ISSN 1870-4069

Sobre la automatización del razonamiento y el aprendizaje en lógicas modales

Yensen Limón[1], Everardo Bárcenas[2],
Guillermo Molero-Castillo[2], Rocío Aldeco-Pérez[2]

[1] Universidad Veracruzana,
México
[2] Universidad Nacional Autónoma de México,
México

zs16017367@estudiantes.uv.mx,
{barcenas,gmoleroca,raldeco}@fi-b.unam.mx

Resumen. Las lógicas modales forman una familia de lenguajes formales ampliamente utilizados en diversos dominios de las ciencias de la computación, tales como la verificación de sistemas, la web semántica, y la demostración automática de teoremas. El Aprendizaje Automatizado es una rama de la Inteligencia Artificial que versa sobre el estudio y desarrollo de algoritmos capaces de aprender la solución de problemas a partir de un conjunto de ejemplos. Recientemente, se han publicado estudios sobre la aplicación de estas técnicas en la optimización de los algoritmos de razonamiento en lógicas clásicas de alto orden. Motivados por estos resultados, se propone en este artículo el estudio de técnicas de aprendizaje automatizado en el contexto del razonamiento automatizado en lógicas modales. De manera particular, en este trabajo se presenta un algoritmo de satisfacibilidad para el cálculo μ y se describe como un algoritmo de aprendizaje por refuerzo puede ayudar en el desempeño del algoritmo de satisfacibilidad.

Palabras clave: Razonamiento automatizado, aprendizaje automatizado, lógicas modales.

On Automated Reasoning and Machine Learning in Modal Logics

Abstract. Modal logics are a family of formal languages widely used in many domains of computer science. Machine Learning is a branch of Artificial Intelligence that deals with the study and development of algorithms capable of learning problem solving from a set of examples. Recent studies have been published on the application of these techniques in the optimization of reasoning algorithms in classical higher-order logics. Motivated with these results, the study of machine learning techniques in the context of automated reasoning in modal logics is proposed in this

article. In particular, in this work we describe a satisfiability algorithm for the μ-calculus and describes how a reinforcement learning algorithm can help in the performance of the satisfiability algorithm.

Keywords: Automated reasoning, machine learning, modal logics.

1. Introducción

Las lógicas modales (ML), cuyo origen se remonta a la lógica aristotélica [4], forman una familia de lenguajes formales que incluyen la Lógica Temporal Lineal (LTL), la Lógica Temporal Ramificada (CTL), la Lógica Dinámica Proposicional (PDL), el Cálculo μ, las Lógicas Descriptivas, por mencionar algunos. Además del campo de la filosofía, las ML han encontrado un dominio de estudio muy amplio, en campos como las matemáticas, las humanidades y las ciencias computacionales. Dentro de este último ámbito, un tema largo y exitosamente estudiado es el desarrollo de métodos automáticos de razonamiento. El estudio de estos métodos, en particular algoritmos, es el principal tema de interés de este proyecto [13].

El Teorema de Correspondencia de van Benthem (1977) versa sobre la caracterización de la Lógica Modal K en términos del fragmento invariante bajo bisimulación de la Lógica Clásica de Predicados [3]. Caracterizaciones similares fueron posteriormente establecidas entre otras ML y contrapartes clásicas, por ejemplo, entre el Cálculo μ y la Lógica Monádica de Segundo Orden [9]. Estas caracterizaciones fundamentan lo que en la práctica ha resultado en el particular balance entre expresividad y complejidad computacional de los algoritmos de razonamiento asociados a las ML. Abundantes problemas en diversas áreas, tales como la Verificación Formal de Sistemas (*software* y *hardware*), Bioinformática, y Demostración Automática de Teoremas, por mencionar algunas, han sido modelados y solucionados de forma automática gracias al citado balance de las ML [3]. En este proyecto, estamos interesados en el estudio de técnicas de Aprendizaje Automatizado (*machine learning*) que puedan ser utilizadas en la optimización de algoritmos de razonamiento para ML.

El Aprendizaje Automatizado (AA) es una rama de la Inteligencia Artificial ampliamente estudiada en la actualidad y con muy variadas aplicaciones tales como en Visión Computacional, Robótica, Ciencia de Datos, etc [1]. El AA se basa, *grosso modo*, en el estudio y desarrollo de algoritmos capaces de aprender la solución de problemas a partir de un conjunto de ejemplos. Recientemente, se han publicado estudios sobre la aplicación de estas técnicas en la optimización de los algoritmos de razonamiento en lógicas clásicas de alto orden [1]. El conjunto de ejemplos estudiados en este caso han sido demostraciones matemáticas. Motivados por estos resultados, proponemos en este trabajo el estudio de técnicas de AA en el contexto del razonamiento automatizado en ML.

2. Motivación y antecedentes

En esta sección se describen los trabajos relacionados y se justifica la propuesta de este trabajo.

2.1. Demostración automática de teoremas y aprendizaje automatizado

En [1] se describe un demostrador automático de teoremas, basado en lógica de alto orden (HOL Light), optimizado con técnicas de aprendizaje profundo. Este demostrador se utilizó para demostrar la conjetura de Kepler. Existen diversos trabajos relacionados, tal como la herramienta TitacToe, basada en un algoritmo A* y lógica de alto orden (HOL4), pero la diferencia es que no emplea aprendizaje profundo [7]. Otro trabajo similar se reporta en [8], en el cual se describe la herramienta Gamepad. Esta herramienta también está basada en Hol Light, sin embargo, el conjunto de entrenamiento solo cuenta con 1602 teoremas y lemas. Por otra parte, en la herramienta reportada en [1], se consideran 2199 lemas y 29462 teoremas como conjunto de aprendizaje.

Adicionalmente, el entrenamiento puede considerar pruebas humanas. Estas pruebas se registran y son utilizadas como ejemplos de entrenamiento. Cuando es una demostración exitosa HOL Light registra el resultado de las aplicaciones tácticas utilizadas y el teorema para ser utilizado en un futuro. En este trabajo se propone aplicar técnicas de aprendizaje automático en algoritmos de razonamiento basados en lógicas modales. Esto implica una extensión en el dominio de aplicación de este tipo de algoritmos, tales como la verificación de sistemas.

2.2. Satisfacibilidad

Una expresión o fórmula bien formada en un sistema se dice satisfacible si y sólo si existe un modelo para dicha fórmula, es decir, si existe una interpretación que hace verdadera a la fórmula. Se dice que una fórmula es válida si y sólo es verdadera en cualquier interpretación. Nótese que el problema de validez es equivalente al de satisfacibilidad: una fórmula es válida, si y solo si, la negación de la fórmula no es satisfacible.

En el contexto de las lógicas modales un modelo puede entenderse como una estructura de Kripke [3]. Una estructura de Kripke se define como una tripleta $K = \langle D, R, L \rangle$ donde D es un conjunto no-vacío (usualmente de mundos posibles o estados), $R \subseteq D^2$ es una relación entre los miembros de D (usualmente conocida como relación de acceso) y L es una función que asigna valores a las variables del lenguaje base. Las familias particulares de relaciones en R están asociadas a ciertas lógicas, por ejemplo, en CTL las relaciones en R forman estructuras de árbol. Los algoritmos de satisfacibilidad tipo Fischer-Ladner están basado en la construcción explicita de modelos [6]. En los modelos construidos por estos algoritmos los elementos del dominio están definidos como conjuntos de subfórmulas (de la fórmula a la cual se le quiere determinar su satisfacibilidad o insatisfacibilidad).

Las variables proposicionales en estos conjuntos representan las asignaciones por la función L (de la estructura de Kripke), y las fórmulas modales representan la topología del modelo, es decir, si en un elemento del dominio está la fórmula $\langle m \rangle p$, entonces debe existir otro elemento del dominio, conectado a través de m, que contenga p.

A menudo, la complejidad computacional de los algoritmos Fischer-Ladner está determinada por la cantidad elementos del dominio candidatos a formar un modelo [14]. Por ejemplo, para el cálculo μ existe una cantidad exponencial, con respecto al tamaño de la fórmula inicial, de elementos del dominio candidatos a formar un modelo (recuerde que se definen como conjuntos de subfórmula inicial) [13]. Como la construcción de modelos implica una búsqueda en esta cantidad exponencial de elementos del dominio, la complejidad computacional del cálculo μ está en EXPTIME-completa [14,13]. Se propone en este trabajo que un algoritmo de satisfacifiblidad tipo Fischer-Ladner que elija los elementos del dominio a partir de un conjunto de modelos de aprendizaje. Esto con el objeto de optimizar el desempeño del algoritmos ya conocidos [14,13].

2.3. Verificación de sistemas

Un ejemplo sobre la aplicación en la automatización del razonamiento utilizando lógicas modales es la verificación de sistemas conscientes de contexto [11,12]. Este tipo de sistemas son capaces de sensar su contexto y reaccionar de acuerdo a la información obtenida. La gran cantidad de variables de contexto hacen que el desarrollo y mantenimiento de este tipo de sistemas sean costosos y proclives a errores.

En [10] se introduce una noción de consistencia en sistemas conscientes de contexto. La verificación de consistencia se basa en un algoritmo de satisfacibilidad tipo Fischer-Ladner del cálculo μ. Además, se describe en [11] la aplicación de este algoritmo en la verificación de un sistema de comunicación consciente de contexto. Este tipo de sistemas están compuestos por un conjunto de usuarios, ubicados en lugares, con diversas restricciones de comunicación (por ejemplo, sin señal de teléfono móvil), de acuerdo a un horario. Los canales de comunicación pueden ser síncronos (teléfono móvil) o asíncronos (correo electrónico). Este tipo de sistemas se definen consistentes en caso de que los usuarios se puedan comunicar siempre entre sí, sin importar las restricciones descritas.

Otro ejemplo de verificación de sistemas conscientes de contexto es el que se describe en [12]. En este artículo se presenta un ambiente de aprendizaje inteligente, el cual está compuesto por un conjunto de actividades de aprendizaje a realizarse en diversas ubicaciones equipadas con posiblemente distintos dispositivos (computadoras, proyectores, sensores, etc.). A través de diversos sensores el sistema es capaz de identificar a usuarios desatentos a las actividades de aprendizaje, y de acuerdo a las restricciones antes descritas, aunadas a las restricciones de aprendizaje de los estudiantes, el sistema interviene para atraer la atención del estudiante (videos, textos, avisos al instructor, etc.). El sistema se define consistente cuando es capaz de realizar una intervención sin importar las restricciones antes descritas.

La verificación de consistencia de estos sistemas también se basa en un algoritmo de satisfacibilidad tipo Fischer-Ladner del cálculo μ.

En ambos trabajos [11,12] también se describen diversos experimentos. Sin embargo, debido a la complejidad exponencial computacional del algoritmo de satisfacibilidad, los casos de experimentación fueron modestos. Se conjetura que una optimización del algoritmo de satisfacibilidad con técnicas de aprendizaje automatizado puede conllevar a experimentos más realistas.

2.4. Razonamiento sobre consultas

Dentro del análisis estático de lenguajes de programación, en particular, los lenguajes de consultas para bases de datos, existen diversos problemas de razonamiento. El problema de vacuidad de consultas consiste en determinar si el resultado de cierta consulta es siempre vacío, sin importar la base de datos. Determinar si el resultado de cierta consulta siempre se encuentra dentro del resultado de otra consulta, sin importar la base de datos, se le conoce como el problema de contención. La equivalencia de dos consultas se define en términos de contención: dos consultas son equivalentes, si y solo si, están contenidas entre sí. En el contexto de datos semi-estructurados, más concretamente, del lenguaje de consultas *XPath*, estos problemas de razonamiento son modelados en términos de satisfacibilidad en el cálculo μ en [14,13]. El algoritmo es tipo Fischer-Ladner. Diversos experimentos fueron también descritos en estos artículos. Motivados por experimentos más extensos, conjeturamos que optimizaciones, basadas en aprendizaje automatizado, al algoritmo permitirán su aplicación en escenarios más complejos.

3. Cálculo μ en árboles

En esta sección, se presenta el cálculo μ. Las fórmulas son interpretadas sobre árboles finitos. El alfabeto es considerado por dos conjuntos, $PROP$ y MOD, donde $PROP$ es un conjunto de letras proposicionales y $MOD = \{1, 2, 3, 4\}$ es un conjunto de modalidades.

Definición 1 (Sintaxis) *El conjunto de fórmulas del cálculo μ es definido por la siguiente gramática: $\varphi ::= p \mid X \mid \neg\varphi \mid \varphi \vee \psi \mid \langle m\rangle\varphi \mid \mu X.\varphi$. donde p es una proposición, m es una modalidad, y X es una variable.*

Se asume que las variables únicamente ocurren sobre el alcance de una modalidad. Las fórmulas son interpretadas como un subconjunto de nodos en árboles. Las proposiciones son utilizadas para etiquetar los nodos, la negación ($\neg$) es interpretada como el conjunto complemento, la disyunciones son interpretadas como la unión de conjuntos. En el caso de $\phi \wedge \varphi$ se escribe $\neg(\neg\phi \vee \neg\varphi)$. Las fórmulas modales $\langle m\rangle\phi$ se mantienen en nodos donde hay un nodo adyacente por m que soporta ϕ. El punto menos fijo es intuitivamente interpretado como un operador de recursión. Ahora se presenta la definición de las estructuras estilo Kripke.

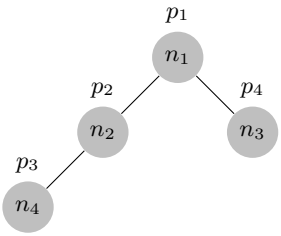

Fig. 1. Ejemplo de un modelo de árbol, donde $\phi = \mu X.(p_3 \vee \langle 1 \rangle X)$, entonces $n_1 = \{\phi, \langle 1 \rangle p_2, p_1\}$, $n_2 = \{\phi, p_2\}$, $n_3 = \{p_4\}$ y $n_4 = \{\phi, p_3\}$.

Definición 2 (Estructura de árbol) *Una estructura de árbol, o simple árbol, es una tupla* $\langle N, R^m, L \rangle$ *donde:*

- N *es el conjunto de nodos;*
- R^m *es una familia binaria de relaciones de nodos* $(N \times N)$ *formando una estructura de árbol; y*
- L *es una función de etiquetado* $L : N \to 2^{PROP}$.

Ahora se da una representación formal de la semántica de las fórmulas.

Definición 3 (Semántica) *Considere un árbol* K *y una valuación* $V : Var \to 2^N$, *donde* Var *es un conjunto de variables. La semántica de la fórmula es definida de la siguiente manera:*

$$
\begin{aligned}
[\![p]\!]_V^K &= \{n \mid p \in L(n)\} &,\\
[\![\neg\varphi]\!]_V^K &= N \backslash [\![\varphi]\!]_V^K &,\\
[\![\varphi \vee \psi]\!]_V^K &= [\![\varphi]\!]_V^K \cup [\![\psi]\!]_V^K &,\\
[\![\langle m \rangle \varphi]\!]_V^K &= \{n \mid R(n,m) \cap [\![\varphi]\!]_V^K \neq \emptyset\} &,\\
[\![X]\!]_V^K &= V(X) &,\\
[\![\mu X.\varphi]\!]_V^K &= \bigcap \{N' \mid [\![\varphi]\!]_{V[^{\mu X.\varphi}/_X]}^K \subseteq N'\} &.
\end{aligned}
$$

Una fórmula ϕ es satisfacible, si y solo si, existe un árbol (modelo), tal que la interpretación de ϕ sobre el árbol no sea vacía, esto es $[\![\phi]\!]_V^K \neq \emptyset$.

Ejemplo 1 *En la Figura 1, existe una representación gráfica del modelo de árbol. Considere por ejemplo la siguiente fórmula:* $\langle 1 \rangle p_2 \wedge p_1$. *Esta fórmula selecciona los nodos nombrados* p_1 *con un hijo llamado* p_2. *En la Figura 1, esta fórmula es soportada en* n_1. *La navegación recursiva puede ser expresado con el punto fijo. Esta fórmula* $\mu X.(p_3 \vee \langle 1 \rangle X) \wedge \langle 1 \rangle p_2 \wedge p_1$ *mantiene en un nodo a* p_1 *con un descendiente* p_3 *en el nodo* n_4 *y un hijo* p_2 *en el nodo* n_2. *En la Figura 1, la fórmula es soportada por el nodo* n_1.

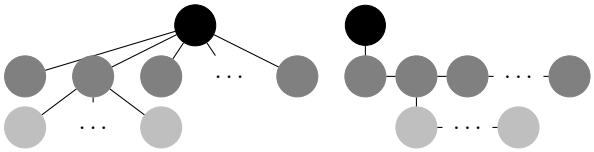

Fig. 2. Ejemplo de transformación de árbol de n-aridad a binario [2].

4. Satisfacibilidad y aprendizaje por refuerzo

El algoritmo de satisfacibilidad construye una versión sintáctica de una estructura de árbol, llamado árbol Fischer-Ladner [6], donde los nodos son conjuntos de subfórmulas.Ahora se define la noción de árboles. Es conocida la biyección entre árboles de n-aridad y binarios: Una relación es para el primer hijo(padre), y la otra para el siguiente hermano (previo), En la Figura 2, existe una representación gráfica de esta biyección. Por lo tanto, sin perder la generalidad, por conveniencia técnica, se trabaja con árboles binarios. Además, las fórmulas modales deben ser reinterpretadas:

- $\langle 1\rangle\varphi$ se mantiene en nodos donde su primer hijo es soportado por φ;
- $\langle 2\rangle\varphi$ es interpretada en nodos donde tiene en su siguiente hijo (derecha) a φ;
- $\langle 3\rangle\varphi$ se encuentra donde φ es el padre; y
- $\langle 4\rangle\varphi$ donde φ es el hermano previo (derecha).

Ahora consideramos fórmulas únicamente en forma normal negada. La negación solo ocurre en frente de proposiciones y fórmulas modalidades $\langle m\rangle\top$, donde para $\top$ se presenta por $p \vee \neg p$. La función $nnf(\phi)$ se define a continuación:

$$\begin{array}{rr} nnf(X) = \neg X & nnf(\varphi \vee \psi) = nnf(\varphi) \wedge nnf(\psi), \\ nnf(p) = \neg p & nnf(\mu X.\varphi) = \mu x.nnf(\varphi)\left[{}^{X}/_{\neg X}\right], \\ & nnf(\langle m\rangle\varphi) = \langle m\rangle nnf(\varphi) \vee \neg\langle m\rangle\top. \end{array}$$

La fórmula en forma normal negada se define como la fórmula resultante de remplazar las negaciones $\neg\varphi$ con $nnf(\varphi)$. Algunas nociones son requeridas antes de introducir los árboles Fischer-Ladner trees.

Definición 4 *Se define una relación binaria R^{FL} en fórmulas con $i = 1, 2$:*

$$\begin{array}{lll} R^{FL}(\varphi, nnf(\varphi)) & R^{FL}(\varphi_1 \wedge \varphi_2, \varphi_i) & R^{FL}(\varphi_1 \vee \varphi_2, \varphi_i), \\ R^{FL}(\langle m\rangle\varphi, \varphi) & R^{FL}(\mu X.\varphi, \varphi[{}^{\mu x.\varphi}/_{X}]). & \end{array}$$

Definición 5 (Fischer-Ladner Closure) *Dada una fórmula φ, Fischer-Ladner closure de φ es definida como $FL(\varphi) = FL(\varphi)_k$, Donde k es el más pequeño entero positivo que satisface $FL(\varphi)_k = FL(\varphi)_{k+1}$, donde:*

$$\begin{array}{l} FL(\varphi)_0 = \{\varphi\}, \\ FL(\varphi)_{i+1} = FL(\varphi)_i \cup \{\psi' \mid R^{FL}(\psi, \psi'), \psi \in FL(\varphi)_i\}. \end{array}$$

A continuación, se define el conjunto lean para los nodos sintácticos. Este conjunto está compuesto intuitivamente por las proposiciones y subfórmulas modales obtenidas de la fórmula de entrada. Las proposiciones son utilizadas para nombrar los nodos y las subfórmulas modales dan la topología de la información sobre los árboles candidatos.

Definición 6 (Lean) *Dada una fórmula φ y una proposición p' no ocurrente en φ, el lean de φ es definido de la siguiente manera para todas $m \in MOD$:*

$$lean(\varphi) = \{p, \langle m \rangle \varphi \in FL(\varphi)\} \cup \{p', \langle m \rangle \top\}$$

.

Ejemplo 2 *Considere la siguiente fórmula: $\varphi = \langle 1 \rangle \langle 1 \rangle \neg p_1 \wedge \langle 1 \rangle \neg p_1 \wedge \neg p_1 \wedge \mu X.(p_1 \vee \langle 1 \rangle X) \wedge \langle 2 \rangle p_2 \wedge p_3$. El lean de φ es definido como:*

$$lean(\varphi) = \{p_1, p_2, p_3, \langle 1 \rangle \neg p_1, \langle 1 \rangle \langle 1 \rangle \neg p_1, \langle 1 \rangle \mu X.(p_1 \vee \langle 1 \rangle X), p', \langle 2 \rangle p_2, \langle 1 \rangle \top, \langle 2 \rangle \top, \langle 3 \rangle \top, \langle 4 \rangle \top\}.$$

Ahora se definen los nodos en árboles Fischer-Ladner.

Definición 7 (Nodos) *Dada una fórmula φ, un nodo en un árbol Fischer-Ladner es definido como un subconjunto de $lean(\varphi)$, es decir;*

- *contiene al menos una proposición;*
- *si $\langle m \rangle \psi$ ocurre en el nodo, también $\langle m \rangle \top$; y*
- *$\langle 3 \rangle \top$ y $\langle 4 \rangle \top$ ambas no pueden ocurrir.*

Ejemplo 3 *Considerando la misma fórmula y el correspondiente lean presentado en el Ejemplo 2. Entonces se determinan los siguientes nodos:*

- $n_1 : \{p_3, \langle 1 \rangle \mu X.(p_1 \vee \langle 1 \rangle X), \langle 1 \rangle \neg p_1, \langle 1 \rangle \langle 1 \rangle \neg p_1, \langle 2 \rangle p_2\}$;
- $n_2 : \{p', \langle 1 \rangle \neg p_1, \langle 1 \rangle \mu X.(p_1 \vee \langle 1 \rangle X)\}$;
- $n_3 : \{p', \langle 1 \rangle \mu X.(p_1 \vee \langle 1 \rangle X)\}$;
- $n_4 : \{p_1\}$;
- $n_5 : \{p_2\}$.

Finalmente definimos los árboles Fishcer-Ladner.

Definición 8 (Árbol Fischer-Ladner) *Intuitivamente se definen los árboles Fischer-Lader como:*

- *$\emptyset$ representa un árbol Fischer-Ladner vacío ; y*
- *(n, X_1, X_2) es un árbol Fischer-Ladner, dado que n es un nodo (nodo raíz), , X_1 y X_2 son árboles Fischer-Ladner.*

Para aclarar el contexto, nos referiremos como simplemente árbol en lugar de árbol Fischer-Ladner.

Ejemplo 4 *Considere los nodos en el Ejemplo 3. Entonces definimos el árbol como:* $T = (n_1, (n_2, (n_3, (n_4, \emptyset, \emptyset), \emptyset), \emptyset), (n_5, \emptyset, \emptyset))$. *En la Figura 3 se presenta el árbol.*

De esta forma se describe el algoritmo de satisfacibilidad del cálculo μ basado en una búsqueda primero en profundidad. El Algoritmo 1 decide si una fórmula es satisfacible o insatisfacible. Dentro del algoritmo se cuenta con una función *main* la cual es encargada de crear todos los nodos que se utilizarán. Después, los nodos son iterados y busca el nodo que satisfaga la fórmula. Como consecuencia inicia una construcción del árbol para satisfacer la fórmula. El nodo seleccionado es utilizado como candidato y es a partir de él donde comienza la búsqueda. Posteriormente, el siguiente paso del algoritmo es entrar a la función *search*. En el Algoritmo 1 la función *search* presenta diversos casos, cuando la fórmula tiene la forma de $p, \top, \neg p$ o $\neg\langle m\rangle\top$, entonces, el algoritmo evalúa la satisfacibilidad del nodo retornando *true*. Si esta contiene puntos fijos, entonces, el algoritmo realiza la expansión de la fórmula llamando a la función *search* nuevamente. Los casos para la disyunción y conjunción el algoritmo invoca la función *search* para ambos. Cuando las fórmulas tienen la forma de modalidad, este busca el siguiente nodo. La fórmula puede contener modalidades $\langle 1\rangle$, $\langle 2\rangle$, $\langle 3\rangle$ o $\langle 4\rangle$. Entonces el nodo candidato es eliminado de la lista de nodos, el algoritmo obtiene la lista de los posibles nodos siguientes con la función *next*, posteriormente la lista de nodos es iterada y el árbol es actualizado. Después es llamada la función *search*, si este retorna $false$, entonces, el nodo siguiente es removido del árbol. Cada vez que un nodo es agregado, este pasa a un estatus como utilizado y no puede ser reutilizado en niveles subsecuentes del árbol, esto es con el fin de no construir árboles infinitos. En el caso que los nodos se terminen en la búsqueda, entonces el algoritmo retorna $false$. El algoritmo termina cuando ha viajado por todos los nodos o el árbol es construido.

La satisfacibilidad en fórmulas se define a continuación.

Definición 9 *Dada una fórmula* ϕ *y un nodo* n*, se define la satisfacibilidad* $n \vdash \phi$ *como:*

$$\frac{}{n \vdash \top} \qquad \frac{\varphi \in n}{n \vdash \varphi} \qquad \frac{\varphi \notin n}{n \vdash \neg\varphi} \qquad \frac{n \vdash \psi}{n \vdash \varphi \vee \psi} \qquad \frac{n \vdash \varphi}{n \vdash \varphi \vee \psi}$$

$$\frac{n \vdash \varphi \quad n \vdash \psi}{n \vdash \varphi \wedge \psi} \qquad\qquad \frac{n \vdash \varphi[{}^{\mu x.\varphi}/_{x}]}{n \vdash \mu x.\varphi}$$

La función Δ_m es responsable de verificar la unión de un nodo respecto a otro nodo a través de una modalidad m. Cuando n' es el nodo posible a unir, n_1 es el nodo actual, m son las modalidades $\langle 1\rangle$ o $\langle 2\rangle$, $\bar{m}$ son las modalidades inversas $\langle 3\rangle$ o $\langle 4\rangle$ y $\langle m\rangle\varphi$ una fórmula modal.

Definición 10 *Dado dos nodos* n*,* n' *y una fórmula* φ*, esta es determinada por aquellos nodos que son consistentes con respecto a la fórmula* $\Delta_m(n, n')$ *donde* $m \in \{1, 2, 3, 4\}$*, si y solo si, las fórmulas modales* $\langle m\rangle\varphi_1, \langle\bar{m}\rangle\varphi_2 \in lean(\varphi)$*, esto se determina de la siguiente manera:*

Algoritmo 1 Algoritmo de satisfacibilidad del cálculo μ con una búsqueda primero en profundidad sobre árboles.

```
function MAIN(φ)
    Y ← N^φ
    T ← ∅
    for all n_i of Y do
        if n_i ⊢ φ then
            T ← (n_i, ∅, ∅)
            if search(n_i, n_i, φ) then
                return true
            else
                T ← ∅
            end if
        end if
    end for
    return false
end function
```

- $\forall\langle m\rangle\varphi \in lean : \langle m\rangle\varphi \in n \Leftrightarrow n' \vdash \varphi$
- $\forall\langle \bar{m}\rangle\varphi \in lean : \langle \bar{m}\rangle\varphi \in n' \Leftrightarrow n \vdash \varphi$

Ejemplo 5 *Dada la fórmula* $\varphi = \langle 1\rangle\langle 1\rangle\neg p_1 \wedge \langle 1\rangle\neg p_1 \wedge \neg p_1 \wedge \mu X.(p_1 \vee \langle 1\rangle X) \wedge \langle 2\rangle p_2 \wedge p_3$ *reutilizando el Ejemplo 2 la unión es la siguiente:* $\Delta_m(n_1, n_2)$, $\Delta_m(n_1, n_5)$, $\Delta_m(n_2, n_3)$ *y* $\Delta_m(n_3, n_4)$. *Donde* $n_1 = \{p', \langle 1\rangle\mu X.(p_1 \vee \langle 1\rangle X), \langle 1\rangle\neg p_1, \langle 1\rangle\langle 1\rangle\neg p_1, \langle 2\rangle p_2, p_3\}$, $n_2 = \{p', \langle 1\rangle\neg p_1, \langle 1\rangle\mu X.(p_1 \vee \langle 1\rangle X)\}$, $n_3 = \{p', \langle 1\rangle\mu X.(p_1 \vee \langle 1\rangle X)\}$, $n_4 = \{p_1\}$ *y* $n_5 = \{p_2\}$. *Esos nodos son consistentes debido a que los nodos hijos contiene a los testigos de las fórmulas modales.*

Definición 11 (nodes) *Dado un árbol* T, *la función nodes es definida como:*

- $nodes(\emptyset) = \{\}$.
- $nodes((n, T_1, T_2)) = \{n\} \cup nodes(T_1) \cup nodes(T_2)$.

Un nodo T_n es denotado como:

- T_n es un $n \in nodes(T)$.

Definición 12 (Subárbol) *El subárbol* T' *de un árbol dado* T ($T' \subseteq T$) *es definido como:*

- $T \subseteq T$
- $T \subseteq (n, T_1, T_2)$ *si* $T \subseteq T_1$ *o* $T \subseteq T_2$

Definición 13 (root) *La función* $root(T)$ *toma el árbol como entrada y retorna el nodo raíz del árbol.*

- $root(\emptyset) = \emptyset$.

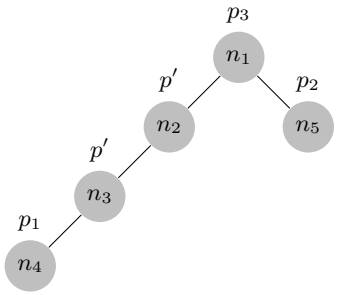

Fig. 3. Modelo de Árbol Fischer-Ladner para $\varphi = \langle 1\rangle\langle 1\rangle\neg p_1 \wedge \mu X.(p_1 \vee \langle 1\rangle X) \wedge \langle 1\rangle\neg p_1 \wedge \neg p_1 \wedge \langle 2\rangle p_2 \wedge p_3$, donde $n_1 = \{p_3, \langle 1\rangle\mu X.(p_1 \vee \langle 1\rangle X), \langle 1\rangle\neg p_1, \langle 1\rangle\langle 1\rangle\neg p_1, \langle 2\rangle p_2\}$, $n_2 = \{p', \langle 1\rangle\neg p_1, \langle 1\rangle\mu X.(p_1 \vee \langle 1\rangle X)\}$, $n_3 = \{p', \langle 1\rangle\mu X.(p_1 \vee \langle 1\rangle X)\}$, $n_4 = \{p_1\}$ y $n_5 = \{p_2\}$

- $root((n, T_1, T_2)) = n$.

Definición 14 (neighborhood) *Dado un árbol T y un nodo n, la función neighborhood obtiene los nodos adyacentes. Considerando que existen los árboles T', T_1 y T_2.*

- $parent(n, T) = n'$ *tal que,* $(n', T_n, T') \subseteq T$.
- $child(n, T) = n'$ *tal que,* $(n, (n', T_1, T_2), T') \subseteq T$.
- $ps(n, T) = n'$ *tal que,* $(n', T', T_n) \subseteq T$.
- $fs(n, T) = n'$ *tal que,* $(n, T', (n', T_1, T_2)) \subseteq T$.
- $neighborhood(n, T) = \{parent(n, T), child(n, T), ps(n, T), fs(n, T)\}$.

Definición 15 (delete) *Dado un árbol T y un nodo n que es eliminado del árbol, la función delete se define como:*

- $delete(\emptyset, n) = \emptyset$.
- $delete((n, T_1, T_2), n) = (\emptyset, T_1, T_2)$.
- $delete((n', T_1, T_2), n) = (n', delete(T_1, n), delete(T_2, n))$.

donde $n \neq n'$.

Definición 16 (update) *Dado un árbol T, dos nodos diferentes n y n' y una modalidad m, la fórmula update es definida como:*

- *cuando* $n' \notin nodes(T)$.
  - $update((n, T_1, T_2), n, n', 3) = (n', (n, T_1, T_2), \emptyset)$.
  - $update((n, T_1, T_2), n, n', 4) = (n', \emptyset, (n, T_1, T_2))$.
  - $update((n, \emptyset, T_2), n, n', 1) = (n, (n', \emptyset, \emptyset), T_2)$.
  - $update((n, T_1, \emptyset), n, n', 2) = (n, T_1, (n', \emptyset, \emptyset))$.
  - $update((n'', T_1, T_2), n, n', m) = (n''$, $update(T_1, n, n', m)$, $update(T_2, n, n', m))$.
  - $update(\emptyset, n, n', m) = \emptyset$.
- *cuando* $n' \in nodes(T)$.

– $update(T, n, n', m) = T$.

La función *next* obtiene el conjunto de nodos subsecuentes para identificar un nodo candidato. Esta función es definida por una tripleta de elementos $(n, \langle m \rangle \varphi, Y)$. Donde n es el nodo actual, $\langle m \rangle \varphi$ es una fórmula modal, Y es el conjunto de nodos no utilizados y T es el árbol.

Definición 17 (next) *Dado un nodo n, una fórmula modal $\langle m \rangle \phi$, un conjunto de nodos no utilizados Y, y un árbol T se define la función next para determinar la unión de los nodos que sean considerados modalmente consistente con respecto a la fórmula. Esta declaración es denotada por $\Delta_m(n, n')$, si y solo si, para todas las fórmula $\langle m \rangle \phi$ en n y $\langle \bar{m} \rangle \phi$ en n', determinadas por la dirección m. La función next es definida como:*

– $next(n, \langle m \rangle \varphi, Y, T) = \{n' \in Y \cup neighborhood(n, T) \mid \Delta_m(n, n')\}$, *donde* $m = 1, 2, 3, 4$.

Como puede observarse del algoritmo. Los nodos no son generados previamente a la construcción del modelo, si no que se generan al vuelo a través de la función *next*. Sin embargo esta función aún tiene que buscar en el conjunto completo de posibles nodos. A continuación describimos como a través de aprendizaje por refuerzo podemos determinar cuáles son los nodos *siguientes*. Para esto primero definimos que es un proceso de decisión de Markov.

Definición 18 (Proceso de Decisión de Markov) *Un Proceso de Decisión de Markov se define como un tupla (S, A, P, R), donde*

– *S es un conjunto de estados;*
– *A es un conjunto de acciones;*
– *P es una función de transición probabilistica entre los estados; y*
– *R es una función de recompensas de transición.*

Un proceso de decisión de Markov es construido a partir de un conjunto de entrenamiento. Las transiciones entre los estados se definen en función de la probabilidad de transición aprendida [5]. En el contexto del algoritmo de satisfacibilidad, el conjunto de aprendizaje está compuesto por modelos ya construidos, como los resultantes de la experimentación en [12,13,14].

5. Conclusiones

Las lógicas modales representan una familia de lenguajes formales con aplicación en un amplio dominio de campos en las ciencias de la computación, por ejemplo, en la verificación de sistemas. En este trabajo se describe un algoritmo de satisfacibilidad para una de las lógicas modales decidibles con mayor expresividad, el cálculo μ. Se propone además el uso de métodos de aprendizaje automatizado para optimizar el desempeño del algoritmo. De manera más específica, se propone la aplicación de aprendizaje por refuerzo en el proceso de

Algoritmo 1 Función Search del algoritmo de satisfacibilidad

```
function SEARCH(n_0, n_1, φ)
    if φ = p then
        if p ∈ n_1 then
            return true
        end if
    end if
    if φ = ¬p then
        if p ∉ n_1 then
            return true
        end if
    end if
    if φ = ¬⟨m⟩⊤ then
        if ⟨m⟩⊤ ∉ n_1 then
            return true
        end if
    end if
    if φ = ⊤ then
        return true
    end if
    if φ = φ_1 ∧ φ_2 then
        if search(n_0, n_1, φ_1) and
            search(n_0, n_1, φ_2) then
            return true
        end if
    end if
    if φ = φ_1 ∨ φ_2 then
        if search(n_0, n_1, φ_1) or
            search(n_0, n_1, φ_2) then
            return true
        end if
    end if
    if φ = ⟨m⟩φ then
        Y ← Y\{n_1}
        X ← next(n_1, ⟨m⟩φ, Y ∪ {n_0}, T)
        for all n_j of X do
            T ← update(T, n_1, n_j, m)
            if search(n_1, n_j, φ) then
                return true
            else
                T ← delete(T, n_j)
            end if
        end for
    end if
    if φ = μx.φ then
        return search(n_0, n_1, φ[μx.φ/x])
    end if
    return false
end function
```

búsqueda para la construcciones de modelos del cálculo μ. La implementación de esta propuesta es nuestro principal interés como trabajo a futuro.

Es también de nuestro interés la aplicación de esta propuesta en la verificación de sistemas [11].

Agradecimientos. Esta obra se financió con recursos del proyecto IA105420 del Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica (PAPIIT) de la UNAM.

Referencias

1. Bansal, K., Loos, S.M., Rabe, M.N., Szegedy, C., Wilcox, S.: Holist: An environment for machine learning of higher-order theorem proving (extended version). CoRR (2019)
2. Bárcenas, E., Lavalle, J.: Global numerical constraints on trees. Logical Methods in Computer Science 10(2), 1–28 (2014)
3. Blackburn, P., van Benthem, J.F., Wolter, F.: Handbook of modal logic, vol. 3. Elsevier (2006)
4. Castro Manzano, J.M., Medina Delgadillo, J.: Pons asinorum: una versión terminística. In: XVII Congreso latinoamericano de filosofía medieval, La edad media desde América latina: Aportes a la tradición. pp. 163–179 (2019)
5. Doberkat, E.: Stochastic relations. Foundations for Markov transition systems (01 2007)
6. Fischer, M.J., Ladner, R.E.: Propositional modal logic of programs (extended abstract). In: Hopcroft, J.E., Friedman, E.P., Harrison, M.A. (eds.) ACM Symposium on Theory of Computing. pp. 286–294. ACM (1977)
7. Gauthier, T., Kaliszyk, C., Urban, J.: TacticToe: Learning to reason with HOL4 tactics. arXiv:1804.00595 (2018)
8. Huang, D., Dhariwal, P., Song, D., Sutskever, I.: Gamepad: A learning environment for theorem proving (2018)
9. Janin, D., Walukiewicz, I.: On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In: Montanari, U., Sassone, V. (eds.) CONCUR. Lecture Notes in Computer Science, vol. 1119, pp. 263–277. Springer (1996)
10. Limón, Y., Bárcenas, E., Benítez-Guerrero, E.: Reasoning in context-aware systems with modal logics. Research in Computing Science 133, 51–61 (2017)
11. Limón, Y., Bárcenas, E., Benítez-Guerrero, E., Molero, G.: On the consistency of context-aware systems. Journal of Intelligent and Fuzzy Systems 34(5), 3373–3383 (2018)
12. Limón, Y., Bárcenas, E., Benítez-Guerrero, E., Molero-Castillo, G., Velázquez-Mena, A.: A satisfiability algorithm for the mu-calculus for trees with presburger constraints. In: 2019 7th International Conference in Software Engineering Research and Innovation (CONISOFT) (2019)
13. Limón, Y., Bárcenas, E., Benítez-Guerrero, E., Molero-Castillo, G., Velázquez-Mena, A.: Mu-calculus satisfiability with arithmetic constraints (in press). Programming and Computer Software, PCS (2020)
14. Limón, Y., Bárcenas, E., Benítez-Guerrero, E., Nieto, M.A.M.: Depth-first reasoning on trees. Computación y Sistemas 22(1), 189–201 (2018)