THeM: Una Herramienta Didáctica para Modelos de Herbrand

Juan Ricardo Dahl, Daniel Fujii

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

{juandahl20, danifujii}@gmail.com

Resumen. La herramienta 'THeM' fue creada como trabajo de final de dos materias de segundo año de una carrera de Informática. Esta permite, dado un conjunto de cláusulas de la Lógica de Predicados de Primer Orden, determinar la existencia de Modelos de Herbrand para las mismas (lo cual permite conocer su satisfacibilidad). La herramienta provee una interfaz sencilla de utilizar y de entender, ya que uno de los objetivos es que sea usada por futuros alumnos de materias que estudien el tema.

1. Introducción

Los cursos básicos de Lógica se dictan en la mayoría de las carreras de Informática. En estos cursos, los estudiantes deben resolver gran cantidad de ejercicios para adquirir práctica en el manejo de formalismos. El uso de herramientas educativas puede ser un gran aporte en este sentido. Por esto, se consideró desarrollar como trabajo final para dos materias (ambas se dictan en el primer semestre del segundo año de una carrera de Informática) que involucran contenidos en lógica y en análisis y diseño de algoritmos, una herramienta didáctica que fuera intuitiva en su uso y cuya notación se correspondiera con aquella utilizada en el curso de Lógica.

La Lógica de Predicados de Primer Orden (LdPPO) [1] [2] [7] [8] incluye predicados y funciones (que se aplican sobre un dominio), para definir un modelo o interpretación. El valor de verdad de una fórmula depende de la definición del modelo en el cual se la evalúa, lo que le agrega dificultad en comparación con la lógica proposicional. Evaluar semánticamente una fórmula, en general, resulta más difícil para los estudiantes debido a la posibilidad de definir infinitos modelos arbitrarios.

La teoría de los Modelos de Herbrand [3] [6] [7] se aplica en la LdPPOpara construir modelos de una fórmula en forma canónica. Por lo tanto, puede ser utilizado para probar la validez o la satisfacibilidad de una fórmula en forma clausular.

Sea A un conjunto de cláusulas, una estructura o Modelo de Herbrand M es un modelo de Herbrand para A si es un modelo de A, es decir M $\models$A. Dado esto, podemos probar si A es satisfacible.

Considerando A una fórmula de la LdPPO, tenemos: si A es lógicamente válida o válida, ¬A es insatisfacible o contradictoria. Lo mismo sucede con cualquier forma clausular de A, denominada cl(A). Por esto:

- Si cl (¬A) no tiene Modelos de Herbrand entonces cl (¬A) es insatisfacible y A es válida.
- Si cl (¬A) tiene Modelos de Herbrand, cl (¬A) es satisfacible y entonces A no es válida.

Además, dado un razonamiento $H_1, H_2, \ldots, H_n \models C$ se puede buscar un modelo de Herbrand para probar si es satisfacible. Si se prueba que $H_1 \wedge H_2 \wedge \ldots \wedge H_n \wedge \neg C$ es insatisfacible entonces el razonamiento es válido.

Con este marco teórico, se puede comprender mejor el proyecto y sus logros. El conocer la existencia de Modelos de Herbrand para una fórmula en forma clausular puede utilizarse para diversos fines (probar un razonamiento, satisfacibilidad de una fórmula, entre otros).

La herramienta 'THeM' permite definir la satisfacibilidad de un conjunto de cláusulas de la LdPPO a través de la existencia de Modelos de Herbrand. Esto resulta importante ya que cualquier fórmula de esta lógica puede llevarse a forma clausular, y luego aplicar la herramienta para conocer su validez.

En la Sección 2 se describe el diseño global de la herramienta. La Sección 3 contiene una descripción de las principales funcionalidades de THeM. Finalmente, en la Sección 4 se presentan las conclusiones sobre el trabajo realizado.

2. Diseño de la Herramienta

Para implementar la herramienta, se siguió un diseño orientado a clases para representar cada uno de los elementos de la LdPPO mencionados anteriormente. La funcionalidad de cada uno nos permite resolver el problema planteado en la búsqueda de Modelos de Herbrand. El objetivo perseguido en este trabajo, es que esas clases puedan ser utilizadas para otros fines fuera de Modelos de Herbrand que requieran trabajar con cláusulas. Esto favorece el reuso de las mismas, uno de las ventajas que provee diseñar de esta manera. Se puede ver la conexión entre las clases que representan cada elemento de la LdPPO en la Figura 1.

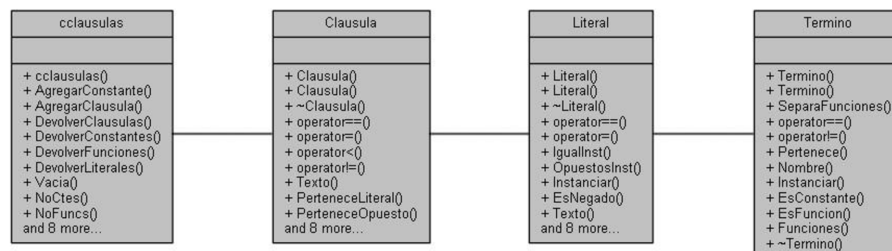

Figura 1: Clases de los elementos de una Clausula

Dentro de la Figura 2 se muestra cómo se representa una cláusula de ejemplo a partir de las clases creadas.

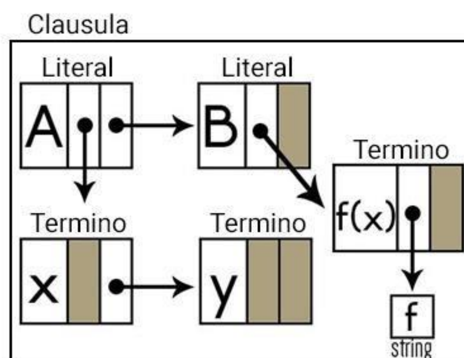

Figura 2: Representación de A(x, y) v B (f(x))

Para el funcionamiento de la herramienta se pueden reconocer tres algoritmos fundamentales: el de instanciar una cláusula, la búsqueda de modelos mediante Backtracking [10][11] y el mostrar el procedimiento para la búsqueda. Los algoritmos pertenecen a las clases ***clausula***, ***clausWindow*** y ***conjuntoClausulas*** respectivamente.

El procedimiento puede resumirse de la siguiente manera: primero, por cada cláusula añadida comparamos la nueva cláusula con las que fueron ingresadas anteriormente. Si una cláusula contiene un único literal, entonces ese literal instanciado formará parte de la solución. Por lo tanto, todas las cláusulas que contengan ese literal tendrán esa opción para la solución. En el caso de que se encuentre el literal, pero negado, sabemos que entonces ese literal no puede ser elegido. En ambos casos, "reducimos longitud" ya que disminuimos la cantidad de opciones disponibles a la hora de formar la solución.

Luego realizamos un ordenamiento de la lista de cláusulas que utiliza el operador < de la clase Clausula. La idea es tener las cláusulas unitarias[1] primero, y luego las que tengan menor cantidad de opciones a la hora de buscar modelos por lo explicado previamente.

Gracias a este mecanismo logramos simplificar tanto el diseño del algoritmo, básicamente un Backtracking, como el algoritmo para buscar modelos. De esta manera, el procedimiento se asemeja al mecanismo que se utiliza en la práctica por lo que resulta también más útil y natural para los alumnos.

Un aspecto esencial en la búsqueda de los Modelos de Herbrand es instanciar cada una de las cláusulas para poder hallar una solución. Realizarlo manualmente resulta intuitivo, pensar en un algoritmo puede que no resulte tanto, ya que podemos encontrarnos con constantes o funciones dentro de un literal.

[1] Definimos cláusula unitaria como una cláusula con un único literal.

De este modo, dicho algoritmo consiguió encontrar todas las combinaciones entre constantes y variables mediante el cálculo matemático c^v, donde c y v son la cantidad de constantes y variables respectivamente.

Tomando una variable, se debe reemplazar por cada una de las constantes c^{v-1} veces. Repitiendo este procedimiento para cada una de las variables es como se consigue instanciar cada una de las cláusulas. A continuación, se muestra un pseudocódigo del algoritmo:

```
voidListaInstancias(list<Clausula>&instanciados,list<Clausula>Claus
       ulas, set<string>constantes)
{ for (cada Clausula C∈Clausulas)
{ list<String> variables = C.variables();
     intcantInstans = Potencia(#ctes, #variables);
     intcantVariables = #variables; for(cada
     variable var∈ variables)
     { cantVariables = cantVariables-1;
      intcantInstan2 = Potencia(#ctes, #variables);
      intpos = 0; while(pos <cantInstans)
      { for(cada constante cte∈constantes)
             {     int limite=cantInstans2+pos;
      for (cada z=pos to limite)
      v_inst[z].Instanciar(var,cte);
                   pos = limite;
                   }
            }
      } for(cada Clausula C ∈ vector
      instanciados)
             instanciados.push_back(C);
  }
}
```

Para realizar la búsqueda de Modelos de Herbrand del conjunto de cláusulas se utiliza un algoritmo de Backtracking. Este es un método de búsqueda exhaustiva que analiza, en nuestro caso, las posibles soluciones que se pueden obtener.

Por cuestiones de eficiencia implementamos dos condiciones de “poda”:

- Al encontrarse insatisfacibilidad en la solución parcial no se continúa.
- En presencia de funciones se limita el número de modelos a 5.

En cada paso del Backtracking, se utiliza una función que devuelve una lista de los literales con los que puede continuar el algoritmo. Estos literales los obtiene de la siguiente cláusula instanciada a evaluar. En el caso de que uno de estos pertenezca a la solución parcial entonces se utiliza tal literal. Sin embargo, sí pertenece el literal

complementario[2] entonces éste no será considerado. El funcionamiento básico del algoritmo se observa en la figura 3.

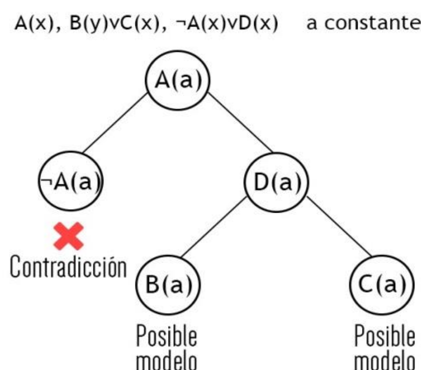

Figura 3: Espacio de búsqueda del Backtracking para las cláusulas dadas

A continuación, se presenta el pseudo-código del algoritmo de Backtracking, omitiendo la limitación de modelos en presencia de funciones por cuestiones de simplicidad. *Instanciadas* es la lista de cláusulas ya instanciadas con los diferentes términos cerrados, *final* es la lista resultado con los diferentes modelos y *solucion* es una lista de literales que forman parte de la solución parcial.

```
bool Back(list<Clausula>Instanciadas, list<list>&final,
       list<Literal>solucion) {
       if (Instanciadas.empty())
              {final.push_back(solucion)
              return 1;}
       else{
              list<Literal>hijos =
              ListaHijos(Instanciadas.front(),solucion,es
              _unico);
       Instanciadas.pop_front();                for
       (c/ literal L ∈ hijos)
              {
                     if (es_unico)
                            hay_solucion= Back(Instanciadas,
                            final,solucion);
                     else{
                     solucion.push_back(L);
                     if (Back(Instanciadas,final,
                                   solucion))
                                   hay_solucion=1;
                            solucion.pop_back();
                     }
```

[2] Fórmula atómica con el signo opuesto.

```
                    returnhay_solucion;
                    }
                }
            }
}
```

Con la intención de que la herramienta sea de utilidad y didáctica para los alumnos que estudien el tema, se implementó un algoritmo que muestre lo más claro posible el procedimiento que uno realiza de forma manual para resolver este tipo de problemas.

En primera instancia, se genera el conjunto S, es decir, el conjunto de cláusulas definidas,seguidas por el Universo y Base de Herbrand respetando su definición y expresión formal. Cabe destacar que en el Universo y la Base cuando existen funciones se convierten en un conjunto infinito por lo que para indicarlo se agregan puntos suspensivos a la expresión. Esta última regla es también utilizada en las demás ocasiones donde el conjunto no sea finito, como es el caso de las instanciaciones y los posibles modelos de las cláusulas.

Luego de expresar la suposición de que existe un modelo de Herbrand que satisface cada una de las cláusulas, se busca probar la existencia de éste. Para hacerlo se realiza la selección cláusula por cláusula de literales ya instanciados, evitando contradicción con aquellos que pertenezcan a la solución parcial. En caso de insatisfacibilidad, se realiza un procedimiento similar, utilizando los números de pasos para mostrar la contradicción que lleva a tal resultado.

En el momento de la implementación, se buscó que fuera lo suficientemente flexible para que se adapte a cualquier entrada posible. Detalles que resultan intuitivos durante el estudio de Modelos de Herbrand fueron tratados con sumo cuidado en dicho algoritmo. Por ejemplo, lograr que la herramienta pueda diferenciar cuando una solución es un conjunto finito o infinito es una tarea sencilla, sin embargo, algorítmicamente requiere un gran análisis de los datos ingresados en cada modelo. Otra característica agregada fue que, para mostrar el seguimiento se enumeren las cláusulas que deben tomarse de modo tal que, cuando un camino genera una contradicción se vea fácilmente el motivo por el cual se ha descartado. Cuando se trabaja con un modelo insatisfacible, es decir que no se puede hallar una solución se muestra solo un camino que muestre la inconsistencia. Esta idea se consideró por el simple hecho de que no es posible mostrar en una única imagen todas las ramas que el algoritmo de Backtraking crea para garantizar que no hay solución.

3. Interfaz

Para la interfaz se utilizó el entorno de desarrollo Qt [4] [5] [9]. Este framework resultó sumamente útil, ya que no solo cuenta con una interfaz sencilla para crear aplicaciones, sino que también tiene un gran número de librerías y documentación para comenzar a crearlas. Al ser extenso en funcionalidad nos permitió mejorar la aplicación, como por ejemplo al permitir guardar conjuntos de cláusulas (lo que puede resultar tedioso cuando se quieren reingresar varias) o resaltar ciertas partes del

procedimiento mostrado para que resulte más claro al usuario. Afortunadamente, no se requirió modificar los widgets provistos por Qt, ya que los mismos fueron suficientes para realizar la aplicación.

El ingreso de cláusulas y constantes se realiza a través de dos campos de texto. Para el **ingreso de constantes** se aceptan palabras o letras que contengan mayúsculas y/o minúsculas sin acentos. Toda cadena de caracteres que contenga espacios, símbolos, etc. será rechazada.

Con respecto al **ingreso de cláusulas**, se omite el símbolo de cuantificador universal ∀. Dentro de los literales son aceptadas constantes, variables (que respetan el mismo formato que una constante) y funciones. Es requisito escribir correctamente las cláusulas para el correcto funcionamiento de la herramienta, ya sea mediante el número debido de paréntesis, comas, etc. hasta respetando la aridad de los literales y funciones. El ingreso de ambos tipos de datos se realiza a través de una interfaz como la mostrada en la Figura 4.

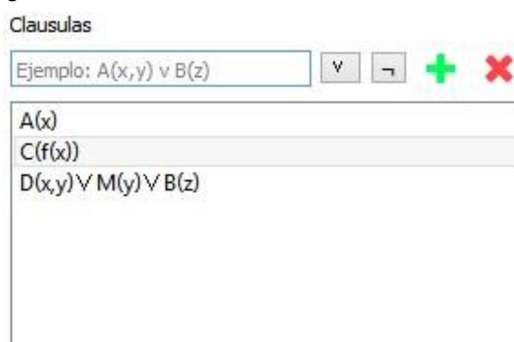

Figura 4: Interfaz para el ingreso de cláusulas

En el ejemplo de la figura 4, el conjunto de cláusulas ingresado corresponde a la fórmula:

$\forall x A(x) \wedge \forall x C(f(x)) \wedge \forall x \forall y \forall z (D(x, y) \vee M(y) \vee B(z))$

De este modo, se simplifica el trabajo del usuario, ingresando sólo las cláusulas por separado, ignorando el conector lógico *and*. Al mismo tiempo cada cláusula puede contener uno o más literales, que deben estar separados obligatoriamente por el símbolo *or* (∨), dado que la herramienta sólo acepta la forma clausular. Por otra parte, los literales pueden (o no) estar negados y se conforman por funciones o variables sin limitar su aridad.

Se realiza un control sobre los datos ingresados (tanto constantes como cláusulas), asegurando que cumplan el formato de la LdPPO. Tanto estos errores, como aquellos que contradicen de alguna manera lo ingresado, se muestran en una línea de texto. Cada error presenta un mensaje y un número, el cual es útil para buscar información más detallada en la ayuda en caso de ser necesario. Esta ayuda posee información sobre diversos aspectos de la herramienta, como el ingreso de los datos y la

presentación de la solución. En el caso de los errores se explica más detalladamente la causa de cada uno y se dan ejemplos de casos en donde pueden ocurrir.

Además de determinar e informar si un conjunto de cláusulas es satisfacible, la herramienta muestra el procedimiento mediante el cual se obtuvo tal respuesta. Si existen Modelos de Herbrand, se permite elegir uno de los modelos y la resolución se ajusta a tal, mostrando el procedimiento para obtenerlo (véase Figura 5). A cada literal del modelo se le asigna un número a medida que se procede. En caso de insatisfacibilidad, se muestra uno de los caminos en los que se encuentra contradicción, haciendo uso de estos números.

Otra característica útil de la herramienta es la posibilidad de guardar y cargar conjuntos de constantes y cláusulas ingresados. De esta manera, se evita tener que repetir el trabajo, que dependiendo el caso puede resultar tedioso, de reingresar todos estos datos cada vez que se quiera trabajar con un conjunto dado.

Figura 5: Interfaz con los posibles modelos y sus procedimientos

4. Conclusión

La herramienta THeM fue diseñada tanto para resolver el problema de encontrar modelos de Herbrand para un conjunto de cláusulas de forma eficiente, como también para acompañar el aprendizaje del tema. Como fue explicado, el encontrar un Modelo de Herbrand para un conjunto de cláusulas dado nos permite determinar, dependiendo de cómo se llegó a las mismas, ciertos aspectos sobre éstas. Ya sea la correctitud de un razonamiento, la validez de una fórmula, entre otros.

Las clases creadas nos permiten representar la jerarquía de los elementos de la LdPPO con facilidad, lo que brindó claridad y prolijidad al diseño. Asimismo, esta identificación representativa de los datos hace al código prolijo y entendible, y como consecuencia, genera un impacto positivo en el momento de su análisis.

Se mejora la eficiencia al ordenar la lista de cláusulas, para reducir el espacio de búsqueda del algoritmo de Backtracking. En éste no se genera un espacio de búsqueda aleatorio a partir del conjunto de cláusulas instanciadas, sino que se tienen en cuenta varios aspectos además del orden para crearlo. Se analiza la presencia de funciones, el estado de la solución parcial, entre otros.

Desde nuestro punto de vista, se logra una aplicación didáctica cuando usarla resulta sencillo e intuitivo. Este es el motivo por el cual decidimos crear una interfaz simple con ejemplos para que el usuario (el alumno) entienda de antemano cómo ingresar cláusulas o constantes. Para reforzar esto implementamos los mensajes de error, para ayudar al usuario a descubrir en qué está fallando. Incluso se provee un número de error para que se busque en el manual incluido, en donde se explica en detalle la causa del mismo. Además, se muestra el procedimiento seguido para determinar la satisfacibilidad del conjunto de cláusulas, lo cual resulta sumamente útil a la hora de aprender el tema.

La carencia de software que realice el mismo trabajo aumenta la potencial utilidad de la herramienta. No obstante, hubiera sido interesante comparar con otras implementaciones y así, poder agregar mejoras que hasta el momento no se han podido detectar.

Por otra parte, creemos que la herramienta ha cumplido los objetivos planteados en un principio debido a que ya es utilizada por los nuevos alumnos del curso como ayuda para entender y aprender a resolver modelos de Herbrand. Sería interesante realizar una formalización de este suceso como posible extensión al trabajo.

Otro objetivo subyacente que creemos importante para resaltar es que durante todo el proyecto se buscó crear soluciones que respeten las buenas prácticas de la programación y no se debe perder de vista que todo el proceso se realizó por alumnos de un segundo año de la carrera.

5. Referencias

1. ARENAS, Marcelo. *Lógica de Primer Orden*, Universidad Católica de Chile.
2. BEN-ARI, Mordechai. Mathematical Logic for Computer Science. ThirdEdition, 2003.
3. CUENA, José. *Lógica informática*. Alianza Madrid. 1985
4. DigiaPlc. *QtDocumentation*. http://doc.qt.io
5. The c++ Resource Network http://www.cplusplus.com/
6. JIMÉNEZ, José. FRANCO, Andrés. HIDALGO DOBLADO, María. *Lógica informática: Modelos de Herbrand*, Universidad de Sevilla. 2012.
7. MAUCO, Virginia. *Filminas de cátedra Ciencias de la Computación II: Modelos de Herbrand*, http://ccomp2.alumnos.exa.unicen.edu.ar/filminas.

8. MOONEY, Raymon. *First Order Predicate Logic*, The University of Texas at Austin. 2010
9. SGI: Silicon Graphics International. *Introduction to the Standard Template Library*http://www.sgi.com/tech/stl
10. Cormen, T.; Lieserson, C.; Rivest, R. *Introduction to Algorithms*. Ed. The MIT Press. 2009.
11. Aho, A. Ullman, J. *Foundations of Computer Science*. C Edition. Computer Science Press. 1995.