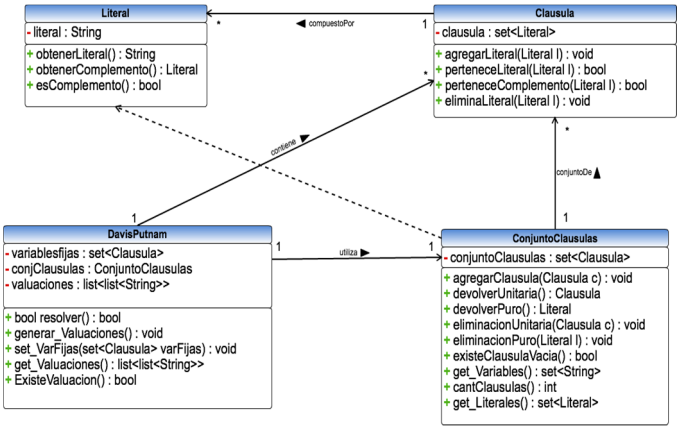

Fig. 8. Diagrama de clases de SAT.

5 Conclusiones

La herramienta SAT (Problema de la satisfacibilidad booleana) fue desarrollada tanto

para determinar la satisfacibilidad de un conjunto de cláusulas de manera eficiente, como también para servir de incentivo en el aprendizaje de la lógica proposicional. A pesar que los estudiantes, en los primeros años de la carrera, no tienen un amplio dominio del tipo de problemas que pueden ser resueltos mediante el uso de dicha rama de la lógica, esta herramienta alienta a la ejercitación en este área y la familiarización con herramientas educativas.

Uno de las aspectos más importantes a destacar es el uso de las clases Literal, Clausula y ConjuntoClausula. Debido a la detallada elección para la implementación de las estructuras, se facilitó la implementación de los algoritmos, logrando una buena performance en cuanto a la complejidad temporal de dichos algoritmos.

Se abordó este proyecto con objetivos claros y ambiciosos. Se logró implementar una herramienta didáctica gracias a que, por un lado, se puede mostrar al usuario (el alumno) de una forma sencilla e intuitiva los pasos que sigue el algoritmo Davis Putnam Logemann Loveland para determinar la satisfacibilidad de un conjunto de cláusulas. Por otro lado, enumerar las diferentes valuaciones encontradas para tal conjunto.

Es posible agregar la representación de otros problemas computacionales a la herramienta a fin de completarla para que pueda ser usada como soporte en cursos más avanzados.

Referencias

1. Code::Blocks, http://www.codeblocks.org
2. Digia Plc. Qt Documentation. http://doc.qt.io
3. E. Horowitz, S. Sahni, S. Rajasekaran, *Computer Algorithms / C++*, 2da ed., Silicon Press; 2 edition, 1998.
4. J. A. Alonso Jiménez, A. Cordón Franco, M. J. Hidalgo Doblado, *Lógica informática, Algoritmos para SA Aplicaciones,* Universidad de Sevilla, 2011.
5. J. A. Alonso Jiménez, A. Cordón Franco, M. J. Hidalgo Doblado, *Temas de "Lógica informática"*, Universidad de Sevilla, Recuperado de: http://www.cs.us.es/~jalonso/cursos/li-12/temas/temas-LI-2012-13.pdf
6. M. Ben-Ari, *Mathematical Logic for Computer Science*. Prentice Hall, Series in Computer Science, 1993.
7. SGI: Silicon Graphics International. I*ntroduction to the Standard Template Library* http://www.sgi.com/tech/stl
8. The c++ Resource Network http://www.cplusplus.com
9. V. Mauco, *Filminas de cátedra Ciencias de la Computación II: Lógica Proposicional,* Universidad Nacional del Centro de la Provincia de Buenos Aires, 2015.